A REFERENCE ARCHITECTURE FOR LARGE SCALE DISTRIBUTED DOMAIN-DRIVEN BIG DATA SYSTEMS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisor
Dr Alan Litchfield
Dr Stephen Thorpe

30th June 2024

By
Pouya Ataei
School of Engineering, Computer and Mathematical Sciences

Abstract

Today, people are the ceaseless generators of structured, semi-structured, and unstructured data that, if gleaned and processed, can reveal game-changing patterns. Additionally, advancements in technology have made it easier and faster to collect and analyse this data. This has led to the age of big data. The age of big data began when the volume, variety, and velocity of data overwhelmed traditional systems.

Many businesses have attempted to harness the power of big data; nevertheless, the success rate is low. According to multiple surveys, only a 20% of big data projects are successful. This is due to the challenges of adopting big data, such as organisational culture, rapid technological change, system complexity, and data architecture. This thesis aims to address data architecture challenges of adopting big data by introducing a domain-driven, decentralised big data reference architecture.

This reference architecture is designed specifically to mitigate big data challenges by providing a scalable data architecture for big data systems, flexible and rapid data processing for varied velocity, adaptable management for a wide variety of data formats, maintainable approach for data discovery and aggregation, and increased attention to cross-cutting concerns such as metadata, privacy and security. This research uses design science research as the underlying research framework while utilising empirically grounded reference architecture guidelines for the development of the artefact. The evaluation of the artefact involves two distinct methods: a case-mechanism experiment and expert opinion, ensuring a comprehensive assessment of the big data reference

architecture.

The reference architecture's usefulness and effectiveness are supported by this process, which shows that it can handle volume, velocity, and variety of big data by processing data quickly, being scalable, and being able to adapt to different data formats. Additionally, the reference architecture's design mitigates the complexity of monolithic data pipelines, decentralises data ownership to avoid bottlenecks, and fosters a more integrated, agile approach to big data systems. This study positions itself as a progressive step in big data reference architectures, directly targeting and offering solutions to the existing shortcomings of big data architectures. It is aimed primarily at data architects and researchers seeking innovative approaches in big data system design and development, as well as practitioners looking to understand and apply the latest advancements in big data architectures.

Contents

List of Tables

List of Figures

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of candidate

Publications

1. Ataei, P. & Litchfield, A. (2020). Big data reference architectures: A systematic literature review. In 2020 31st Australasian Conference on Information Systems (ACIS) (pp. 1–11). doi: 10.5130/acis2020.bf

2. Ataei, P. & Litchfield, A. (2021b, Dec). Neomycelia: A software reference architecture for big data systems. In 2021 28th Asia-Pacific Software Engineering Conference (APSEC) (p. 452-462). Los Alamitos, CA, USA: IEEE Computer Society. Retrieved from https://doi.ieeecomputersociety.org/10.1109/APSEC53868.2021.00052 doi: 10.1109/APSEC53868.2021.00052

3. Ataei, P. & Litchfield, A. (2022). The state of big data reference architectures: A systematic literature review. IEEE Access.

4. Ataei, P. & Litchfield, A. (2023). Towards a domain-driven distributed reference architecture for big data systems. AMCIS 2023.

5. Ataei, P. & Staegemann, D. (2023). Application of microservices patterns to big data systems. Journal of Big Data, 10(1), 1–49.

6. Staegemann, D., Ataei, P., Lautenschläger, E., Pohl, M., Haertel, C., Daase, C., Volk, M., Abdallah, M.,& Turowski, K. (2024). An Overview on Testing Big Data Applications. In Proceedings of the Ninth International Congress on Information and Communication Technology. Springer, Lecture Notes in Networks and Systems.

7. Ataei, P., Regula, S., Haertel, C., & Staegemann, D. (2024). Impact of Big Data Analytics on Business Performance: A systematic Literature Review. AMCIS 2024.

8. Ataei, P. Neomycelia 2.0: A Domain-Driven, Decentralised Reference Architecture for Big Data Systems. 2024 IEEE International Conference on Big Data (In Review).

9. Ataei, P. (2024). Terramycelium: A Reference Architecture for Adaptive Big Data Systems. Information Systems (In Review).

10. Ataei, P. (2024). Cybermycelium: a Domain-Driven Distributed Reference Architecture for Big Data Systems. Frontiers in Big Data (In Review).

Acknowledgements

I would like to express my sincere gratitude to Dr. Alan Litchfield for his invaluable guidance and supervision throughout this research. His insights have been fundamental in shaping the direction of this study.
I am grateful to the experts who participated in the study and to all staff members at Auckland University of Technology (AUT) involved with PhD students, whose support and wisdom have been indispensable.
I extend my appreciation to fellow researchers, including Daniel Staegemann, for their collaboration and insights which significantly contributed to my academic development.
Additionally, I acknowledge the constructive feedback received from various journals and conferences, particularly on papers that were not accepted, as this feedback was crucial in refining my research.
Lastly I would like to thank Dr. Stephen Thorpe for reviewing parts of this thesis and providing valuable feedback.

Chapter 1

Introduction

1.1 Introduction

This study is situated within the domain of Information Systems Research, specifically addressing the architectural challenges in BigData (BD) systems. The research problem centers on the limitations of current BD Reference Architectures (RAs), particularly their struggles with system evolution, data governance, and architectural adaptability in rapidly changing technological landscapes. Through the application of Design Science Research (DSR), this study develops and evaluates Metamycelium, a novel domain-driven distributed RA that addresses the scalability, maintainability, and adaptability of BD systems while providing data governance frameworks. The research contributes both to the theoretical understanding of BD architectures and to practical implementation approaches in complex data environments.

This introductory chapter lays the foundation for the thesis, which addresses the development of a novel RA, Metamycelium, for BD systems. Metamycelium presents a means for capturing best practices and expressing them through architectural constructs. This artefact provides a solution to the limitations of current BD RAs, such as maintainability, scalability, data quality, and cross-cutting concerns, through the application of DSR methodology.

The chapter begins by providing essential context in Section 1.2, defining BD, and explaining its value across various sectors. It highlights how BD is conceptualised in this research and discusses the tangible benefits and applications of BD in different fields, including healthcare, energy exploration, and entertainment. The chapter also introduces the concept of RAs and their significance in the IT field, particularly in the development of complex systems like BD environments. It then explores the concepts of microservices and decentralised, distributed architectures, underscoring their relevance and benefits.

Following this, Section 1.3 presents an overview of the research, situating the study

within the context of the rapidly evolving BD landscape and the proliferation of BD technologies. It emphasises the need for advanced data management and processing systems in the BD era.

Section 1.4 then delves into the motivation, methods, and philosophy underpinning the research. This section examines the critical challenges in current BD architectures, including data ownership conflicts, scalability limitations, and integration complexities. It explores how these challenges impact system effectiveness and organizational success. The section then outlines the research methods employed, including systematic literature reviews (SLRs) and a narrative literature review, and discusses the philosophical approach that guides our investigation of these challenges.

Finally, the chapter concludes by presenting the primary research questions in Section 1.6 that were instigated in this thesis. These questions aim to address critical challenges in the field of BD systems and their architectural designs, setting the stage for the exploration and analysis in the subsequent chapters.

1.2 Context and Significance

This section provides foundational definitions essential for comprehending the nuances of the research. This section provides the conceptual framework necessary to understand the terminology used in the thesis.

1.2.1 What is Big Data?

To define BD within the scope of this research, various definitions from the body of knowledge have been examined. Kaisler, Armour, Espinosa and Money (2013) defines BD as “the amount of data that is beyond technology’s capability to store, manage, and process efficiently”. Srivastava (2018) state that BD pertains to “the use of large

data sets to handle the collection or reporting of data that serves various recipients in decision making".

Sagiroglu and Sinanc (2013) describe BD as "a term for massive data sets having a large, more varied, and complex structure with the difficulties of storing, analysing, and visualising for further processes or results".

Drawing from these definitions, BD in this research is conceptualised as "datasets characterized by high volume, velocity, and variety, which require advanced technologies and analytical methods for their transformation into value. These datasets typically exceed the processing capabilities of conventional relational database management systems (RDBMS) for managing and analyzing data. Big Data is further defined by its veracity, or the level of reliability and accuracy of the data, which poses significant challenges in data processing and analysis".

1.2.2 The Value of Big Data

The significance and value derived from BD remain pronounced (Ataei & Litchfield, 2022). Extensive discussions on the concept permeate reports, statistics, research, and conferences (H. Chen, Chiang & Storey, 2012). Notably, prominent companies like Google, Facebook, Netflix, and Amazon have propelled this momentum with substantial investments in BD initiatives (Rada 2017). A compelling illustration of the tangible benefits that BD offers can be seen in the Netflix Prise recommender system. This system capitalised on a diverse array of data sources, including user queries, ratings, search terms, and various demographic indicators (Amatriain, 2013). By implementing BD-powered recommendation algorithms, Netflix not only achieved a considerable increase in TV series consumption but also observed certain series experiencing up to a fourfold surge in viewership (Amatriain, 2013).

In a healthcare context, the Taiwanese government adeptly merged its national

health insurance database with customs and immigration datasets as part of a BD strategy (C. J. Wang, Ng & Brook, 2020). The resulting real-time alerts during clinical visits, informed by clinical symptoms and travel history, among other factors, facilitated the proactive identification of potential COVID-19 cases. Such strategic, data-driven initiatives significantly bolstered Taiwan's effectiveness in managing the epidemic.

In the realm of energy exploration, Shell harnesses BD to optimise the decision-making process and reduce exploration costs (Marr, 2016). By analyzing data from global drilling sites, companies can identify new locations with similar characteristics to known resource-rich areas. Prior to BD's integration, identifying energy resources presented formidable challenges. Traditional exploration methods, which relied heavily on deciphering waves of energy travelling through the earth's crust, were not only error-prone but also exorbitantly expensive and time-intensive.

Similarly, Rolls Royce capitalises on BD's potential by collecting intricate performance data from sensors fitted to its aircraft engines (Marr, 2016). Such data, transmitted wirelessly, provides insights into key operational phases, from take-off to maintenance. Leveraging this wealth of information, Rolls Royce can more accurately detect degradation, enhance diagnostic and prognostic accuracy, and effectively reduce false positives.

1.2.3 Reference Architectures

RAs have emerged as pivotal elements in contemporary system development, guiding the construction, maintenance, and evolution of increasingly complex systems (Cloutier 2010). RAs connect stakeholder needs and system implementation by linking context and requirements to system specification and design decisions (Muller, 2008). They guide the translation of business and operational requirements into architectural decisions that affect both functional capabilities and quality attributes. While system

components and their interactions are important aspects, RAs primarily enable architects to make decisions based on stakeholder needs, technical constraints, and quality requirements. This approach ensures architectural decisions align with stakeholder requirements and business objectives, while providing a framework for evaluating design trade-offs. This clarity fosters the creation of manageable modules, each addressing distinct aspects of complex problems, and provides a high-level platform for stakeholders to engage, contribute, and collaborate (Nakagawa, Oquendo & Maldonado, 2014).

Given the lack of a standard definition for “reference architecture", the definition from Clements, Garlan, Little, Nord and Stafford (2003), is taken, which describes it as a pre-established framework (an abstract blueprint) for designing software within a specific field. This framework, comprising structures, elements, and their relationships, serves as a template for creating concrete architectures (Pourmirza, Peters, Dijkman & Grefen, 2017).

1.2.4 Significance of Reference Architectures

As elucidated in Section 4.2, employing RAs for system conceptualisation enhances understanding of system components, behaviour, and evolution. This understanding is critical for influencing quality attributes like maintainability, scalability, and performance (Cloutier et al., 2010a). RAs function as standardisation artefacts and mediums of communication, offering stakeholders a common language to facilitate discussion and progression in BD projects.

Notably, the significance of RAs in IT is underscored by the success of widely adopted technologies like OAuth (OATH, 2007) and ANSI-SPARC architecture (ANSI, 1975), which have their origins in well-structured RAs. These RAs not only define the qualities of a system but also shape its evolution. While every system inherently possesses an architecture, RAs distinguish themselves by focusing on more abstract

qualities and higher levels of abstraction. They aim to capture the essence of practice and integrate well-established patterns into cohesive frameworks, encompassing elements, properties, and interrelationships.

The significance of RAs in BD is multifaceted, encompassing aspects like communication, complexity control, knowledge management, risk mitigation, fostering future architectural visions, defining common ground, enhancing understanding of BD systems, and facilitating further analysis.

1.2.5 Insights from Current Big Data Reference Architectures

The systematic analysis in Chapter 4 reveals insights about current BD RAs and their implementation challenges. The SLR, examining 79 studies from both academic and industrial sources, identifies several key findings regarding the current state of BD RAs.

Current BD implementations face significant architectural challenges. Many organizations develop BD systems using ad-hoc architectural approaches that diverge from established software engineering practices (Gorton & Klein, 2015). This leads to suboptimal architectural decisions and creates difficulties in system evolution. The SLR reveals that while RAs exist in the BD domain, they often mirror traditional data warehousing approaches, lacking the flexibility and scalability required for modern BD systems.

The analysis identifies several important limitations in current BD RAs. Notably, the NIST BD RA and Lambda architecture (S1), while providing comprehensive frameworks, show deficiencies in addressing essential elements such as metadata management, data quality controls, and privacy considerations. These limitations are particularly evident in areas of data governance, quality management, and system scalability.

The findings highlight a gap between theoretical architectural models and practical implementation needs. While BD RAs aim to provide standardized approaches, they

often lack concrete guidance for addressing common implementation challenges such as data quality management, system scalability, and cross-team collaboration. This gap is particularly evident in the context of evolving BD technologies and changing business requirements.

These insights from the SLR underscore the need for more mature and practical BD RAs that can effectively address current implementation challenges while providing flexible frameworks for future evolution. The findings suggest that future research should focus on developing RAs that better integrate modern architectural patterns, particularly in areas of microservices, event-driven architectures, and metadata management.

1.2.6 Reference Architecture's Role in Addressing Big Data Challenges

While recognising that RAs are not the exclusive solution for tackling BD challenges, they represent an effective approach due to several inherent advantages. It is important to note that alternative methods, such as bespoke system development and modular architecture approaches, also hold merit in certain contexts. However, the specific strengths of RAs in addressing BD challenges include:

1. **Standardisation and Scalability:** RAs offer a standardised framework that can be consistently applied across different projects, reducing the complexity associated with BD scalability. While bespoke solutions can be tailored to specific needs, RAs provide a more holistic and adaptable framework for managing large volumes of data.

2. **Integration of Diverse Data:** RAs are particularly effective in integrating various data types and sources, a critical need in BD systems. Other approaches, such as modular architectures, may also facilitate integration but often require more customisation and additional integration effort.

3. **Reducing Technological Complexity:** The structured nature of RAs can simplify the inherent technological complexities of BD systems. This simplification is often more challenging to achieve with approaches that lack a standardised or holistic framework.

4. **Incorporating Robust Security and Privacy Protocols:** RAs can embed security and privacy considerations into the architecture from the outset. While other methods can also integrate these aspects, RAs provide a comprehensive and uniform approach to ensure system reliability and address vulnerabilities.

Considering the range of artefacts available for tackling BD challenges, RAs stand out for their capacity to provide a comprehensive, standardised, and scalable framework. Their suitability for the complexities and diverse demands of BD systems is evident (Cloutier et al., 2010b). Nonetheless, the decision to utilise an RA should be guided by the specific requirements of the project, the organisational context, and the characteristics of the data at hand.

1.2.7 Microservices and Service Distribution Patterns

Microservices architecture, representing an evolution in software engineering, involves structuring applications as a collection of loosely coupled services (Bucchiarone et al., 2020). This approach, emerging from the broader concept of service-oriented architectures (SOA), focuses on developing small, independently deployable modules that collaborate to form a comprehensive application.

While microservices enhance scalability, facilitate continuous deployment, and foster a more agile development environment (S. Newman, 2015a), they also introduce significant complexities in service management, data consistency, and system monitoring. Teams must carefully weigh these trade-offs when adopting microservices architecture.

The architectural decisions around service distribution are nuanced and context-dependent, requiring careful consideration of when to centralize or distribute components, and to what degree. This architectural style, as highlighted by Richards (2015), requires balancing various factors including data consistency, network latency, and system reliability.

In modern systems, data and processing distribution decisions are made based on specific requirements and constraints, as discussed by Coulouris, Dollimore and Kindberg (2005). These decisions introduce trade-offs around the CAP theorem: systems must balance consistency, availability, and partition tolerance based on their specific needs and use cases.

The convergence of microservices within these architectures represents both an opportunity and a challenge in software engineering. While it enables systems that are modular and adaptable, it also requires careful consideration of service boundaries, inter-service communication patterns, and failure handling mechanisms. Organizations must evaluate whether the benefits of specific distribution patterns outweigh their operational overhead for each component and use case.

A key consideration in these architectures is their scalability. In the context of modern systems, scalability refers to a system's ability to handle increased load. There are two primary approaches to scaling, each appropriate in different contexts:

- **Horizontal Scaling**: Also known as "scaling out", this involves adding more machines to a resource pool. In microservices, this often means deploying more instances of a service. While this approach provides better fault tolerance and can handle larger loads, it introduces complexity in load balancing and data consistency.

- **Vertical Scaling**: Also known as "scaling up", this involves adding more power

(CPU, RAM) to an existing machine. For microservices, this might mean increasing the resources allocated to a service instance. This approach is simpler to manage but has physical limitations and potential single points of failure.

Microservices architecture facilitates both types of scaling, but teams must carefully consider their specific requirements, operational capabilities, and resource constraints when choosing between these approaches. The decision often involves balancing factors such as cost, complexity, reliability, and performance requirements. The key is not choosing between centralization and distribution as absolute approaches, but rather understanding when and why to apply each pattern based on concrete requirements and constraints.

1.3 Overview of the Research

This section entails a brief history of BD, its adoption, the challenges, the solutions to those challenges, and the contribution of this thesis. This section is important as it instills a general understand of current state of BD, and the elements surrounding it.

1.3.1 Big Data, Big Bang?

The growth of the internet and digital technologies has significantly increased global connectivity. The widespread adoption of smartphones, IoT devices, and high-speed networks has led to a substantial increase in data generation and transmission. Modern network infrastructures now commonly support data speeds of 1 Gbps or higher, while mobile devices and IoT sensors continuously collect and transmit data. This technological landscape has made data increasingly prevalent in various aspects of daily life and business operations. This ubiquity of data acts as a fundamental element, laying the groundwork for the intricate interplay between advanced technology and architectural

design. This connection, or nexus, is pivotal in understanding how technology and architecture co-evolve in the digital age (Ataei & Litchfield, 2021b).

According to Internetlivestates.com (Stats, 2019), in only one second, there are 9,878 tweets sent, 1,138 Instagram photos uploaded, 3,117,720 emails sent, 99,738 Google searches made, and 94,144 Youtube videos viewed. That is, if it took 5 seconds to read the preceding paragraph, during that time, 15,588,600 emails were sent.

1.3.2 The Challenge

Driven by the ambition to harness the power of this large amount of data, the term 'Big Data' was coined (Lycett, 2013). BD initially emerged to address the challenges associated with various characteristics of data, such as velocity, variety, volume, and variability (Rada, Ataeib, Khakbizc & Akbarzadehd, 2017a). BD engineering is the practice of extracting patterns, theories, and predictions from a large set of structured, semi-structured, and unstructured data for the purposes of business competitive advantage (Rada, Ataeib, Khakbizc & Akbarzadehd, 2017b; Huberty, 2015). BD represents an important innovation, marking the beginning of a new era in the data-oriented industry. However, BD is not a solution that can automatically resolve any business process challenge. While BD offers numerous opportunities, integrating such a complex and influential technology into the existing frameworks of organisations is a challenging task. While a lot of opportunities exist in BD, subsuming an emergent and rather high-impacting technology like BD into the current state of affairs in organisations is a daunting task.

According to a recent survey from technology review insights in partnership with Databricks (2021), only 13% of the organisations excel at delivering on their data strategy. Another survey by NewVantage (2021) indicated that only 24% of organisations have successfully gone data-driven. This survey also states that only 30% of

organisations have a well established strategy for their BD endeavour. In addition, surveys from Henke et al. (2016) and Nash (2015) further support these numbers, which illuminates the scarcity of successful BD systems implementations in the industry.

There are various challenges for the adoption of BD, including the complexity of data architecture, rapid technological changes, a lack of sufficiently skilled data engineers, and organisational cultural barriers to becoming data-driven (Rad & Ataei, 2017a; Singh, Lai, Vejvar & Cheng, 2019). Among these challenges, the complexity of data architecture is highlighted. A successful BD system can be built upon an effective data architecture, serving as a blueprint that affects data lifecycle management, guides data integration, controls data assets, and handles change (Reis & Housley, 2022).

Nevertheless, most BD systems are typically developed on-premise as ad-hoc solutions, primarily using traditional centralised data architectures using data warehouses or data lakes (Gorton & Klein, 2015; Nadal, Herrero, Romero, Abelló et al., 2017; Dehghani, 2022). Underlying these centralised architectures, as the data ecosystem grows and new technologies and data processing techniques are introduced, the software architect will have a harder time maintaining a solution that addresses the emergent requirements.

This can potentially create the basis for an immature architecture that is difficult to scale, challenging to maintain, and raises high entry barriers (Ataei & Litchfield, 2021b).

1.3.3 The Solution

Since the approach of ad-hoc design to BD system development may not provide the most impactful results, novel data architectures that are designed specifically for BD are required. To contribute to this goal, the notion of RAs is explored, and a distributed domain-driven software RA for BD systems is presented. This RA is called

Metamycelium. The main contributions of this study are threefold: 1) explicating design theories underlying current BD systems and their limitations; 2) explicating design theories that generate the artefact's constructs; and 3) the artefact itself. A detailed description of unique contributions of this study is depicted in Chapter 10, Section 10.5.

1.4 Motivation, Methods, and Philosophy

This section presents the motivations driving the research, the methods employed to navigate the research, and the philosophical underpinnings guiding the approach.

1.4.1 Motivation

The motivation for this research is derived from a critical examination of prevailing BD RAs, as detailed in a SLR (Chapter 4). This examination reveals a significant reliance on monolithic data pipeline architectures in current BD RAs, presenting constraints in scalability, adaptability to rapid technological changes, and effective integration of diverse data sources. Furthermore, challenges such as centralised data ownership models and inefficiencies in data management by specialised, yet isolated teams are recurrently identified.

In response, this study introduces Metamycelium, a proposed RA that shifts towards a decentralised, domain-driven approach. This architectural shift aims to enhance scalability and adaptability, contrasting with the limitations of traditional BD RAs. Metamycelium also focuses on implementing efficient data management practices and reinforcing security and privacy measures. The motivation behind this research is to propose a more dynamic and responsive RA for BD systems, contributing a novel perspective to the BD RA field and addressing some of the key gaps identified, thereby laying the groundwork for future advancements in this area.

1.4.2 Methods and Philosophy

This research is founded on a pragmatic philosophy, highlighting the significance of empirical evidence and experience in shaping knowledge. Pragmatism, valuing practical solutions, advocates for the adoption of methods, procedures, and techniques most aligned with the study's objectives and the phenomena being investigated (Cherryholmes, 1992).

Following this pragmatic approach, the study utilises the DSR framework, integrating theoretical knowledge with practical application. DSR is "the scientific study of artificial design, investigating both the process of designing and the designed artifact in context" (Wieringa, 2014). A generic DSR process is portrayed in Figure 1.1.

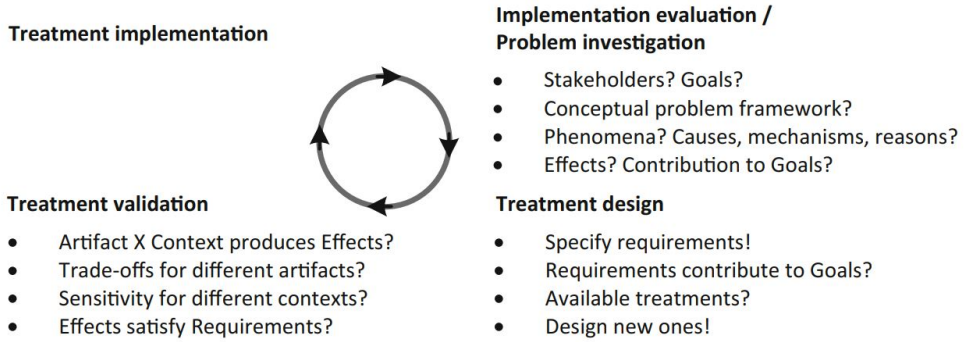


Figure 1.1: DSR process adapted from Wieringa (2014)

DSR is a problem-solving paradigm that seeks to create and evaluate IT artifacts intended to solve identified organizational problems. It involves a rigorous process of designing artifacts, such as constructs, models, methods, and instantiations, to address specific problems, and then evaluating these artifacts for their effectiveness (A. R. Hevner, March, Park & Ram, 2004).

DSR is particularly suitable for examining complex artefacts like the RA in BD systems. The framework transitions from overarching assumptions to specific methods of data collection, analysis, and interpretation, tailoring the research approach to the study's audience and the posed research questions.

Within the DSR framework, defining key elements like research approach, underlying philosophy, research design, and research methods is essential. This ensures a systematic transition from a broad problem in BD to detailed methodologies. The chosen research approach is qualitative, given its appropriateness for exploring emerging phenomena in the context of a novel domain-driven distributed architecture for BD systems. This approach enables inductive building from data, moving from particulars to general themes, and interpreting the data's relevance in the context of BD system development.

This DSR-driven research design encompasses several key stages:

1. Problem investigation: Identifying and analyzing challenges in current BD systems.

2. Treatment design: Developing the reference architecture as a solution to address identified problems. This stage involves:

   (a) Synthesizing insights from literature and practice

   (b) Formulating architectural principles and components

   (c) Ensuring theoretical robustness by grounding design decisions in established theories and best practices

3. Treatment implementation: Creating a prototype or detailed specification of the reference architecture.

4. Validation and evaluation: Assessing the effectiveness of the proposed solution through expert reviews and case studies.

Each stage is important for ensuring the artefact's alignment with practical needs in BD systems, its effectiveness, and its knowledge contribution.

In the problem investigation stage, SLRs and narrative literature review methods capture the current state of BD RAs, identifying existing gaps and trends. Treatment design integrates empirical data into the architectural design, ensuring that the RA is both theoretically robust and resonant with real-world BD practices and challenges. Subsequent stages, namely treatment implementation, validation, and evaluation, involve developing and evaluating a RA prototype through methods like expert opinions and case mechanism experiments. These methods emphasise the RA's practical relevance and applicability in the dynamic field of BD.

Adhering to the DSR framework, underpinned by a pragmatic philosophy, the study demonstrates a commitment to blending academic rigour with practical utility in BD systems development. It aims to bridge theoretical knowledge and practical application in BD, focusing on innovative design and effective evaluation to address practical problems in the field.

1.5 Emergence of Research Gaps and Direction of Study

The analysis of existing BD RAs (as presented in Chapter 4) illuminates several research gaps, particularly around identified failure modes in current architectural designs:

1. **System Evolution and Adaptation Constraints:** Current architectures exhibit significant limitations in:
    - Managing the proliferation of data sources and consumers
    - Adapting to evolving processing requirements (batch and stream)
    - Accommodating technological changes without major architectural disruption
    - Maintaining system performance and reliability as scale increases

Copy

2. **Data Management and Governance Limitations:** The analysis reveals persistent challenges in:
    - Maintaining data quality across growing volumes and varieties of data
    - Implementing comprehensive metadata management frameworks
    - Ensuring consistent security and privacy controls
    - Establishing clear data ownership and responsibility patterns
    - Managing data lineage and provenance effectively
3. **Architectural Rigidity:** Current approaches demonstrate:
    - Increased complexity in incorporating new data sources and technologies
    - Difficulties in maintaining system coherence during evolution
    - Challenges in supporting rapid experimentation and innovation
    - Limited ability to adapt to changing organizational requirements

To address these gaps, this research advocates the development of a new RA that embraces decentralised and domain-driven principles, aiming to enhance robustness in data management and adaptability.

1.6 Research Questions

The research questions for this thesis are informed by the failure modes and gaps identified in existing BD RAs. These questions are designed to confront the challenges in the field of BD systems and their architectural designs:

RQ1 **What are the limitations of current big data reference architectures?** This question aims to identify and analyse the key limitations in existing BD RAs, particularly in areas critical to their effectiveness, such as scalability, adaptability, and maintainability.

RQ2 **How can a reference architecture be designed to mitigate or address these limitations in big data projects?** This question builds on the insights from the first, focusing on conceptualising a new RA that effectively addresses the limitations and failure modes identified in current BD systems.

1.7 Thesis Structure

This thesis embarks on a detailed exploration of BD systems, emphasising RAs and microservice patterns. Each chapter contributes to a comprehensive understanding of these concepts.

- **Research Methodology:** Chapter 2 discusses the methodological underpinnings of the study, including research approaches, philosophical worldviews, designs, and methods, laying a solid foundation for the research.
- **Literature Review:** Chapter 3 sets the stage by establishing a foundational understanding of BD through a narrative literature review.
- **Big Data Reference Architectures:** Chapter 4 delves into an SLR on BD RAs. It evaluates the role, challenges, and limitations of current BD RAs, following a structured journey from the necessity of RAs to a discussion of their current state and implications.
- **Microservices Patterns in Big Data Systems:** In Chapter 5, an SLR focused on microservices patterns is presented, identifying academic gaps and discrepancies.

This chapter aligns with the broader research objectives of understanding the architectural complexities of BD systems.

- **Design of the Artefact:** The design and development of the artefact, Metamycelium, are thoroughly discussed in Chapter 6. It covers the artefact's requirements, the underlying theories, and the development process.
- **Evaluation of the Artefact - Part 1:** The first part of the artefact's evaluation, a single-case mechanism experiment, is detailed in Chapter 7. This chapter outlines the research design and presents the results of the prototype evaluation.
- **Evaluation of the Artefact - Part 2:** Chapter 8 continues the evaluation phase with expert review. It systematically analyses expert opinions and presents a comprehensive analysis of the findings.
- **Discussion:** The Chapter 9 integrates the research findings into a broader discussion. It synthesises key insights, analyses the results, and discusses the implications and generalisability of the study.
- **Conclusion:** Finally, the Chapter 10 concludes the study by providing a summary and discussing limitations, challenges, and future directions.
- **Appendix A: Expert Opinion Guide:** Detailed guide used for gathering expert opinions, as referenced in Appendix A.
- **Appendix B: Reference Architecture Classification:** A framework for classifying reference architectures, as outlined in Appendix B.

Each chapter is interconnected, collectively advancing our understanding of BD systems and architectures and guiding the reader through the research journey.

1.8 Conclusion

This chapter establishes the foundation for the research by outlining its objectives and methodologies and framing the context for BD exploration. The following chapters will delve into the research design, philosophies and methods that guide this study.

As the thesis progresses, each chapter builds upon this initial conceptual framework, exploring the complexities and challenges of BD systems and their architectural solutions. This chapter serves as a stepping stone into the exploration of BD, setting the stage for analyses and discussions in the subsequent chapters.

Chapter 2

Research Methodology

2.1 Introduction

Building upon the foundations set for the research in the previous chapter, wherein motivations, objectives, and philosophies are discussed, this chapter delves into the foundational principles that drive this research design and methodology.

All research is influenced by the assumptions made by the researcher about what constitutes valid research and the methods deemed appropriate for the specific problem. To ensure the rigour and quality of research, it is therefore imperative to first elucidate these underlying assumptions (Myers & Avison, 2002). In alignment with this perspective, this chapter aims to elaborate on the principal components of the current research: *research approaches*, *philosophical worldviews*, *research designs*, and *research methods*. Moreover, the central research questions are restated and further examined. Figure 2.1 portrays a high-level view of all the elements incorporated in the research methodology of this study.

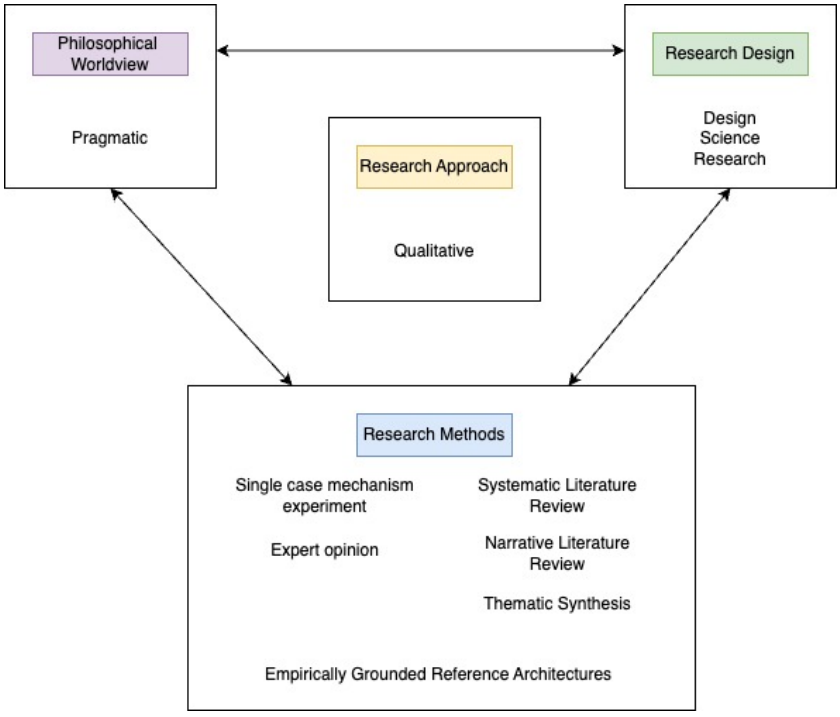


Figure 2.1: The high-level view of all research elements

Subsequent sections discuss the essence of good Information Systems (IS) Research (Section 2.4), illuminate the selection of the research approach, and shed light on the multifaceted plans influenced by the researcher's philosophical assumptions about the subject (Section 2.5). This discourse leads to the intricacies of the research design, exploring the architectural layout of the research (Section 2.6). The chapter concludes by summarising the aforementioned discussions, laying the groundwork for subsequent chapters.

2.2 What Makes a Good Information Systems Research?

Among the discernible trends in IS research, one often referred to as *traditional* has its roots in the positivist quantitative approach of natural science. While this approach is internally consistent and can yield significant results, there are instances where it might not lead to practical or definitive research outcomes (D. R. Vogel & Wetherbe, 1984). Creswell and Creswell (2017) suggests that many of these studies rely heavily on laboratory-based experiments or field surveys, prioritising statistical analysis. This approach has several noted challenges:

1. Research conducted in laboratory conditions provides a controlled environment, but it is crucial to understand that these settings might not always replicate the complexities of real-world scenarios. An example is that assessing the impact of a decision support system on a manager's decision-making might yield different insights in a genuine operational context compared to a lab (Myers & Avison, 2002).

2. When setting values for variables, there might be occasions where certain assumptions could misrepresent real-world complexities.

3. A focus is often not given to artefacts in these studies, possibly sidelining the applied significance of IS and its business relevance.

Considering these challenges, many researchers proactively highlight potential validity concerns in their studies (Myers & Avison, 2002). It is imperative to note that while achieving a flawless study is aspirational, the genuine value of IS research lies in how the garnered knowledge can be applied practically. There might be limited merit in research insights if they don't resonate with or are not applicable to real-world contexts.

IS, as a field, encompasses diverse disciplines such as software engineering, computer science, and social sciences. It is not merely a pursuit of pure scientific knowledge but also about its applied facets. If the conclusions drawn from IS research are not congruent with real-world applications, their relevance might be debated.

With the above considerations in mind, the methodology selected for this investigation is tailored to embrace the multifaceted nature of real-world scenarios, the subject's inherent depth, and the myriad of contextual influences.

2.3 Research questions

Based on the findings from Chapter 3, 4, and 5, and the concepts discussed in Chapter 1 and in the previous section, the research questions formulated for this thesis are as follows:

RQ1 What are the limitations of current big data reference architectures?

RQ2 How can a reference architecture be designed to mitigate or address these limitations in big data projects?

2.4 The Selection of the Research Approach

Research approaches serve as procedures and plans for research, guiding the progression from broad assumptions to detailed methods of data collection, analysis, and interpretation (Creswell & Creswell, 2017). Such plans are shaped by philosophical assumptions regarding the subject matter, the chosen research design (the procedure of enquiry), and specific methods for data collection, analysis, and interpretation. The research approach derives from the research problem, the study's audience, and the research questions.

For the extraction of information from a broad research problem into successive methods, four critical elements need definition: *research approach*, *underlying philosophy*, *research design*, and *research methods*.

According to Creswell and Creswell (2017), the three prevalent research approaches include a) quantitative, b) qualitative, and c) mixed methods. The choice among them depends on the philosophical assumptions (positivist, interpretive) held, the selected research strategies (qualitative grounded theory, quantitative surveys), and the methods chosen to execute these strategies (collecting data through observation for qualitative research or using instruments for quantitative research, which can also include textual data).

A frequent distinction between quantitative and qualitative research approaches lies in data collection and presentation methods. While quantitative approaches often employ numbers, they can also rely on textual data. On the other hand, qualitative research typically seeks to describe existing phenomena, although it can also explore emerging ones. Furthermore, quantitative research does not strictly rely on closed-ended questions based on a priori knowledge and an existing theory; it can also encompass open-ended inquiries. Mixed-method research combines elements of both qualitative and quantitative research, merging the results to shed light on significant societal issues (Myers & Avison, 2002; Sevilla, 1992).

The continuum of research approaches ranges from qualitative at one end to quantitative at the other, with mixed methods occupying an intermediary position (I. Newman, Benz & Ridenour, 1998). Historically, quantitative research has been dominant across various disciplines, not just the social sciences, throughout the 19th and mid-20th centuries. On the other hand, the appeal of qualitative research grew in the latter half of the 20th century (Myers & Avison, 2002).

For this thesis, a qualitative approach has been adopted, given its suitability for exploring emerging phenomena like a novel domain-driven distributed architecture for BD systems (Myers & Avison, 2002; Creswell & Creswell, 2017). Qualitative research allows for an in-depth understanding of the complexities and nuances involved in the development of BD systems and their associated software components. This approach is particularly valuable when investigating a new and evolving field, as it enables the researcher to delve into the intricacies of the subject matter and consider the contextual factors that influence the design and implementation of these systems.

Furthermore, the exploratory nature of qualitative research aligns well with the aim of this study, which is to inductively build knowledge from data and interpret the significance of the findings in relation to the broader context of BD systems development (Merriam & Grenier, 2019).

While a quantitative approach could provide valuable insights through the evaluation phase of the artefact, it may not fully capture the depth and breadth of understanding required to address the research problem and questions at hand. Similarly, a mixed-methods approach, although potentially beneficial, might not allow for the same level of focus and immersion in the qualitative aspects of the study. By adopting a primarily qualitative approach, this thesis aims to contribute to the understanding of domain-driven distributed architectures for BD systems in a meaningful way while still incorporating quantitative elements as needed to support and complement the qualitative findings.

2.5 The Underlying Philosophy

The underlying philosophy of research, despite being largely tacit, inextricably influences the overall momentum of the research (Slife, Williams & Williams, 1995). This research philosophy is built upon the foundational concepts of a researcher's worldview, which in turn, guides their approach to conducting research.

Numerous phrases are used interchangeably with the term worldview, such as ontologies, epistemologies (Crotty, 1998), paradigms (Lincoln, Lynham, Guba et al., 2011), and philosophies (Neuman, 2007). For the purpose of this study, *worldview* is defined as the overarching philosophical framework that underpins the research.

Following the classic classification of research epistemologies provided by Chua (1986) and more modern works of Phillips, Phillips and Burbules (2000), Crotty (1998), Creswell, Hanson, Clark Plano and Morales (2007), Lincoln et al. (2011), Mertens (2008), Creswell and Creswell (2017), Myers and Avison (2002), and Neuman (2007), five philosophies frequently discussed in the literature are highlighted: 1) positivism 2) post-positivism 3) constructivism 4) transformative, and 5) pragmatism.

A detailed description of each philosophy and its drawbacks remains outside the scope of this study. However, a brief description focuses on the philosophy selected for this study and its justification. Positivists perceive reality objectively, with measurable properties to be described; thus, the observer does not influence the observed (Myers & Avison, 2002). Positivists aim to test a theory, predicting potential outcomes. IS research is perceived as positivist when evidence exists for quantifiable measures of variables, drawing from samples of the stated population, and if formal propositions are present (Orlikowski & Baroudi, 1991). This represents one of the oldest research philosophies and aligns more with quantitative research.

Following positivism, a new school of thought, termed *postpositivism*, emerges that challenges the traditional philosophy of positivism and the notion that knowledge is

absolute (Phillips et al., 2000). Postpositivism holds a key assumption that knowledge is conjectural, suggesting a lack of absolute certainty about claims (J. K. Smith, 1983) (Phillips et al., 2000). This philosophy centres on determinism and theory verification.

Constructivism, often merged with interpretivism, is viewed as a suitable philosophy for qualitative research and derives mainly from the works of Berger, Luckmann et al. (1966) and Lincoln and Guba (1985). Contrasting the two previously discussed philosophies, constructivism adopts the stance that reality is perceived subjectively. A central constructivist premise is that individuals form their subjective reality based on internalised meanings, and subjective meaning is derived from cultural and historical norms inherent in the individual's context. Constructivism does not begin with a theory but aims to inductively formulate one.

A philosophy that emerged during the 1980s and 1990s is the transformative philosophy (Creswell & Creswell, 2017). This approach concentrates on marginalised individuals in society. A focal point of this philosophy concerns the ways marginalised groups experience oppression and the strategies that challenge and overturn these constraints (Mertens, 2019). Mixed-method research approaches, such as exploratory convergent mixed-method research, are typically adopted.

Lastly, there is the pragmatic philosophy. Originating from the works of Peirce (1878), who first introduced the term and laid the groundwork for the pragmatic maxim. This philosophy was further developed by James (1981) and Dewey et al. (1917). Pragmatism emphasises the application of concepts and how action addresses problems (Patton, 2002). Other contributors to the development of pragmatism include Cook (1993), Ochs (1998), and, more recently, Talisse and Aikin (2011). This worldview does not strictly adhere to a singular system of philosophy or reality. Instead, it emphasises the practical consequences of ideas and the importance of context in shaping our understanding. Contemporary pragmatic approaches to research freely incorporate both qualitative and quantitative assumptions to address research inquiries effectively.

Underlying this philosophy, there exists freedom to select methods, procedures, and techniques that align most closely with the objectives and needs of the phenomena under investigation (Cherryholmes, 1992). Pragmatism encompasses belief in both the external world and the internal cognitive realm and tends to sidestep questions regarding the nature of reality and the laws of nature (Rorty, 1990). This philosophy facilitates the exploration of multiple avenues, diverse assumptions, and varied worldviews, coupled with different data collection and analysis methods.

Of all the philosophies described, pragmatism was selected for this study. Its selection is attributed to its problem-centred, pluralistic, and real-world practice-oriented nature (Phillips et al., 2000). This aligns with the research objective of creating a tool that addresses a real-world challenge or at least advances the field. Unlike positivism, which is inclined to view things through determinism and reductionism lenses, pragmatism seeks the most effective means to achieve optimal outcomes.

Constructivism and interpretivism serve as other potential choices for this study due to their tendencies to explore behavioural patterns with open-ended questions. However, the emphasis of pragmatism on real-world, practice-oriented issues gives it the edge. The transformative philosophy remains unchosen as this study does not explore power dynamics and justice.

2.6 Research Design

After choosing a suitable research approach and research philosophy, it is time to decide on the research design. A research design is the research inquiry embodied within quantitative, qualitative, and mixed methods approaches that guide the researcher through specific procedures (Creswell & Creswell, 2017). The research approach can be perceived as the macro high-level approach to research, whereas the research design focuses on the micro-level details of how to conduct certain research. Some researchers

refer to this as a strategy of inquiry (Denzin & Lincoln, 2011).

Over the years, research designs have evolved due to the emergence of complex applications for data gathering and analysis. These advancements empower researchers to analyse complex models and articulate more efficient procedures for conducting research. As the research approach of this study is qualitative, the research designs appropriate for this approach are considered.

According to Creswell et al. (2007), the five most common approaches to qualitative research design are identified: 1) narrative research; 2) phenomenological research; 3) grounded theory; 4) ethnography; and 5) case study.

The primary genesis of qualitative research originates from humanities, sociology, and anthropology (Creswell & Creswell, 2017). Qualitative research designs have a rich history, with roots in the early 20th century and notable developments such as symbolic interactionism emerging in the 1950s (Denzin, 2004).

Clandinin and Connelly (2004) paints a picture of what constitutes a narrative research design; Moustakas (1994) discusses the fundamental premises of phenomenology and the relative procedures; Strauss and Corbin (1998) describes the philosophical tenets of grounded theory and its procedures; Fetterman (2019) and Wolcott (2008) suggest the procedures involved in ethnography research; and R. K. Yin (2009) and Stake (1995) shed light on effective case study research.

This study aims to create an artefact that addresses a real-world problem; therefore, many of the aforementioned research designs may not be suitable. For instance, ethnography and phenomenological designs are unrelated; narrative research is not suitable because the focus is on an artefact in an organisation and not on the lives of individuals; and grounded theory is not appropriate because the aim is not to derive a general abstract theory grounded in the views of participants. Additionally, while a case study could be a viable research design, it does not emphasise the artefact and design aspects of the research.

Based on these premises and after further investigation of research methodologies on popular platforms such as Sage Research Methods and MIS Quarterly, Design Science Research (DSR) presented by Wieringa (2014) is chosen as the most suitable approach for this study. DSR aligns well with the research objectives of creating an artefact (a RA) to address real-world problems in BD systems. Its focus on designing and evaluating innovative solutions fits perfectly with the goal of mitigating failure modes in BD projects (A. R. Hevner et al., 2004; March & Smith, 1995).

The choice of DSR also resonates with the pragmatist philosophy adopted in this study. Pragmatism emphasises the practical consequences of ideas and the importance of problem-solving, which is at the core of DSR (Peffers, Tuunanen, Rothenberger & Chatterjee, 2007). It advocates for the use of multiple methods and the integration of different perspectives, which DSR accommodates by incorporating various research methods and considering the viewpoints of different stakeholders (A. R. Hevner et al., 2004).

Key factors considered for choosing DSR included alignment with research objectives, the ability to address real-world problems, the integration of theory and practice, and compatibility with the pragmatist philosophy. Based on these criteria, DSR emerged as the most appropriate choice for this study.

2.7 Design Science Research

One of the main objectives of IS is to improve organisational efficiency and provide a competitive advantage (Rainer, Prince, Sánchez-Rodríguez, Splettstoesser-Hogeterp & Ebrahimi, 2020). Achieving these objectives relies on the capabilities of the organisation, its culture, its workflow engines, and its methodologies (Silver, Markus & Beath, 1995). Good IS research thus aims to advance knowledge that results in effective and cutting-edge artefacts that facilitate system development.

Advancing the field of IS involves the integration of two complementary scientific paradigms: design science and behavioural science (March & Smith, 1995). Behavioural science, originating from the natural sciences, revolves around developing and justifying theories to predict or explain the human aspect involved in the design, development, and management of IS (A. Hevner & Chatterjee, 2010).

The direct value of theories and knowledge generated from behavioural research in IS can lead to increased efficiency and effectiveness of IS in organisations, which in turn informs decisions regarding the design and development of new IS.

These insights inform decisions regarding the design and development of new IS, complementing the design science paradigm, which is rooted in the sciences of artificial and engineering disciplines (Simon, 1969). Design science aims to solve real-world problems. It explores innovative practices, ideas, and artefacts that result in effective and efficient design, implementation, analysis, and maintenance of IS (Denning, 1997; Weiser, Brown, Denning & Metcalfe, 1998).

Over the years, IS literature has emphasised the importance of design (A. R. Hevner et al., 2004). In fact, the relevance of IS research can be measured by its applicability to practice, the synthesis of the existing body of knowledge, or how it stimulates critical thinking among IS practitioners (Benbasat & Zmud, 1999). Creating innovative artefacts to advance an emerging domain is complicated, but the result can extend the boundaries of organisational capabilities. Therefore, IS research presents a great opportunity to make significant contributions to problems in practice.

Pragmatists' philosophy argues that utility (artefact) and truth (theory) complement each other and that science IS research should be evaluated in light of its practical implications (A. Lee, 2000; Aboulafia, 1991). IT artefacts can be models, constructs (symbols or vocabulary), methods (practices and algorithms), and instantiations. These artefacts are concrete blocks of solutions, prescriptions, and suggestions that illuminate

problems inherent in the development of IS and address them to allow successful implementation of these systems within organisations (March & Smith, 1995; Nunamaker Jr, Briggs, Mittleman, Vogel & Pierre, 1996).

Markus, Majchrzak and Gasser (2002a) and Walls, Widmeyer and El Sawy (1992) describe DSR as an endeavour aimed at developing IS to support emerging knowledge processes, embodied in IS design theories.

These theories are prescriptive means that facilitate effective system development for a class of problems and are usually evaluated concerning their utility in a real-world context (Markus, Majchrzak & Gasser, 2002b).

Therefore, based on the premises above, the purposes of this study and the inherent properties of design science align, which are to develop and evaluate an IT artefact designed to solve an identified organisational problem.

To attain a clear understanding of this research design, one first has to accept a dichotomy; that is, design is both the product (artefact) and the processes (a set of activities). Holding this dual perspective, the researcher constantly switches between the processes involved in the design and the artefact for the same complex problem (Walls et al., 1992). This process continues until the researcher reaches a point where further iterations yield diminishing returns in terms of improving the artefact or the design process.

Once this point is reached, the evaluation of the artefact begins, going through several iterations to enhance the quality of both the product and the design process. Throughout these stages, both the artefact and the design process evolve. In the context of this study, the theory and design are considered to be axiomatically connected, meaning that the theory inherently informs the design, and the design, in turn, shapes the development of the theory.

Within the IS discipline, DSR is applied to a wide range of problems, including well-defined, structured problems as well as more complex, ill-structured ones (A. R. Hevner

et al., 2004; Peffers et al., 2007). DSR is particularly well-suited for addressing sophisticated or wicked problems (Brooks Jr, 1996; Rittel, 1984), which are characterised by:

- Incomplete, contradictory, and changing requirements
- Difficulties in recognising the problems and their solutions
- Complex interactions among subcomponents of the problem and its solution
- The inherent uniqueness and novelty of the problem and its solution
- Heavy dependence on human cognitive abilities to produce effective solutions

The development and design of Metamycelium, a distributed and decentralised software RA discussed in Chapter 6, incorporates several complexities that align with the characteristics of wicked problems. Firstly, Metamycelium entails a complex ecosystem of components, sub-components, and their interrelations, as described in Sections 6.4.1 and 6.4.2. Secondly, it has rapidly changing or unstable properties and constraints based on the environmental context, as discussed in Sections 6.4.5 and 6.4.6. Finally, it depends on human creative abilities to produce effective solutions, as highlighted in Sections 6.4.3 and 6.4.4.

The presence of these complexities in the development and design of Metamycelium further justifies DSR as the appropriate research design for this study, as DSR is well-equipped to tackle the challenges associated with wicked problems. By employing DSR, this study aims to address the intricacies and evolving requirements inherent in the design of a distributed and decentralised software RA like Metamycelium.

2.8 Research Goal

DSR can be used for a variety of research goals, and the goal is the deciding factor in choosing the right research design. Wieringa (2014) defines 4 major research goals, which are artefact (re)design goals, prediction goals, knowledge goals, and instrument design goals.

The main goal of this study is to improve the performance, scalability, maintainability, and effectiveness of an artefact for a class of problems. Therefore, this research falls under the *design goals* category. Design goals refer to the specific objectives or criteria that a design solution must meet to be considered successful. These objectives can vary greatly depending on the context, industry, or application of the artefact. For example, the design goals of a product may be focused on improving its usability and marketability, while the design goals of a software application may prioritise its performance and reliability.

Because design goals have different objectives, they typically require a unique design cycle that is tailored to their specific needs. This design cycle may involve different stages or processes, such as user research, prototyping, testing, and evaluation, depending on the goals of the design. By going through this cycle, designers can refine and iterate their design solutions to better meet the specific goals and needs of their users or stakeholders.

2.9 Design Cycle

Simon (1981) is a pioneering figure in the field of design science, having significantly contributed to its development by adapting and formalising principles from engineering and applying them to the study of artificial systems. His work laid the groundwork for the further evolution of design science as a discipline. He is a prominent scholar,

researcher, and thinker who has made significant contributions to a variety of fields, including economics, psychology, and computer science. In his work on design science, Simon emphasised the importance of a design cycle that incorporates iterative processes of problem-solving, prototyping, and testing. He believed that design was not a linear process but rather a continuous cycle of refinement and improvement. Simon also stresses the importance of incorporating user feedback and evaluating the success of design solutions against specific criteria or objectives.

Furthermore, Nunamaker Jr, Chen, Purdin, Sprague Jr and Takeda (2009) and Takeda (2014), among other researchers in the field of DSR, build upon Herbert Simon's ideas on the iterative design cycle and the importance of evaluating design solutions against specific objectives or design goals. Peffers et al. (2007) work serves as another example of how Simon's ideas on the design cycle have influenced and shaped the field of DSR and how other researchers have built upon and extended these ideas over time.

The common denominator of all these seminal works is the *design cycle*. For the purposes of this thesis and inspired by the aforementioned works, the design cycle of this study follows the guidelines provided by Wieringa (2014). This cycle is made up of three major phases: problem investigation, treatment design, and treatment validation. Wieringa (2014) defines the design cycle as a subset of the engineering cycle, and the engineering cycle is referred to as a 'rational problem-solving process'. The engineering cycle of this thesis is portrayed in Figure 2.2 and is made up of the following steps:

1. Stakeholder and goal analysis: What is the goal? Who is affected by it?
2. Problem investigation: What phenomena are of interest?
3. Requirement specification: a clear understanding of what is expected of the artefact.
4. Treatment design: actual design of an artefact for the problem context.

5. Treatment validation: Does the design treat the problem?

6. Treatment implementation and evaluation: treating a problem with the artefact and measuring the success of the artefact in addressing the problem. The result of this step can result in an iteration.

The conceptual framework used to define the elements of the engineering cycle is defined by Wieringa (2014) and is further explored in the following subsections.

2.9.1 Step 1: Stakeholder and Goal Analysis:

The initial point of this design cycle is the *stakeholder and goal analysis*. This step is pivotal, as it sets the trajectory for the remainder of the study and highlights integral elements. Two of these elements are stakeholders and goals. In accordance with A. R. Hevner et al. (2004) methodology, the DSR process commences with the identification of a problem or opportunity that can be addressed through the creation of an artefact, such as a system, process, or algorithm.

This step first involves specifying the objectives of the research and, second, identifying the potential stakeholders who will utilise or benefit from the artefact. Discussion of stakeholders and goals follows in the subsequent sub-sections.

Goals: In DSR, project goals are determined by the specific requirements of the research undertaking. These goals can be user-centred, object-centred, machine-centred, or focused on any other relevant aspect, depending on the research context and objectives. While some DSR projects may prioritise stakeholder needs, others might focus on addressing research gaps or answering specific research questions (Peffers et al., 2007; A. Hevner & Chatterjee, 2010). It is important to recognise that there is no single DSR methodology; instead, DSR encompasses a range of methodologies and variations that researchers adapt to their particular research settings and aims.

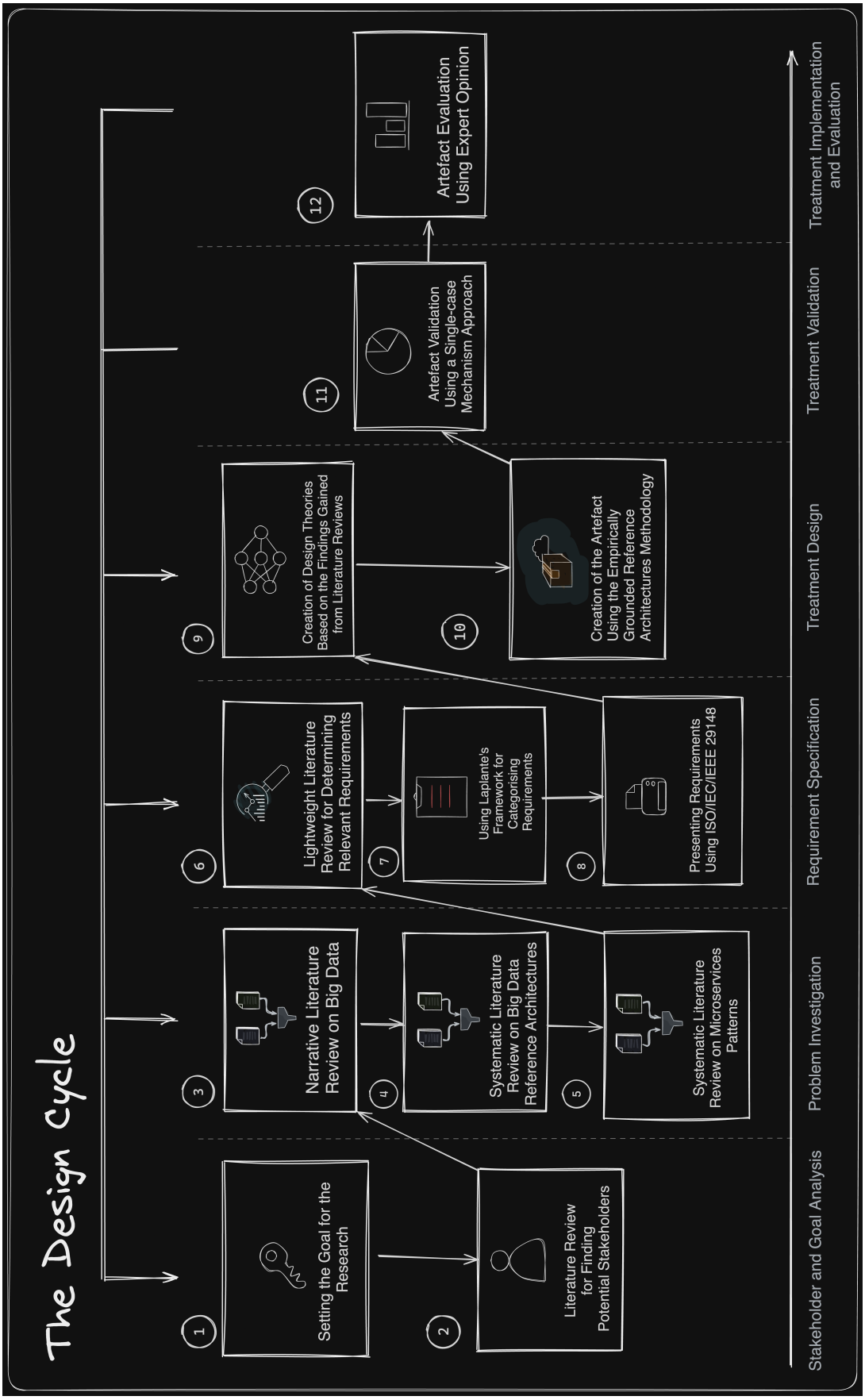


Figure 2.2: The design cycle

Here, the goals of the DSR project are moulded by the research objectives and questions (see Section 2.3), which might not necessarily align with a particular stakeholder group's needs. Hence, while stakeholder needs might be considered during the development of this DSR project, the primary driver remains the research questions and the potential academic contributions from the design and evaluation of a new artefact (Peffers et al., 2007; March & Smith, 1995).

Stakeholders: According to A. R. Hevner, Chatterjee and Galletta (2010), stakeholders in DSR are individuals or groups who can influence or be affected by the design, development, implementation, or use of the artefact being created. Stakeholders can comprise users, customers, managers, technical experts, regulatory bodies, and other relevant parties who have an interest in the artefact's success.

This research represents an exploratory DSR, not funded by any organisation or governmental institution, but is an academic study conducted to generate valuable knowledge and further advance practice. While DSR often involves the creation of an artefact to address a practical problem or seize an opportunity, it is not always necessary to have a clear group of stakeholders identified at the outset. In such instances, considering potential stakeholders who could benefit from the artefact and defining them as precisely as possible becomes essential. The objective, then, is not to provide a definitive set of stakeholders but to highlight the potential to impact certain established stakeholders in practice.

Stakeholder lists and their role descriptions vary between companies. Some roles are defined specifically in one organisation, while others are more established roles that organisations adopt, for instance, a business analyst. To find the relevant roles across the industry, industrial reports and surveys are used. The primary surveys and reports that informed these roles are the Stackoverflow developer survey (Overflow, 2022),

ThoughtWorks Technology Radar (ThoughtWorks, 2021), Linkedin emerging jobs report (LinkedIn, 2020), Gartner Magic Quadrant and Critical Capabilities (Gartner, 2021) and Australian and New Zealand Standard Classification of Occupations (Australian Bureau of Statistics, 2021). These sources are relevant for understanding the roles and trends in the technology industry, as they are based on large-scale surveys, expert analyses, and standardised classifications from reputable organisations that closely monitor and report on the ever-evolving landscape of technology professions. The results of the stakeholder analysis based on these reports and surveys are discussed in Section 6.2 of Chapter 6.

2.9.2 Step 2: Problem Investigation

According to the DSR methodology presented in Wieringa (2014), problem investigation entails the identification and comprehension of the problem the research seeks to address. This often involves a literature review to ascertain the prevailing state of knowledge and the solutions associated with the problem. Stakeholder interviews can also be employed to procure a deeper understanding of the problem and potential solutions and to discern any contextual factors pertinent to the design and evaluation of a solution.

The problem investigation phase is indispensable in ascertaining that the research remains concentrated and germane to the problem in question, ensuring that the resulting artefact or solution aligns with practical needs (A. R. Hevner et al., 2004).

A problem investigation is conducted to discern the research gap and delve into the contemporary state of the art in the domains of BD, BD architectures, and distributed systems, including microservices architecture. Two SLRs are conducted in this regard. The first SLR (see Chapter 4) provides an in-depth exploration of BD RAs, analysing their current state, identifying failure modes, and evaluating their strengths and weaknesses. This SLR directly contributes to the problem investigation phase by pinpointing

the limitations and challenges of existing BD RAs.

The second SLR (see Chapter 5), while supplementary in nature, focuses on microservice patterns, studying their advantages and challenges. This supplementary review aids in understanding the potential of microservice architecture in addressing the identified issues, thereby supporting the problem investigation phase and informing the subsequent design of the artefact.

While both SLRs are nuanced, they are anchored in the PRISMA guidelines (Shamseer et al., 2015). Additionally, PRISMA-S (Rethlefsen et al., 2021) was incorporated to refine the search strategy, and guidelines from Kitchenham, Budgen and Brereton (2015) tailored for evidence-based software engineering and systematic reviews were employed.

Given that PRISMA primarily originates from the healthcare domain, some of its principles may not be directly applicable to the fields of software engineering and IS. Hence, Kitchenham et al. (2015) translated many of PRISMA's tenets to cater to the nuances of software engineering, offering guidance especially beneficial for research projects with constrained resources. The aforementioned SLRs lean on PRISMA for their foundational research design, amalgamating it with auxiliary research methods to mitigate bias, augment transparency, and bolster systematicity.

The choice of SLR is driven by its qualitative nature, which facilitates the enhancement of knowledge and insight around nascent topics and their associated facets. Moreover, an SLR offers a transparent and reproducible modus operandi that surfaces patterns, correlations, and trajectories, painting a holistic picture of the subject matter (Borrego, Foster & Froyd, 2014).

The systematic reviews are orchestrated in four distinct phases. Initially, research questions are posed, and criteria for inclusion and exclusion are set. Literature is then amassed, and a quality framework is formulated. Subsequently, studies are evaluated against the quality framework. Each study is meticulously analysed against the criteria

delineated in the quality framework.

In the penultimate phase, the collated literature is coded in alignment with the research questions using a combination of deductive and inductive coding approaches (see Sections 4.5.3 and 5.2.1 for detailed coding procedures). The coding process involves identifying initial codes based on the research questions and iteratively refining them as new themes emerge from the data. The codes are then analysed and synthesised to develop higher-order themes and models that address the research questions. The inclusion and exclusion criteria for the coding process are consistent across both SLRs, as described in Sections 4.5.2 and 5.2.1. The final phase involves the synthesis of findings through thematic synthesis, as proposed by Cruzes and Dyba (2011), leading to the emergence of distinct themes.

Complementing the SLRs, a narrative review of literature on BD is conducted (see Section 3), adhering to the guidelines set forth by Clandinin and Connelly (2004). This review accentuates the pervasive significance of BD in the contemporary digital milieu. Through these extensive reviews, gaps in existing research are discerned, the state of the art is probed, and a comprehensive grasp of the domain's extant knowledge is acquired, thereby laying a foundation for the research. The nuances and intricacies of these reviews are further expounded upon in Chapters 3, 4, 5.

2.9.3 Step 3: Requirement Specification

In Wieringa (2014) DSR methodology, the requirement specification phase is an essential step in developing a new IS or artefact. This phase involves identifying the requirements that the artefact must satisfy to meet the needs of stakeholders and achieve the desired outcomes.

Wieringa (2014) methodology distinguishes between functional requirements, which describe what the artefact should do, and non-functional requirements, which describe

how the artefact should do it. Functional requirements are typically expressed as use cases, which describe the specific interactions between users and the system. Non-functional requirements may include performance requirements, security requirements, and usability requirements.

The requirement specification designed for this thesis is made up of the following phases:

1. Determining the type of requirements
2. Determining the relevant requirements
3. Identifying the right approach for categorising the requirements
4. Identifying the right approach for the presentation of the requirements

Determining the Type of Requirements:

Defining and classifying software and system requirements is a common subject of debate. Sommerville (2011) classifies requirements as three levels of abstraction; user requirements, system requirements, and design specifications. These abstractions are then mapped against user acceptance testing, integration testing, and unit testing. However, the focus of this study is to define high-level requirements for BD systems, which necessitates a more comprehensive and flexible approach. Nevertheless, in this study, a more general framework provided by Laplante (2017) is adopted. The adopted approach provides three types of requirements: functional, non-functional, and domain requirements. The objective of this step is to define the high-level requirements of BD systems; therefore, the main focus is on non-functional and domain requirements.

Determining the Relevant Requirements:

By analysing the results of the SLR for RAs conducted in Chapter 4, several studies have been identified that deeply explore the relevant requirements for a BD RA from the perspective of the RA's development methodology. These studies provide valuable insights into the types of requirements that are essential for developing a BD RA.

In an extensive effort, the NIST BD Public Working Group embarked on a large-scale study to extract requirements from a variety of application domains such as healthcare, life sciences, commercial, energy, government, and defence (W. L. Chang & Boyd, 2018). The result of this study is the formation of general requirements into seven categories.

Additionally, Volk, Staegemann, Trifonova, Bosse and Turowski (2020) categorises nine use cases of BD projects sourced from published literature using a hierarchical clustering algorithm. Furthermore, Bashari Rad, Akbarzadeh, Ataei and Khakbiz (2016) focuses specifically on security and privacy requirements for BD systems, providing a comprehensive analysis of these critical aspects from the RA development perspective.

J.-H. Yu and Zhou (2019) present modern components of BD systems using goal-oriented approaches, offering insights into the functional requirements of BD architectures and their impact on the RA design process. Eridaputra, Hendradjaya and Sunindyo (2014) created a generic model for BD requirements, which serves as a foundation for understanding the overall requirements landscape in the context of RA development.

Lastly, Al-Jaroodi and Mohamed (2016) investigates general requirements to support BD software development, contributing to the understanding of the necessary features and characteristics of BD systems that should be considered when developing an RA.

By examining these RAs and their development methodologies, and by evaluating the design and requirement engineering processes specific to BD RAs, a set of high-level requirements based on BD characteristics is established. This approach ensures that the

requirements are grounded in the current state of knowledge in the field and are tailored to the unique needs of BD RAs.

The resulting set of requirements encompasses the essential aspects of BD systems, data volume, velocity, variety, veracity, and value, as well as critical considerations such as security, privacy, metadata and performance. These requirements form the basis for the design and development of the Metamycelium RA, ensuring that it addresses the key challenges and requirements identified in the literature and aligns with the best practices in RA development methodologies.

Identifying the categorisation approach for requirements:

After clarifying the type of requirements and the relevant requirements, current BD RAs and their requirements are assessed to gain insights into the available BD requirement categorisation methods. The analysis of these studies reveals a common approach to categorising requirements based on BD characteristics such as velocity, veracity, volume, variety, value, security, and privacy (Ataei & Litchfield, 2022; Bahrami & Singhal, 2015; Rad & Ataei, 2017b; H.-M. Chen, Kazman & Haziyev, 2016a).

The V's of BD (volume, velocity, variety, veracity, and value) provide a multi-dimensional framework for categorising requirements. Each characteristic represents a distinct aspect of BD systems, and by aligning requirements with these characteristics, a comprehensive and structured approach to requirement categorisation is achieved. This categorisation method enables a holistic view of the requirements landscape, ensuring that all critical aspects of BD systems are considered in the development of the Metamycelium RA.

Identifying the right approach for the presentation of the requirements:

To represent the functional requirements, the guidelines provided in the ISO/IEC/IEEE 29148 standard (ISO/IEC, 2018) are adapted. This standard outlines the key elements

that should be included in a requirement specification, such as the requirement ID, description, and priority. By following these guidelines, the functional requirements are documented in a clear and consistent manner, facilitating their understanding and implementation in the RA development process.

The requirements representation is organised in system characteristics, where the major components of the system and their associated requirements are described. This approach is inspired by the requirement specification structure used in the NASA Wide-field InfraRed Explorer (WIRE) system (Laplante, 2017) and the Software Engineering Body of Knowledge (SEBoK) (Abran, Moore, Bourque, Dupuis & Tripp, 2004). By adopting this structure, the requirements are presented in a logical and hierarchical manner, aligned with the system's architecture and functionality.

2.9.4 Step 4: Treatment Design

The *Design* phase is a crucial step in the methodology, occurring after the problem identification, motivation, and objective definition stages. This phase involves two integral components: theory and artefact.

The terms *treatment* and *artefact* are used in a specific manner, drawing inspiration from Wieringa's design science research methodology (Wieringa, 2014). *Treatment* refers to the major steps in the design cycle, such as treatment design, treatment validation, and treatment evaluation, whereas *artefact*, alongside theory, refers to the actual created artifact that emerges from the treatment design phase.

Theory

The first output activity of this phase involves the creation of a design theory, which represents the foundational knowledge behind the artefact that is being developed. The design theory encompasses the requirements, constraints, and best practices that are

pertinent to the artefact's design.

Design theory can be defined as a formal or informal framework that provides a structured and systematic approach to designing artefacts (Fischer, Winter & Wortmann, 2010). It offers a set of guidelines, principles, and best practices that can help researchers and practitioners design artefacts that are effective, efficient, and meet the specified requirements and constraints.

The elements that underpin the creation of these theories are learned from the SLRs discussed in Chapter 4 and Chapter 5; the RAs analysed from Chapter 4, the patterns gleaned from Chapter 5, requirements specification from Section 2.9.3, and best industrial and academic practices for designing distributed and decentralised BD systems (see Section 6.4).

The design theory is employed to steer the development of the artefact throughout the remaining phases of the methodology.

Artefact

The second part of the treatment design phase involves designing a solution or artefact to address the identified problem or opportunity (see Section 6.5). The goal of this phase is to create a design that meets the identified requirements and can be implemented and tested.

As discussed in Section 4.6.3, there are several approaches to the systematic development of RAs. Cloutier et al. (2010b) demonstrate a high-level model for the development of RAs through the collection of contemporary architectural patterns and advancements. Bayer et al. (1999) introduces a method for the creation of RAs for product line development called PuLSE DSSA.

Stricker et al. (2010a) present the idea of a pattern-based RA for service-based systems and use patterns as first-class citizens. Similarly, Nakagawa, Guessi, Maldonado, Feitosa and Oquendo (2014) presents a four-step approach to the design and

development of RAs. Influenced by ISO/IEC 26550 *ISO/IEC 26550: 2015-Software and systems engineering–Reference model for product line engineering and management* (2015), Derras et al. (2018a) presents a four-phase approach for practical RA development in the context of domain engineering and SPL.

Additionally, Galster and Avgeriou (2011a) (2011) propose a 6-step methodology with two primary concepts: empirical foundation and empirical validity. Taking all these into consideration, the empirically-grounded RA methodology provides the most appropriate methodology to meet the purposes of this study. The reasons are that this methodology for RA development is adopted more than any other and that the methodology supports the objectives of this study.

Nevertheless, the methodology needs to be augmented with other approaches to arrive at the desired level of rigour and relevance. For example, comprehensive guidelines on how to collect empirical data in step 3 are missing. Thus, the approach to data collection, synthesis, and modelling was not clear. To address this, Nakagawa, Martins, Felizardo and Maldonado (2009) provides investigation guidelines and the RAModel concept, which has been absorbed to the artefact development method.

Also, the methodology does not describe how to evaluate the RA; thus, a more systematic and stronger evaluation approach is required. To address this, a single case mechanism experiment and expert opinion are employed to evaluate the artefact. The high-level overview of the artefact development method is discussed in Section 2.9.4, however, the detail of each step in this method is discussed in this section.

This artefact development method is composed of six phases, which are, respectively: 1) Decision on the type of the RA 2) Design strategy 3) Empirical acquisition of data 4) Construction of the RA 5) Enable RA with variability 6) Evaluation of the RA. The phrase *empirically grounded* refers to two major elements; firstly the RA should be grounded in well-established and proven principles; secondly, the RA should be evaluated for applicability and validity. These don't only belong to Galster and Avgeriou

(2011b) methodology, other researchers such as Cloutier et al. (2010a) and Derras et al. (2018a) have promoted the same ideas.

Phase 1: Decision on Type of the RA The precursor to effective RA development lies in the determination of its type, which sheds light on the requisite information collection and subsequent RA construction phases. The selection of the RA type for the present study draws upon the classification framework by Angelov, Grefen and Greefhorst (2009) and the contextual usage delineated by Angelov and Grefen (2008a).

The framework by Angelov et al. (2009) outlines five RA types, formulated to aid the analysis of RAs in terms of context, goal, and architecture specification/design relationships, employing three principal dimensions: context, goals, and design, with each dimension comprising relevant sub-dimensions identified through standard interrogatives for problem analysis.

These dimensions are systematically addressed: when, where, and who are mapped to the context; why is linked to the goal; and how and what are pertinent to the design. RAs are subsequently categorised into two principal groups: standardisation and facilitation RAs. This bifurcation aligns with the current study's aims and aids in clearly demarcating the domain for the RA being developed. Detailed categorisation of RAs, supplemented by industry examples supports the decision of choosing the right type of a RA, clarifying aim and demarcating boundaries.

Moreover, the recent SLR by Ataei and Litchfield (2020) on BD RAs expands upon Angelov's classification, offering a refined list with further RA examples, which is discussed in Appendix B. The domain-driven distributed BD RA chosen for this study pursues two major goals; 1) supporting the development of BD systems addressing the current limitations of BD RAs and 2) enabling an effective and scalable data architecture. Therefore, the outcome artefact will be a BD RA, that is a *classical standardisation RA* designed to be implemented in multiple organisations.

Phase 2: Selection of Design Strategy Angelov, Trienekens and Grefen (2008) and Galster and Avgeriou (2011a) have both presented that RAs can have two major design strategies: 1) RAs that are designed from scratch (practice driven) and 2) RAs that are based on other RAs (research-driven). Designing RAs from scratch is rare and usually takes place in an emergent domain that has not perceived a lot of attention (Angelov et al., 2008). On the other hand, most RAs today are the amalgamation of a priori concrete architectures, RAs, models, patterns, and best practices that together help shaping a compelling artefact for a class of problems.

RAs developed from scratch tend to create more prescriptive theories, whereas RAs developed based on the available body of knowledge tend to provide more descriptive design theories. The RA designed for this study is a *research-based RA* based on existing RAs, concrete architectures, patterns and best practices.

Phase 3: Empirical Acquisition of Data

As aforementioned, due to the limitations witnessed by empirically grounded RAs research methodology, this phase is augmented to increase the systematicity and transparency of data collection and synthesis through two SLRs. These SLRs are presented in detail in Chap 4 and Chap 5.

Phase 4: Construction of the RA

Based on the themes, theories, and patterns realised in the previous steps, the process of RA construction takes place. Integral to this step is the identification of elements that the RA should contain, how these elements should be synthesised, and how the RA can be portrayed and communicated. To describe the RA, ISO/IEC/IEEE 42010 standard is adopted (International Organization for Standardization (ISO/IEC), 2017). While this standard primarily focuses on concrete architectures, this study selectively adopts the relevant and applicable aspects of the standard. For instance, architecture viewpoints,

statements of corresponding rules, and expression of the architecture through architecture description languages (ADLs) have been adopted to help with the construction of Metamycelium.

A key challenge in the development of Metamycelium was to strike a balance between the specificity of the micro patterns and approaches to system development and general architectural concepts that reflect a view of the system as an array of interrelated entities. Angelov, Grefen and Greefhorst (2012) approached this problem by means of interrogatives through a defined framework that aims to guide the creation of RAs.

Cloutier et al. (2010a) suggests that a RA should entail technical, business, and customer context views, whereas O. Vogel et al. (2009) provided classified RA views based on the usage context, such as industry-specific, platform-specific, industry crosscutting, and product line RAs.

Stricker et al. (2010b) expressed their pattern-based RA by adhering several distinct views into one. W. L. Chang and Boyd (2018) presented NIST BD RA as a system constituent of logical components connected through interoperability interfaces in several fabrics. On the other hand, ISO/IEC/IEEE 42010 refrains from using phrases such as *technical architecture*, *physical architecture*, or *business architecture*.

Taking into account the practices demonstrated by Stricker et al. (2010b) and W. L. Chang and Boyd (2018) in their respective RAs, as well as Archimate's suitability for representing high-level architectural diagrams and its recognition as a standard architecture description language in ISO/IEC/IEEE 42010 (M. Lankhorst, 2013), it is decided to adhere several views into one and express the RA through Archimate. Archimate is a mature modelling language developed by Open Group that provides a uniform representation of high-level architectural diagrams aimed at portraying and delineating enterprise architecture (M. Lankhorst, 2013).

Archimate, being listed as a standard architecture description language in ISO/IEC/IEEE 42010, is designed based on a set of related concepts that are specialised towards the

system at different architectural layers. This means that the architect is enhanced with an integrated architectural tool that visualises and describes different architecture domains and their underlying relations (M. M. Lankhorst, Proper & Jonkers, 2010; Engelsman, Quartel, Jonkers & van Sinderen, 2011).

Phase 5: Enabling RA with Variability

The integration of variability into RAs is a fundamental step to ensure its applicability within the constraints of organisational-specific regulations and regional policies (Rurua, Eshuis & Razavian, 2019). Variability management is a documented necessity in both Business Process Management (BPM) (La Rosa, van der Aalst, Dumas & Ter Hofstede, 2009; Rosemann & Van der Aalst, 2007) and Software Product Line Engineering (SPLE) (L. Chen & Babar, 2011; Sinnema, Deelstra & Hoekstra, 2006), where it serves to tailor business processes and software artefacts to specific contextual needs.

Precise identification and explicit communication of variability are crucial to foster stakeholder dialogue, ensure traceability of decisions, and support the decision-making process (Czarnecki, Grünbacher, Rabiser, Schmid & Wasowski, 2012). Variability decision points are informed by data gathered in earlier analysis stages. Galster and Avgeriou (2011a) identify three methods to integrate variability into RAs, namely: annotating the RA, developing variability views, and creating variability models.

Existing literature does not extensively detail the criteria for selecting a method to enable variability. Following the approach by Rurua et al. (2019), this study applies Archimate annotations to introduce variability into the RA. This procedure comprises two stages: first, creating a bespoke layer to encapsulate primary variability concepts, and second, annotating the RA. The intention is not to exhaustively enumerate all potential variability points but to outline principal system-related architectural variabilities for consideration by architects to refine the design and facilitate the adoption of the RA.

The variability model is depicted in Figure 2.3 using Archimate's motivation layer. This model is informed by the graphical notations of variability by Pohl, Böckle and Van Der Linden (2005) and the concepts of variability management by Rurua et al. (2019), providing architects with a tool for application to the variable components of Metamycelium described in 6.11.

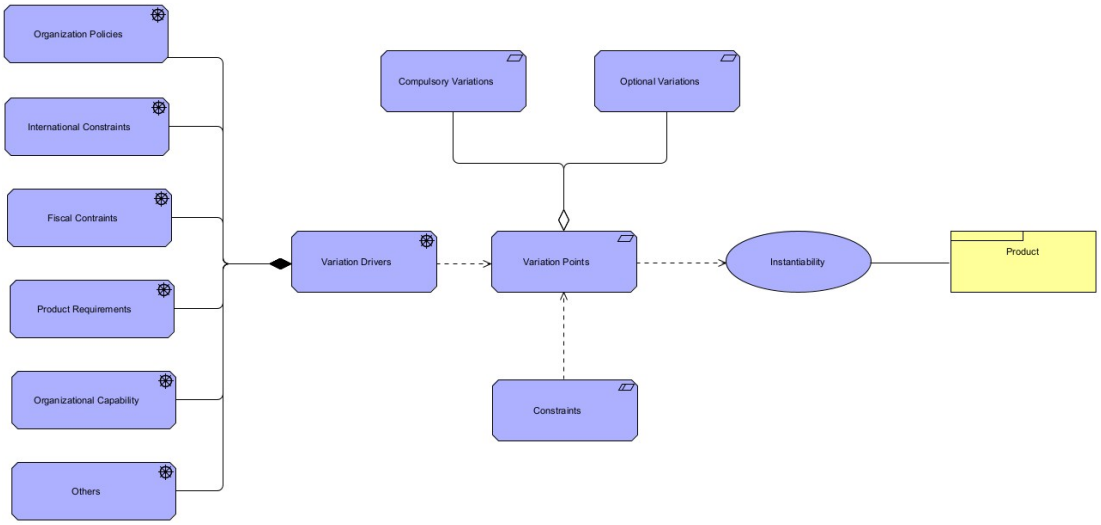


Figure 2.3: Variability management concepts model

Phase 6: Evaluation of the RA

For completeness, the evaluation of the RA is included as Step 6 in the artefact development method proposed by Galster and Avgeriou (2011b). However, it is important to note that the actual process of artefact validation and evaluation is carried out within the broader context of the DSR design cycle, specifically in the treatment validation and treatment evaluation steps, which are discussed in the subsequent sections.

While Galster and Avgeriou (2011b) methodology provides some high-level guidelines on evaluating an artefact, it is not exhaustive. Therefore, the guidelines presented by Wieringa (2014) are employed for conducting artefact validation and evaluation. Wieringa (2014) DSR framework and the artefact development method presented by Galster and Avgeriou (2011b) align well in their emphasis on the importance of artefact

evaluation.

2.9.5 Step 5: Treatment Validation

Treatment validation assesses the artefact in context to determine if it addresses the research problems and achieves its objectives. At this stage, requirements defined in Section 2.9.3 are run against the artefact to ascertain if they are met.

This validation process has challenges, notably the potential scarcity of real-world experiments that test artefacts in diverse settings. Nonetheless, the artefact must undergo validation, and subsequent design theories should be generated.

In this step, a validation model of the artefact, often interchangeably termed as a prototype, is created. This model is then subjected to a case mechanism experiment for validation purposes. It is essential to note that the validation phase focuses on determining the artefact's ability to satisfy research requirements and objectives.

Design theories, as described by Wieringa (2014), evolve from the interaction between an artefact and its intended problem context. The artefact in this thesis is validated using the case mechanism experiment. The design of this experiment follows the checklist provided by Wieringa (2014). This design is depicted in the following sections.

Research Context

This evaluation method is designed with a specific knowledge goal in mind. The goal is to validate the artefact against a specific context to prove its feasibility, to infer its internal mechanisms and to validate the theories discussed in Chapter 6. The current state of BD RAs and the theoretical framework that guides the creation of the artefact's prototype is discussed in Chapters 3, 4, 5 and 6.

Research Problem

The conceptual framework for this research comprises architectural structures. The framework's validity relies on the clarity of definitions and the avoidance of mono-operation bias (relying on a single operationalization of a construct) and mono-method bias (using a single method of measurement). Within this context, the case mechanism experiment specifically aims to investigate the internal workings and performance of the Metamycelium artefact in handling BD challenges.

The experiment focuses on understanding the mechanisms, trade-offs, sensitivities, and architectural explanations that underlie the artefact's behaviour and effectiveness. To achieve this, the experimental design employs open-ended knowledge questions that probe into the artefact's mechanisms, trade-offs, sensitivity, and architectural aspects. Of particular interest is understanding the stimuli that trigger specific mechanisms and how the interaction of various components within the artefact leads to the emergence of observed phenomena.

Research Design and Validation

This section focuses on the third part of the single-case mechanism experiment design, namely, the research design and validation. In this section, inference support, repeatability, and ethics are presented. These aspects are important to the overall success of the experiment.

Inference Support

In the context of the evaluation, the inference approach adopted is abductive reasoning. Abductive inference is well-suited for evaluating a distributed and domain-driven BD RA due to its focus on seeking the simplest or most plausible explanations for observed phenomena (Hoffman, 1997). The abductive reasoning process for evaluating

Metamycelium will involve:

- Observing and documenting the architecture's behaviour, performance, and effectiveness in handling BD challenges.
- Generating plausible hypotheses to explain the underlying mechanisms, interactions, and emergent behaviours that contribute to the observed phenomena (C. H. Yu, 1994).
- Systematically evaluating and refining the hypotheses based on their explanatory power, consistency with existing knowledge, and ability to account for observed data and patterns.
- Iteratively revising or discarding hypotheses based on new observations or insights gained during the evaluation process (Magnani, 2011).

While abductive reasoning is chosen as the primary inference approach for evaluating Metamycelium, other inference methods were also considered. Statistical inference, which primarily deals with drawing conclusions about a population based on a sample, may not capture the complexities of the architecture's behaviour. Descriptive inference, which focuses on summarising data patterns, may not provide insights into the underlying mechanisms driving the architecture's performance.

Analogic inferences, which involve drawing conclusions based on similarities with other cases or domains, were deemed less suitable for evaluating a distributed BD architecture due to the unique complexities and intricacies of the architecture and its specific domain. The unique complexities and intricacies of the architecture and its specific domain require a more nuanced approach like abductive reasoning, which allows for a deeper understanding of the system's behaviour and the underlying mechanisms driving its performance (Wieringa, 2014).

Repeatability

To ensure repeatability and enhance the transparency and systematicity of the artefact prototype evaluation, rigorous research methods are employed. Notably, all code implementations related to the distributed BD architecture artefact are stored in a publicly accessible version control repository on Github in two repositories (Polyhistor, 2023b, 2023a). This practice enables researchers and practitioners to replicate the experiments, review the codebase, and validate the findings. By providing a centralised and versioned repository, the artefact project fosters collaboration, knowledge sharing, and reproducibility within the research community.

Furthermore, the explanations accompanying the artefact prototype encompass comprehensive guidelines and instructions, allowing independent researchers to replicate the experiments and validate the results. The inclusion of detailed procedural descriptions, datasets, and configuration parameters ensures that others can conduct similar evaluations and compare their outcomes against the reported results.

Research Execution and Analysis

In the Research Execution and Analysis phase, the practical application of the design is initiated, and a prototype is developed based on the selected technologies deemed most appropriate for the RA. This phase aims to evaluate the design in a controlled environment, yielding data that enables an analysis of the design's effectiveness and its alignment with the defined research objectives. Crucial to this phase is the translation of theoretical constructs into a functional entity. It provides empirical data for rigorous analysis, facilitating invaluable insights for the validation or the necessary revision of the design.

To ensure that the technology choices are justified, these decisions are aligned with research objectives (Chapter 2) and the requirements of the artefact (Chapter 6.1). This

involves a rigorous, structured technology selection process, which can be accomplished through a few steps.

Technology Selection Process

The technology selection process section outlines the systematic approach employed to identify and evaluate the most suitable technologies for instantiating the Metamycelium prototype, ensuring alignment with the research objectives and artefact requirements.

Requirement Analysis

The first step is requirement analysis, which starts by defining the precise capabilities of the prototype needs. These are based on the requirement specification done in Section 6.1 and the components of Metamycelium discussed in Section 2.9.4.

Evaluation Criteria

For a thorough evaluation of the potential tools, adherence to an established quality standard is crucial. In this context, 'tools' refers to the specific software technologies, frameworks, and platforms that will be used to create a concrete implementation of our Reference Architecture. The ISO/IEC 25000 SQuaRE standards (*ISO/IEC 25000:2005. Software Engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*, 2014), globally recognised for software quality measures, are chosen to guide the tool evaluation process. Since we are selecting technologies for instantiating a concrete architecture from the RA, these software quality standards are directly applicable to evaluating the implementation tools and technologies.

The SQuaRE standards encompass criteria such as functional suitability, performance efficiency, usability, reliability, and security, among others, serving as a comprehensive framework for assessing software quality.

However, the entire ISO/IEC 25000 SQuaRE standards series is quite extensive and may be time-consuming to fully apply, particularly given the resources associated with this research. Therefore, a condensed version of the SQuaRE standards is proposed, focusing on critical factors aligned with the research context and project requirements.

This modified framework comprises functional suitability, maintainability, compatibility, and portability. Functional suitability is concerned with whether the tool functions as expected and meets the stated requirements; Maintainability refers to the ease with which the software can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment; Compatibility explores the degree to which a tool can perform its required functions together with other tools; and lastly, Portability assesses the effectiveness and efficiency with which a system or component can be transferred from one hardware, software, or operational environment to another.

For the scoring of these criteria within the evaluation matrix, a systematic and objective approach is followed wherever possible. Each criterion is assigned a score on a scale of 1–5, with 1 indicating poor alignment with the criterion and 5 indicating excellent alignment.

- Functional Suitability is evaluated based on a checklist of required features and functionalities explained in Sections 6.3 and 7.3. The more features a tool has that are aligned with the requirements, the higher the score.

- Maintainability is scored by reviewing the tool's documentation, update frequency, and the complexity of making changes. Tools that have good documentation, frequent updates, and a simpler process for making modifications get a higher score. The tool documentation is sourced from the official website, GitHub/GitLab repositories, or any platform where the tool is hosted. Update frequency is determined by checking the release notes, commit history, and issue tracking systems associated with the tool's repository or project management platform.

Additionally, hands-on exploration and testing are conducted to evaluate the complexity of making modifications to the tool, such as adding new features, fixing bugs, or integrating with other components of the Metamycelium architecture.

- Compatibility is assessed by examining the tool's interoperability with other systems and tools, its adherence to industry standards, and its ability to integrate seamlessly into the existing infrastructure. Tools that show a high level of compatibility receive a higher score. This assessment involves analysing the tool's documentation, specifications, and integration guides to understand its supported protocols, data formats, and communication interfaces. Additionally, industry reports, case studies, and user forums are reviewed to gather insights on the tool's real-world compatibility and integration experiences. Where possible, hands-on testing and prototyping are conducted to evaluate the tool's compatibility with other components of the Metamycelium architecture and its ability to operate within the target deployment environments.

- Portability is scored by testing the tool in different environments representative of the target deployment scenarios for Metamycelium. The environments are selected based on factors such as the predominant operating systems used in BD deployments (e.g., Linux distributions like Ubuntu, CentOS), common hardware configurations (e.g., x86-64 architecture, varying CPU and RAM specifications), and cloud environments (e.g., AWS, Azure, Google Cloud). The selection process also considers the tool vendor's recommendations and real-world usage data from industry reports and user forums.

Technology Research

The research methodology employed for the technology research phase is a lightweight, structured literature review combined with hands-on exploratory testing. This dual

approach ensures a comprehensive understanding of the potential technologies and their practical implications.

The literature review involves examining academic papers found in Chapters 3, 4 and 5 , industry reports, product documentation, and user testimonials related to the potential technologies. The aim is to understand the theoretical and practical aspects of the technologies, their strengths, limitations, and how they align with the established evaluation criteria. The sources are identified and accessed through academic databases, industry publications, online forums, and the official websites of the technologies.

The exploratory testing involves a hands-on examination of the technologies. This practical exploration provides valuable firsthand experience with the tools and their capabilities, while acknowledging the inherent subjectivity of such hands-on assessment. The testing process is structured around the evaluation criteria to ensure a focused and relevant exploration. Each technology is tested for functional suitability, maintainability, compatibility, and portability, with findings documented systematically.

A synthesis of the insights gained from the literature review and exploratory testing forms the basis of the technology evaluation. While following a structured methodology guided by established evaluation criteria and prototype requirements, we acknowledge that complete elimination of researcher bias is not possible in such hands-on exploration.

Comparison and Justification

Upon the completion of the technology research phase, the evaluation process is initiated. This stage capitalises on the detailed evaluation matrix, which offers a comprehensive visual representation of each technology's performance based on the aforementioned criteria. Each technology is scored and the scores are aggregated to create a composite score, thereby facilitating comparison across different tools.

In the first phase of this process, all the RAs in Chapter 4 are analysed for prototypes. That is, each RA is analysed with regard to how it is instantiated and what technologies

are used in that instatiation. This includes the researcher's approach to technology selection. In the second phase, all platforms that offer technology categorisation and comparison are reviewed. These platforms are Apache Software Foundation (Apache, 2023b), GitHub (GitHub, 2023), G2 (G2, 2023), Capterra (Capterra, 2023), and StackShare (StackShare, 2023).

Additionally, prominent industry and academic sources are leveraged, each offering distinct perspectives and insights on current trends and capabilities. These sources encompass the "2022 Gartner Magic Quadrant for Analytics and Business Intelligence Platforms" (*2022 Gartner® Magic Quadrant™ for Analytics and Business Intelligence Platforms*, 2022), McKinsey & Company's "Technology Trends Outlook" (*Technology Trends Outlook*, 2023), MIT Technology Review Insights' report on "Building a High-Performance Data and AI Organisation" (*Building a High-Performance Data and AI Organization*, 2023), and the annual StackOverflow Developer Survey (Overflow, 2022). These tools are checked against the components of Metamycelium that are described in Section 6.5.1.

The aim of the evaluation is not to evaluate all technologies in the market but to only analyse the most supported and most used tools and technologies. The result of the search discussed in the previous section provided a wide variety of tools. While the majority of these tools are specifically designed for BD, some are general software engineering tools, and some are more data science and business intelligence oriented.

The most supported and most used tools were determined based on factors such as the number of GitHub stars, total downloads, and mentions in well-established industry surveys like the Stack Overflow Developer Survey. The results of the technology selection research, including the chosen technologies for each component of Metamycelium, are presented in Section 7.2 of Chapter 7.

2.9.6 Step 6: Treatment Implementation and Evaluation

The treatment implementation and evaluation phase of this thesis involves creating and deploying the prototype. Evaluation of the RA ensures that it achieves the goals stated prior to its development, tests its effectiveness and usability, and ensures it addresses the identified problems. Key pillars of the evaluation are the RA's correctness, utility, and adaptability efficiency (Galster & Avgeriou, 2011a).

The RA's quality is assessed by its transformation capability into an effective concrete architecture. Building this RA on top of former RAs offers the advantage of drawing insights from other studies for the evaluation process (Sharpe et al., 2019).

However, evaluating RAs presents distinct challenges. For one, RAs are abstracted at a higher level, causing stakeholders to remain ungrouped and leading the RAs to be more focused on architectural qualities than concrete architectures. Existing methods designed for concrete architectures, like Scenario-based Architecture Analysis Method (SAAM) (Kazman, Bass, Abowd & Webb, 1994), Architecture Level Modifiability Analysis (ALMA) (Bengtsson & Bosch, n.d.), Performance Assessment of Software Architecture (PASA) (Williams & Smith, n.d.), and Architecture Trade-off Analysis Method (ATAM) (Kazman et al., 1998a), are ill-suited for direct application to RAs.

There are three principal challenges:

1. Lack of a clearly defined group of stakeholders for RAs makes applying existing evaluation methods difficult (Angelov et al., 2008). Since methods like ATAM heavily rely on stakeholder participation, the abstract nature of RAs makes reaching diverse stakeholder groups challenging. Potential biases may also be introduced when stakeholders from varied backgrounds participate, with some possibly lacking architectural visions.

2. Evaluation methods for concrete architectures heavily utilise scenarios. Given the abstract nature of RAs, creating scenarios is cumbersome. Developing a

broad spectrum of scenarios can complicate data analysis, and prioritising these scenarios, defining, and validating them becomes problematic. Generalizing scenarios might make RA evaluation incomplete and ineffective (Avgeriou, 2003a). Such issues are noticed even while evaluating complex concrete architectures (Bengtsson & Bosch, 1998).

Due to the aforementioned limitations, existing architecture analysis methods are not adept for RA evaluation. Nonetheless, industry researchers have taken steps to address these challenges. For instance, Angelov et al. (2008) modified ATAM for RAs by incorporating various contexts and defined scenarios. They also extended ATAM to assess completeness, buildability, and applicability. However, the process remains tedious and may yield incomplete results. Angelov et al. noted that some participants in their study possibly lacked architectural insights, which could affect the evaluation outcomes.

Other approaches, like the one proposed by Graaf, Van Dijk and Van Deursen (2005), extend SAAM to minimise its organisational impact. Galster and Avgeriou (2011a) recommend prototyping, reference implementations, and incremental methods for RA validation. Rohling, Neto, Ferreira, Dos Santos and Nakagawa (2019) evaluated their RA against set requirements, facilitated by methodologies from Nakagawa et al. (2009) and RAModel (Nakagawa, Oquendo & Becker, 2012).

As discussed in Section 2.9.5, an RA prototype is developed based on the theories presented in Section 6.4. This prototype is subjected to a case mechanism experiment for validation purposes. However, this case mechanism experiment not only validates the artefact's ability to meet the research requirements and objectives but also serves as an evaluation of its effectiveness in addressing real-world challenges and its potential for practical application. Through this comprehensive assessment, which exposes the artefact to a range of scenarios designed to test various aspects of its performance and

functionality (see Chapter 7), the case mechanism experiment simultaneously validates and evaluates the artefact. That is, the case mechanism experiment is achieving both *artefact validation* and *artefact evaluation*. The validation part is highlighted by the creation of the artefact out of the design theories discussed in Section 6.4, and the evaluation part is highlighted by the creation of a range of scenarios and running them against the artefact as discuseed in Chapter 7.

Following this validation and evaluation through the case mechanism experiment, the artefact and the prototype undergo further scrutiny through expert opinion collection. This additional step assesses the artefact's effectiveness and usability from a practitioner's perspective (see Chapter 8). The ISO/IEC 25000 Software Product Quality Requirements and Evaluation (SQuaRE) (*ISO/IEC 25000:2005. Software Engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*, 2014) guide the technology selection for the RA's instantiation. In the following sections, the research methodology employed for expert opinion collection is discussed.

Research Methodology for Gathering Expert Opinion

Inspired by the works of Kallio, Pietilä, Johnson and Kangasniemi (2016), the research methodology chosen for eliciting expert feedback encompasses five distinct stages: 1) identifying the rationale for gathering expert opinion, 2) developing the preliminary expert opinion gathering guide, 3) desigining the research method for collecting data in a rigorous manner and free of bias 4) pilot testing the guide, 5) presenting the results.

Given that the conceptual framework of this study is comprised of architectural constructs, expert feedback is posited as a suitable approach (Creswell et al., 2007). Venturing into a new domain replete with latent potential, it is postulated that these erudite perspectives can furnish invaluable insights. The expert opinions can offer prospective trajectories warranting exploration, thereby enhancing the robustness of

this study's theoretical underpinnings.

The Rationale for Gathering Expert Opinion

The intricacy of domain-driven distributed RAs for BD like Metamycelium demands an evaluation that transcends theoretical boundaries. Experts in this field possess an unparalleled depth of knowledge, allowing them to assess the architecture's practical relevance and identify any potential oversights. Their profound understanding ensures that the design is aligned with industry challenges, scalable to evolving data needs, and benchmarked against existing solutions.

Furthermore, their position as external reviewers, combined with their passionate expertise and hands-on experience, provides valuable perspective beyond internal evaluations. Thus, seeking expert opinion is not merely a commendable step but an essential one, providing a comprehensive and nuanced review of the proposed architecture. Moreover, their insights are instrumental in validating the architecture's robustness against varied challenges, from scalability concerns to evolving data patterns.

Developing the Guide

The protocol chosen for expert feedback gathering is based on the research objectives discussed in Section 2.3, thereby aspiring to extract the most exhaustive dataset possible. This protocol is semi-structured, augmenting the potentiality for unearthing insight while ensuring participant-centricity. Nonetheless, a suite of close-ended questions are chosen, acting as efficacious conversation initiators and facilitating quantifiable data extraction.

The protocol guide is segmented into main themes and ancillary follow-up questions. The themes adhere to a progressive and logical trajectory, congruent with the tenets explained by Kallio et al. (2016).

Desigining the research method for collecting data

After finalising the guide for collecting expert opinion, purposive sampling is applied to identify experts (Baltes & Ralph, 2022). This method was chosen because it provides in-depth insights through targeted expert selection. Moreover, it guaranteed representation without needing a specific sampling framework.

Contacts with peers, associates on ResearchGate and LinkedIn, and specifically targeted professionals with titles such as 'data engineer', 'data architect', 'senior data engineer', 'solution architect', 'lead architect', and 'big data architect' are initiated. Other relevant titles that work in the area of BD engineering or BD architecture are listed. Additionally, founders of major data enterprises and scholars active in the realm of BD systems are approached. Over the span of two months, three specialists across different sectors were selected for feedback collection. A detailed profile of these experts is portrayed in Table 2.1.

Table 2.1: Participants Demographics

Expert	Role	Years of Experience	Industry
i1	Solution Engineer	10	Software Development (Event Driven Solutions)
i2	Solution Architect	11	Software Development (Big Data Solutions)
i3	Principal Architect	32	Biotechnology Research (Veterinary Hardware and Software)

Expert opinions are collected using the software Zoom. Following each session, recordings are downloaded along with their automatically generated transcripts. These transcripts underwent an initial review to rectify any inconsistencies or issues before being uploaded to the NVivo software. Each transcript is coded in accordance with a predefined set of primary themes. These themes, detailed in Section A.5 of the guide,

include ‘Applicability & Industrial Relevance’, ‘Strengths & Distinctive Features’, and ‘Challenges & Potential Barriers’. As each interview progressed, the transcript was actively coded, and new codes were generated as necessary. The results are then collected and provided in the following sub-sections.

Pilot testing the guide

The protocol guide is internally tested by peer review. This occurs by inviting an experienced academic to review the guide and provide feedback on it. Employing this approach facilitates the diagnosis of potential incongruities, culminating in the excision of unnecessary elements. Ancillary questions are strategically deployed to recalibrate the discourse towards the research objectives, thereby rendering comprehension more facile.

Certain interrogatives undergo ad-hoc adaptations, obviating an overly prescriptive discourse trajectory. Subsequent to this preliminary validation, an empirical pilot study is conducted to ascertain the protocol’s appropriateness. This iterative refinement is pivotal in calibrating the protocol, thereby accentuating the rigour of the study. This protocol is catalogued in Appendix A.

2.10 Iterations and Evolution

The evolution of Metamycelium follows a series of iterative refinements as portrated in Table 2.2. It began with the first generation, termed *Neomycelia* (Ataei & Litchfield, 2021a). This iteration integrated event-driven architectures and microservices but was mainly technocentric, emphasising tools and technologies over business needs.

The second iteration transitioned to a domain-centric approach, applying domain-driven design principles. However, this shift highlighted challenges in addressing cross-cutting concerns crucial to BD solutions.

The third iteration prioritised security and privacy, which are essential for safeguarding data and maintaining user confidentiality. As data volumes expanded, the architecture needed to address growing potential vulnerabilities and exposures.

The fourth iteration, spurred by challenges in security measures, especially secret management, further developed the domain-driven design. This evolution targeted clearer inter-domain communication and defined responsibilities. By aligning architectural constructs with business domains, the design ensured solutions were more in tune with real-world challenges.

Subsequently, the architecture adopted the service mesh, a method apt for managing the complex network of services in BD systems. This infrastructure layer streamlines service-to-service communication. With the complexity of the architecture, the need for observability became evident. It was not merely about monitoring but understanding how data moved, changed, and integrated throughout the system, allowing for timely identification and mitigation of issues.

Further iterations identified challenges in maintaining consistent standards across distributed domains. The non-conformity by local agents jeopardised data integrity. This highlighted the need for universal standards and policies. To enforce these standards, an automation step was added to apply global policies consistently.

As the architecture was further refined, ensuring data consistency became pivotal. The introduction of bitemporality and immutability addressed this. Bitemporality allowed for data management across two temporal dimensions, and immutability ensured data, once stored, remained unchanged, reinforcing data integrity and traceability.

Although these iterations trace the architectural progression, the intricate details of each step lie beyond this study's purview.

Table 2.2: Artefact Evolution

Iteration	Name	Description
1	Neomycelia	Technocentric, Event-Driven, and Microservices
2	Neomycelium	Domain-Centric, Addressing Cross-Cutting Concerns
3	Neomycelium 2.0	Security and Privacy, Safeguarding Data and User Confidentiality
4	Cybermycelium	Domain-Driven Design, Clearer Inter-Domain Communication
5	Cybermycelium 2.0	Service Mesh, Streamlined Service-to-Service Communication
6	Cybermycelium 3.0	Observability, Understanding Data Movement and Integration
7	Neomycelia 2.0	Standards and Policies, Ensuring Consistent Standards
8	Terramycelium	Data Consistency, Data Management Across Temporal Dimensions

2.11 Conclusion

In this chapter, firstly, the characteristics of good IS research are discussed. From there on, the research philosophy and research approaches are discussed. Further, the embodied research design and research methods are elaborated. This chapter aims to provide a high-level understanding of the research methodology designed for this thesis.

Building upon this methodological groundwork, the ensuing chapter delves into a foundational narrative literature review on BD and its environs.

Chapter 3

A Narrative Literature Review on Big Data

3.1 Introduction

The previous chapter provided an introduction to the studies, depicting major elements such as motivation, research methods, and philosophy. This narrative literature review presents a comprehensive overview of BD , its significance, and its impact on the current technological landscape. This chapter aims to establish a solid contextual framework for the subsequent exploration of BD RAs and the development of the Metamycelium architecture.

This review traces the evolution of BD, examining the key technological advancements that have fueled its growth and the profound impact of emerging technologies on BD architectures. It highlights the ubiquity of BD applications across diverse sectors, underlining the transformative potential of BD and the importance of effective BD management strategies.

The chapter also delves into the business benefits and challenges associated with BD adoption, emphasising the need for a more comprehensive and unified approach to BD system development. It explores the fundamental characteristics of BD, namely volume, variety, velocity, veracity, and value, which are crucial considerations in designing architectures that can effectively handle BD workloads.

This review begins with the Section 3.2, tracing the evolution of BD and its growing importance. Next, Section 3.3 provides a concise historical overview of data management and BD development. The 'Impact of Emerging Technologies on Big Data Architectures' is examined in Section 3.4, highlighting how technological advancements are shaping BD systems.

Section 3.5 then explores the ubiquity of BD applications across various sectors, emphasising its transformative potential in fields such as healthcare, social media analysis, crime prevention, and tourism. The chapter proceeds to examine the business benefits and challenges associated with BD adoption in the subsequent section, discussing the

shift towards data-driven decision-making and the need for alignment between insights and business objectives.

'An Architecture-Centric Approach to Big Data' is addressed in its own section, introducing the limitations of traditional data management systems in handling BD and the need for more flexible and scalable solutions. Subsequently, Section 3.8 delves into the key characteristics of BD, commonly known as the '5 Vs' (Volume, Variety, Velocity, Veracity, and Value), elucidating how these properties shape the challenges and opportunities in BD systems. The chapter concludes with a critical analysis of the BD landscape in Section 3.9, highlighting gaps in current research and practice

3.2 The Age of Big Data

Along the axis of technological change, there is substantial progress in the field of software engineering. The rapid growth of this field has led to a discernible gap in the adoption of new techniques and tools, particularly within the realm of data engineering (Dehghani, 2022; Reis & Housley, 2022). This phenomenon, characterised by the slow uptake of innovative software engineering advancements, has significant implications for big data management and analysis (Ataei & Litchfield, 2023).

ThoughtWorks' Technology Radar provides insights into the evolving landscape of software engineering tools and techniques, highlighting a notable trend: the low adoption rates of these advancements in the data engineering sector (ThoughtWorks, 2023). To understand how this gap emerged, we take a brief look at its history.

3.3 A Brief History

Initially, computers were viewed merely as tools for calculations and executing algorithms. However, with the commercial availability of computers in the mid-1950s and

their subsequent utilisation for business operations, the creation and accumulation of large data volumes began. This shift underscored the value of data and the importance of effectively storing and managing it.

That's when the industry came up with the concept of a Database Management System (DBMS), and humanity began to store data for various purposes. In 1968, as a result of a NATO-sponsored conference, the term 'software engineering' emerged, which refers to a highly systematic approach to software development and maintenance (Wirth, 2008).

By 2005, as BD's implications began to solidify, pivotal advancements emerged: Yahoo developed Hadoop, a distributed processing framework; Google introduced MapReduce, a programming model for processing and generating large datasets; and in 2009, the Indian government undertook the ambitious project of capturing the iris scans of 1.2 billion inhabitants, epitomising the scale of BD applications.

Such large-scale initiatives underscored the inadequacy of extant data systems. McKinsey et al. (2011) publication, "Big Data: The Next Frontier for Innovation", further emphasised this inadequacy, triggering more investments in the BD domain.

While the growth of computational power, the emergence of open-source communities, and the widespread use of the internet have accelerated advancements in BD, an important challenge persists: contemporary data systems are still striving for the maturity to effectively handle the magnitude and intricacies of BD.

3.4 Impact of Emerging Technologies on Big Data Architectures

The evolution of BD can be attributed to the increasing availability of data, particularly with the proliferation of digital devices (Lycett, 2013). The interconnectedness among

suppliers, customers, and stakeholders has been amplified with the rise of digital platforms (Bughin, 2016).

The launch of 5G technology, especially in regions such as the UK, represents more than a mere speed advancement from 4G. Advanced features, including bi-directional large bandwidth sharing, gigabit-level broadcasting, and support for AI-equipped wearables, are poised to massively influence the data landscape (Gohil, Modi & Patel, 2013). Such features not only increase the volume of data but also introduce complexity into data management, storage, and processing (X. Jin, Wah, Cheng & Wang, 2015).

Such technological advancements necessitate robust, scalable, and adaptable data architectures. The sheer volume of data generated every second poses significant architectural challenges (Rad & Ataei, 2017c). As we navigate through the era of technological advancements and the relentless accumulation of data, the challenge becomes not just about gathering vast amounts of data but also making sense of it. This scenario sets the stage for an exciting area of research, particularly in the realm of RAs. The concept of domain-driven distributed RAs emerges as a particularly intriguing solution, promising a structured approach to harnessing the power of BD.

Yet, despite its potential, this area remains largely unexplored, sparking a series of questions about its effectiveness and scope. This study is motivated by the opportunity to delve into this underexplored area, aiming to uncover how domain-driven distributed RAs can provide a systematic framework for extracting valuable insights from BD. Through this exploration, the aim is not only to address the challenges posed by BD but also to turn these challenges into opportunities for innovation in the rapidly evolving technological landscape.

3.5 The Ubiquity of Big Data

Social, commercial, and industrial trends demonstrate the ubiquity of BD. At the 2013 World Government Summit in Abu Dhabi, Joseph S. Nye, a former US Assistant Secretary of Defence and a professor at Harvard University, highlighted the potential for future governance in the age of information (Nye, 2013). Joseph S. Nye proposed a scenario where central governments could use BD to strengthen control. The field of BD is vast and multifaceted, showcasing its breadth through the substantial body of research conducted on the subject. According to Google Scholar, from January 2010 to August 2023, there have been 17,800 studies exploring a wide array of topics within big data, including mathematical techniques, decision-making methods, data characteristics, technical challenges, and adoption failures.

In the realm of social networking, a study conducted by Dodds, Harris, Kloumann, Bliss and Danforth (2011) analysed a dataset of 46 billion words from nearly 4.6 billion expressions posted by 63 million unique users over 33 months. This research, aimed at understanding temporal patterns of happiness, exemplifies the extensive scale and depth at which BD is utilised in contemporary analysis.

S. Jin et al. (2015) proposed the Distributed Community Structure Mining (DCSM) framework for BD analysis, utilising local information data in conjunction with the MapReduce paradigm and algorithms such as FastGN and Radetal. This approach effectively addresses key aspects of BD challenges, including scalability, velocity, and accuracy. In a broader social context, Durahim and Coşkun (2015) employed a sentiment analysis model to research the overall well-being of Turkish citizens, demonstrating the application of BD techniques in social science studies.

Similar research efforts include the analysis of suspended spam accounts on X (formerly Twitter), focusing on profile properties and interactions to identify spammers and malicious users using BD techniques (Almaatouq et al., 2016). Another study by

Chainey, Tompson and Uhlig (2008) explores hotspot mapping for identifying spatial patterns of crime. Their research concludes that hotspot mappings, utilising historical data, can effectively pinpoint areas where crimes occur most frequently.

Moreover, Li, Yen, Lu and Wang (2012) used a large dataset from the Bank of Taiwan to develop a BD system for identifying signs and patterns of fraudulent accounts. They develop a detection system applying the Bayesian classification and association rules. In a related vein, other research predicts negative behaviour spreading dynamics, emotional response detection when browsing Facebook, and identifies the impacts on national security using the US intelligence community datasets (Crampton, 2015).

Exploration into various domains, as depicted in Table 3.1 reveals the significant progress made with the adoption of BD. For instance, Popescu, Iskandaryan and Weber (2019) undertook a study with data from the Plekhanov Russian University of Economics and HAN University of Applied Science to comprehend how BD can enhance the understanding of multifaceted aspects of international accreditations. The study illustrated that BD can provide fresh methodologies for these institutions, emphasising the transformative role of BD.

Furthermore, M. Zhang, Liu and Feng (2019) delved into the potential of BD for tour and creative agencies. The goal of this research was not just data extraction but also the formation of strategic objectives that businesses can subsequently utilise for tangible benefits.

As can be discerned from Table 3.1, there is an extensive amount of research on BD's application across diverse sectors. The table offers a snapshot of these studies, categorising them based on their contribution, title, and domain.

Table 3.1: Ubiquity of BD Applications

Contribution	Title	Domain
Luo, Wu, Gopukumar and Zhao (2016)	Application of big data in health care	Health Care
Murdoch and Detsky (2013)	Adoption of big data in health care	Health Care
Y. Zhang, Qiu, Tsai, Hassan and Alamri (2015)	Application of big data in cloud in health-care cyber-physical system	Health Care
Bates, Saria, Ohno-Machado, Shah and Escobar (2014)	Exerting big data analytics to identify and manage high-risk and high-cost patients	Health Care
K. Lin, Xia, Wang, Tian and Song (2016)	Designing systems for emotion-aware healthcare using big data	Health Care
Srinivasan and Arunasalam (2013)	Analysing health insurance claims to detect frauds, errors, waste, abuse	Health Care
Mehta and Pandit (2018)	Analysis of application of big data in healthcare	Health Care
Firouzi et al. (2018)	Amalgamation of Internet of Things (IoT) and Big Data for a smarter healthcare	Health Care
Firouzi et al. (2018)	Analysis of application of big data in healthcare	Health Care
M. Chen et al. (2018)	Analysis of application of big data in healthcare	Health Care

Continued on next page

Table 3.1 – Ubiquity of BD Applications

Contribution	Title	Domain
Asur and Huberman (2010)	Predictive analytics using social networks	Social
Dodds et al. (2011)	Revealing temporal patterns of happiness	Social
Guo and Vargo (2015)	Utilising big data to examine message networks such as Twitter and traditional news media	Social
S. Jin et al. (2015)	Developing a novel distributed community structure mining framework	Social
Durahim and Coşkun (2015)	Analysis of overall happiness in Turkey through twitter analysis and big data	Social
Bohlouli, Dalter, Dornhöfer, Zenkert and Fathi (2015)	Knowledge discovery from big data	Social
Chainey et al. (2008)	Predicting of spatial patterns of crime using big data	Crime and Fraud
Li et al. (2012)	Using bank data to identify patterns of fraud	Crime and Fraud
Liao, Squicciarini and Griffin (2015)	Predicting abusiveness in online commentaries and preventing them	Crime and Fraud
Tran et al. (2018)	Data driven approaches for credit card fraud detection	Crime and Fraud
Sigala (2019)	A book on big data and how it can bring innovation to tourism business	Tourism

Continued on next page

Table 3.1 – Ubiquity of BD Applications

Contribution	Title	Domain
M. Zhang et al. (2019)	Analysing the application of big data in tourism	Tourism
Dezfouli, Shahraki and Zamani (2018)	Developing a tour model using big data	Tourism
Qin et al. (2019)	Utilisation of big data with Call Detail Record (CDR) data and mobile real-time location data to monitor the tourist flow and travel behavior	Tourism

3.6 Business Benefits and Challenges

BD and the benefits it brings have led to novel approaches in decision-making, often referred to as data-grounded decision-making (Comuzzi & Patel, 2016). Such organisational adaptations aim to enact meaningful interventions both internally and externally, especially in areas that previously relied more on intuition than on data and precision (Wamba et al., 2017).

As data-driven strategies, philosophies, tools, and methodologies develop, they revolutionise traditional views on decision-making, business process management, and predictive models (Popovič, Hackney, Tassabehji & Castelli, 2018).

There is, and will continue to be, immense focus on the evident advantages of BD to understand customer behaviour patterns, experimental orientation, organisational functions, business insight, predictive decision-making, and more comprehensive business and marketing plans compared to traditional methods (van den Driest, Sthanunathan & Weed, 2016).

However, with profound changes come challenges. Beyond technical difficulties and a lack of skilled personnel, transitioning to data-based management or a data-grounded decision-making system demands a vision that aligns business goals with insights. This alignment presents substantial potential for mistakes (Ranjan, 2019).

Investing in BD and deriving insights is essential for many competitive businesses. Yet, it is crucial to ensure that these insights align with business objectives. From planning to mentoring, strategizing, networking, and communicating, data can offer a consolidated perspective that encompasses the field's primary aspects, guiding executives towards successful outcomes.

For most businesses, the business plan remains central to an effective strategy and execution. It guides decisions on 'where to play' and 'how to win' (van den Driest et al., 2016). The business plan dictates resource allocation, financial forecasting, and outlines the roadmap for achieving targets. Here, insights can significantly influence strategy, guide activities, and set business phases (Y.-S. Chen, 2018).

Despite BD's prominence, most of these planning processes rely on an executive's judgement, which can be biassed by past experiences, cognitive patterns, emotions, and personal beliefs. By incorporating insights into this critical phase, business activities can align more precisely with overarching goals. That's why high-performing companies integrate insights into all primary decision-making stages, fostering a data-centric culture (van den Driest et al., 2016).

Nevertheless, there is a notable lack of research into BD architecture and RAs for data-driven systems. Most studies focus on BD analytics capability, pattern recognition, and BD challenges. Meanwhile, other crucial areas, like insight orchestration, necessary socio-technological developments, BD RAs, and data-centric artefacts, remain overlooked (Mikalef, Pappas, Krogstie & Giannakos, 2018).

Several authors have stated that research often overlooks the concept of *insight* and its role in achieving specific goals (van den Driest et al., 2016; H.-M. Chen, Kazman &

Matthes, 2016; H.-M. Chen, Kazman, Haziyev & Hrytsay, 2015). Various aspects of data and its potential to systematically generate value require a well-defined architecture within the IT-business value domain (Serra, 2024b).

Insufficient research into BD system architectures and system development significantly constrains the potential benefits of BD, leaving professionals to navigate unfamiliar territories (Mikalef et al., 2018). This gap in knowledge becomes particularly critical when considering the transformative changes that BD promises across various sectors. Previously, the concept of 'transformative changes' was introduced in the context of BD's academic research potential to revolutionise industries by enabling data-driven decision-making and innovation. Indeed, such transformative changes introduce a degree of uncertainty as organisations strive to evolve into data-driven entities. These efforts demand a high level of adaptability to fully leverage BD in the current market (McAfee & Brynjolfsson, 2012).

3.7 An Architecture-Centric Approach to Big Data

The burgeoning volume, velocity, and variety of data in modern BD landscapes necessitate a reevaluation of traditional system development methodologies, particularly within the realm of database design and data architectures. Traditional methodologies, anchored in the ANSI standard 3-tier DBMS architecture, have focused on relational models that emphasise structured, tabular formats and ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data integrity and transactional consistency (Reis & Housley, 2022). While these models offer a systematic approach to data management and evolution within defined schema constraints, they often fall short in accommodating the dynamic and complex nature of contemporary BD, which demands more flexible and scalable solutions (Elmasri, 2017).

Responding to these inadequacies, the industry has increasingly embraced agile

methodologies, facilitating iterative progress through practices such as Kanban and sprints of varying durations. This shift signifies a departure from linear, phase-gated development models towards more dynamic, responsive paradigms that better align with the unpredictable demands of modern software and data system development. Agile approaches are supported by a plethora of architectural references and patterns, providing practitioners with reliable roadmaps for constructing software systems. These contemporary systems typically adopt multitier or multilayered architectures, ensuring a clear demarcation between presentation, application, and data management functionalities (H.-M. Chen, Kazman & Haziyev, 2016a).

3.7.1 Relational Database Management Systems

When confronted with the idiosyncrasies of BD, traditional relational data models, despite their sustained dominance since the 80s with technologies like Microsoft SQL servers, MySQL, and Oracle databases, manifest palpable inadequacies (Dehghani, 2022). These deficits become apparent in various ways, particularly in the essential need for horizontal scaling.

In BD architectures, the software and system requirements extend beyond mere parallelization, necessitating more advanced approaches such as clustering to achieve optimal scalability. Additionally, these architectures need to support the heterogeneous nature of data, which spans from unstructured to semi-structured forms. RDBMS, due to their preference for centralised and rigid structures, often struggle to manage these varied data forms effectively. An illustrative example is provided by MySQL (Oracle Corporation, 2023), a widely-used traditional RDBMS.

MySQL excels at storing and querying structured data within a predefined schema. However, when tasked with managing large volumes of unstructured data, such as text from social media feeds or log files, MySQL's efficiency diminishes. Its rigid

schema-based architecture is not inherently suited for data that lacks a fixed structure, leading to complexities in data integration and querying processes (Rodríguez-Mazahua et al., 2016).

In response, a number of innovative technological solutions have emerged, including Presto, a distributed SQL Query Engine (Presto, 2023); Airflow, an instrumental platform for authoring, scheduling, and monitoring data pipelines (Apache, 2023a); and Hadoop, epitomising open-source distributed computing (Apache, 2023c). These developments underscore several limitations of traditional RDBMS in the BD milieu:

1. **Data Volume**: Traditional RDBMS encounter difficulties when dealing with datasets of a petabyte scale. Their primarily centralised design, optimised for vertical scaling, struggles with efficiently managing extensive data volumes (Nasser & Tariq, 2015).

2. **Data Variety**: These systems, originally developed with a focus on structured data, face challenges in processing semi-structured and unstructured data, due to their inherent design limitations in accommodating diverse data formats (Dehghani, 2022).

3. **Data Velocity**: Modifying traditional RDBMS to handle high-velocity data streams typically results in compromised efficiency, as these systems are not natively designed for rapid data ingestion and real-time analysis (Rad & Ataei, 2017b).

4. **Data Scalability**: The fundamental architecture of traditional RDBMS, which predominantly limits horizontal scalability, leads to performance issues when scaling up to meet the demands of BD's distributed processing requirements (Bhadani & Jothimani, 2016).

In summary, the preceding discussion has aimed to underscore the limitations of

traditional SQL databases when dealing with Big Data's unique demands. It highlighted the significant challenges these databases face, such as scalability and real-time processing, which are not issues they were originally designed to tackle.

While traditional RDBMS systems have shown limitations in handling BD, the data management landscape has evolved to include more sophisticated architectures designed to address these challenges. Data warehouses, modern data warehouses, and data lakes have emerged as solutions to manage and analyse large volumes of diverse data (Serra, 2024a).

3.7.2 Data Warehouses

Data warehouses represent a significant advancement over traditional RDBMS in handling large-scale analytical workloads. These systems are designed to consolidate data from various sources into a centralised repository, optimized for querying and analysis rather than transactional processing (Serra, 2024a). They typically employ a structured, schema-on-write approach, which ensures data consistency but can limit flexibility when dealing with diverse data types. An example of a traditional data warehouse is Oracle's Exadata, which combines hardware and software to deliver high-performance data warehousing capabilities (Serra, 2024a).

Despite their advantages, traditional data warehouses face several limitations in the context of BD (Russom, 2014):

- Struggle with the variety and velocity of BD, often requiring significant ETL processes
- Limited scalability, especially for petabyte-scale datasets
- Difficulty in handling unstructured or semi-structured data
- High costs associated with hardware and licensing for large-scale deployments

3.7.3 Modern Data Warehouses

The concept of modern data warehouses has emerged to address the limitations of traditional data warehouses in the context of BD (Krishnan, 2013). These systems incorporate advanced features such as:

- Scalable, cloud-based infrastructure to handle massive data volumes
- Support for semi-structured and unstructured data
- Real-time data ingestion and analysis capabilities
- Advanced analytics and machine learning integration

Modern data warehouses aim to combine the reliability and consistency of traditional data warehouses with the flexibility and scalability required for BD workloads (Serra, 2024a). A prominent example of a modern data warehouse is Snowflake, which offers a cloud-native, fully managed data warehouse solution with separate storage and compute layers for enhanced scalability and performance (Serra, 2024a).

While addressing many challenges of traditional data warehouses, modern data warehouses still face some limitations:

- Potential for high costs, especially with increased data volumes and compute requirements
- Scalability Bottlenecks: Despite improvements in scalability, the centralised nature of data warehouses can still create bottlenecks as data volumes and complexity grow exponentially.
- Data Ownership and Domain Expertise: Centralisation often separates data from domain experts, leading to a loss of context and potential misinterpretation of data.

- Lack of Agility: The centralised model can impede the agility of individual teams or domains to innovate and evolve their data models independently.
- Data Quality Issues: With a central repository, there's a risk of propagating data quality issues across the entire organization if not caught early.
- Increased Complexity: As the central data warehouse grows, it becomes increasingly complex to manage, potentially leading to longer development cycles and reduced flexibility.

3.7.4 Data Lakes

Data lakes represent a paradigm shift in data architecture, designed to address the variety and volume challenges of BD. Unlike data warehouses, data lakes employ a schema-on-read approach, allowing for the storage of raw, unprocessed data in its native format (Serra, 2024a). This approach offers several advantages:

- Flexibility to store any type of data, structured or unstructured
- Scalability to accommodate massive data volumes
- Cost-effectiveness, as data can be stored in its raw form without pre-processing
- Support for diverse analytical workloads, including machine learning and artificial intelligence applications

Despite their flexibility, data lakes also face challenges (Gorelik, 2019):

- Risk of becoming "data swamps" if not properly managed, making it difficult to derive value from stored data
- Challenges in maintaining data quality and consistency without a predefined schema

- Potential for increased complexity in data governance and access control
- Requirement for specialized skills to effectively manage and analyse data in a lake environment
- Majority of the points mentioned for the modern data warehouses in the previous section such as data quality issues, data ownership, lack of agility, data quality issues and increased complexity

These limitations point to the need for more advanced, flexible, and scalable architectures capable of handling the full spectrum of BD challenges. A detailed analysis of these limitations and the emerging architectural solutions is presented in Chapter 4, which explores reference architectures specifically designed for BD systems.

The following section will provide a detailed examination of BD's characteristics, thereby distinguishing it from conventional, or what is sometimes referred to as *small data*.

3.8 Big Data Characteristics

Thus far, BD has been defined, and the elements surrounding its adoption and associated challenges have been explored. However, a pertinent question remains: how does one differentiate BD from small data? At which juncture does data qualify as BD? Both academia and industry lack a universally accepted answer to this question, leading to varied interpretations by different practitioners (H. Wang, Xu, Fujita & Liu, 2016).

A consistent theme among various definitions of BD is that a data workload is classified as BD once it exhibits specific characteristics. The subsequent sections detail these characteristics.

3.8.1 Volume

The sheer volume of data can pose significant technical challenges. Architectures must demonstrate elasticity to accommodate data of varying magnitudes. While the process of storing and computing vast quantities of data has been addressed to an extent, achieving efficiency remains a challenge. Within this context, there exists an inclination towards scalable, configurable architectures that leverage distributed and parallel processing (H.-M. Chen, Kazman & Haziyev, 2016a).

3.8.2 Variety

Variety in the context of Big Data encapsulates the multitude of data formats present in modern computational environments. This characteristic distinguishes Big Data not by the sheer amount of data but by the diversity of data types it encompasses. These span structured, semi-structured, and unstructured formats, including, but not limited to, JSON and XML for semi-structured data; traditional databases and CSV files for structured data; and text, images, videos, and logs for unstructured data.

Historically, databases have been designed to handle structured data, notably through RDBMS that utilise tables, rows, and columns. However, these traditional systems were not always adept at managing the burgeoning unstructured data formats. Binary Large Objects (BLOBs) and Character Large Objects (CLOBs), which predate the 2000s, serve as data types to accommodate larger data formats. For instance, BLOBs are apt for multimedia formats such as videos, images, and sounds. These are not storage solutions per se but represent formats within databases.

The limitations of RDBMS in handling diverse data led to the advent of NoSQL databases. Contrary to misconceptions, NoSQL databases do not mark an evolution but a divergence from RDBMS, catering specifically to varied data structures. MongoDB,

for instance, is document-oriented; Redis is a key-value store; Cassandra is column-oriented; and Neo4J is designed for graph data (Han, Haihong, Le & Du, 2011). Importantly, these databases have distinct query languages and are often rooted in data structuring principles that existed before RDBMS.

In theoretical computer science, the CAP theorem, also named Brewer's theorem provides a theoretical underpinning to the trade-offs faced by distributed data systems . It postulates the mutual exclusivity of three system attributes: consistency, availability, and partition tolerance. NoSQL databases exemplify these trade-offs, with each database type emphasising different aspects of the CAP theorem (E. Brewer, 2012).

Furthermore, polyglot persistence emerged as a principle emphasising the necessity for diverse database systems, contingent on the data type and problem domain. This ideology, introduced by Neal Ford in 2006 (P. Khine & Wang, 2019), advocates for the strategic deployment of various database systems, programming languages, and tools, based on specific data needs (Sadalage & Fowler, 2013).

Taken altogether, the variety in BD systems is not a mere expansion of data volume but a complex mosaic of diverse data types. The evolution of databases, from hierarchical models to modern NoSQL varieties, exemplifies the industry's response to the challenges and opportunities posed by data variety.

3.8.3 Velocity

Velocity in the context of BD refers to the speed at which new data is generated and collected. This encompasses the challenges of processing real-time data and making rapid decisions, distinct from issues of volume or variety. Addressing these challenges necessitates specific architectural decisions. Among the prominent architectures developed to handle such challenges are the Lambda Architecture (Marz & Warren, 2015), designed for real-time data processing, and the Kappa Architecture (Kreps, 2014a)

optimised for stream processing.

3.8.4 Veracity

Veracity pertains to the trustworthiness and authenticity of the data. Poor quality data, which may be incomplete, unreliable, or outdated, poses significant challenges for BD processing. Ensuring data integrity requires rigorous data cleansing, modelling, and governance processes.

Data cleansing refers to the process of detecting and correcting (or removing) errors and inconsistencies in data to improve its quality. Modelling, in this context, pertains to the establishment of data standards and formats, ensuring consistent data structures. Governance encompasses the overarching set of processes, policies, and standards that ensure data quality throughout its lifecycle (Eryurek, Gilad, Lakshmanan, Kibunguchy-Grant & Ashdown, 2021).

Legal and ethical challenges arise when data is acquired from unauthorised or dubious sources, leading to concerns about privacy and data security. Thus, veracity encompasses both the intrinsic quality of the data and the context of its acquisition. Factors defining data trustworthiness include the collection method, the data origin, and the platform used for processing. Data consistency, on the other hand, can be gauged through statistical reliability measures (Demchenko, Grosso, De Laat & Membrey, 2013).

Addressing veracity in BD involves several essential components:

- Ensuring data sources are accountable and authentic
- Guaranteeing the platform's trustworthiness
- Validating the origin of data
- Establishing clear data lifecycle processes

- Integrating data from various sources effectively
- Ensuring timely data availability

While conventional database systems, which refer to databases developed based on longstanding and established data models, primarily relational databases, were designed for specific, often well-defined, data contexts, the diverse and dynamic nature of BD sources complicates data validation. This complexity emphasises the need for a rigorous architectural approach to ensure data veracity.

3.8.5 Value

Value is pivotal among the BD characteristics, as it is through value that the potential of data is fully realised. To derive value from data, an integrative approach to both storage and computing is essential. The concept of value in this context refers to the extraction of knowledge, contingent on various events or processes and their interdependencies. Such events or processes might manifest in diverse forms, including stochastic, probabilistic, regular, or random natures (Demchenko et al., 2013).

3.9 Critical Analysis of the Big Data Landscape

The narrative literature review presented in this chapter has explored the evolution of BD, its impact on architectures, applications across various domains, and the challenges associated with its adoption. While the review highlights the significant potential of BD and its transformative impact across industries, it also reveals several gaps, inconsistencies, and areas for further research.

One notable issue is the lack of consensus on the definition and characterization of BD. While the "5 Vs" (Volume, Variety, Velocity, Veracity, and Value) are commonly used to describe BD, there is no universally accepted definition. This inconsistency

can lead to confusion and hinder the development of standardized approaches to BD management and architecture.

Furthermore, while the literature emphasises the challenges organizations face in adopting BD, such as the complexity of data architectures and the rapid pace of technological change, there is a need for more empirical studies on the effectiveness of various BD architectures and management strategies. Many of the proposed solutions remain theoretical or based on limited case studies, making it difficult to generalize their applicability across different contexts.

The review also reveals a gap in the literature regarding the development of RAs for BD systems. While various architectural approaches have been proposed, they often lack the flexibility, scalability, and adaptability required to address the diverse and evolving needs of BD applications across different domains.

These gaps and challenges in the BD landscape underscore the importance of this research and the development of the Metamycelium architecture. By proposing a domain-driven, distributed RA that addresses the limitations of existing approaches, this work aims to contribute to the advancement of BD systems and enable organizations to more effectively harness the potential of BD.

Lastly, it is important to acknowledge the limitations of this narrative literature review. The review focused on a selected range of literature and may not have captured all relevant studies. Additionally, the interpretation of the findings may be subject to the author's bias and perspective.

3.10 Conclusion

In summary, this chapter provides a overview of BD, detailing its evolution, key characteristics and widespread impact across various domains. It highlights the transition from traditional data management systems to advanced architectures necessary for handling

the volume, variety, velocity, veracity, and value of BD.

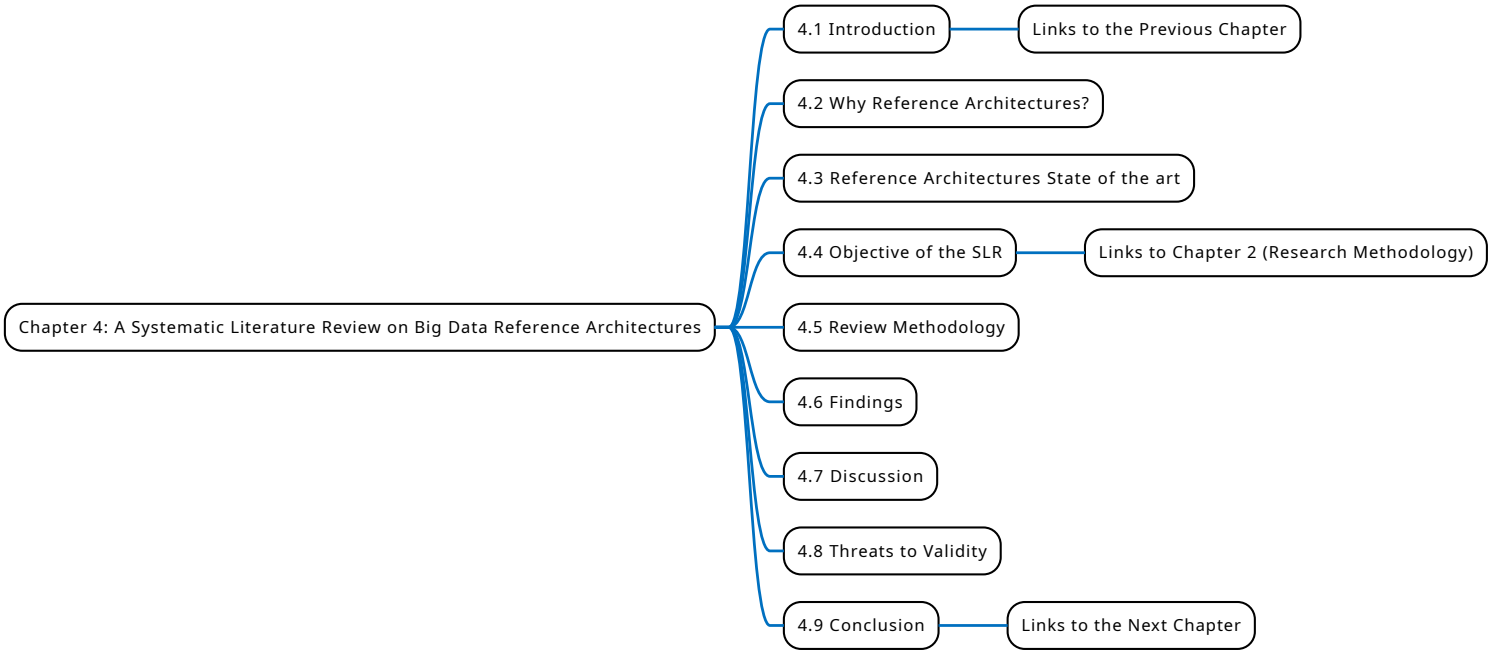
The chapter underscores the need for robust, scalable, and adaptable BD architectures in response to emerging technologies and diverse data types. Additionally, it discusses the business implications of BD adoption, emphasising the importance of aligning data insights with organisational objectives.

Additionally, this narrative literature review leads to an enquiry into why BD projects fail, which indirectly informs the development of Research Question 1 (RQ1). This sets the stage for exploring domain-driven distributed RAs to address the challenges and opportunities presented by BD.

In next chapter, a SLR on BD RAs is explored, delineating the fundamental components of these RAs, their patterns and their limitations.

Chapter 4

A Systematic Literature Review on Big Data Reference Architectures



4.1 Introduction

In the preceding chapter, the significance of BD was explained, spotlighting its omnipresence, associated challenges, and key defining characteristics. Specific components, including the exact definition of BD and criteria distinguishing a BD workload, were delineated. Building on this foundation, the current chapter embarks on a SLR on BD RAs.

The advancements in software technologies, digital devices, and networking infrastructures have bolstered users' ability to produce data at an unprecedented rate. In today's data-centric era, the continuous generation of structured, semi-structured, and unstructured data, when meticulously analysed, can reveal transformative patterns (Ataei & Litchfield, 2020).

As discussed in Section 1.2.2, the proliferation of data has ushered an ecosystem of technologies, BD being a notable member (Rada et al., 2017b). Despite growing interest, integrating BD into existing systems presents numerous challenges. Many organisations struggle to effectively implement BD and realise its benefits, as evidenced by various surveys.

These implementation challenges are further underscored by recent industry surveys. As previously noted in Section 1.3.2, these reports reveal low success rates in data strategy implementation, difficulties in transitioning to BD, and frequent abandonment of data-related projects due to processing delays.

BD challenges encompass a range of issues, from the lack of business context and rapid technological changes to organisational challenges, data architecture, and talent shortages. Though such challenges are not unique to BD, they are accentuated in its realm, given the complexities of BD engineering, real-time processing demands, scalability prerequisites, and data sensitivities.

Presently, many BD systems are developed based on ad-hoc and intricate architectural solutions, which can sometimes diverge from commonly accepted best practices in software architecture and software engineering (Gorton & Klein, 2015). Such inconsistency can lead to suboptimal decisions, hindering the evolution of BD systems. Given the limitations of an ad-hoc design strategy, this study strives to provide a comprehensive SLR on BD RAs, emphasising their potential in fostering a coherent approach to BD system development. This SLR aims to address RQ1 directly and RQ2 indirectly.

The chapter commences by underscoring the necessity of RAs and progresses to explore the challenges tied to the creation of RAs. Thereafter, it dissects the common architectural components of BD RAs, pinpointing their inherent limitations.

This chapter is structured as follows: “Why Reference Architectures?” discusses the indispensable role of RAs in grappling with complex challenges and highlights real-world examples (Section 4.2). The subsequent section, "Reference Architectures State of the Art", delves into the current BD system RAs, spotlighting their unique attributes (Section 4.3).

"Objective of the SLR" clearly outlines the research questions driving this SLR and the overarching aims (Section 4.4). "Review Methodology" details the research methods adopted for this SLR (Section 4.5).

Following this, "Findings" presents the distilled insights and key takeaways from the SLR (Section 4.6). A "Discussion" section engages in a rigorous analysis of the SLR’s results (Section 4.7). Recognising the importance of credibility, the chapter assesses "Threats to Validity", addressing the robustness and rigour of the findings (Section 4.8). Finally, a "Conclusion" encapsulates the essence of the SLR and its implications for the field (Section 4.9).

4.2 Why Reference Architectures?

As discussed in Section 1.2.4, conceptualisation of the system as an RA helps with understanding of the system's key components, behaviour, composition, and evolution, which in turn affect quality attributes such as maintainability, scalability, and performance (Cloutier et al., 2010a).

Therefore, RAs can be a good standardisation artefact and a communication medium that not only results in concrete architectures for BD systems but also provides stakeholders with unified elements and symbols to discuss and progress BD projects.

The practice of leveraging RAs for both system conceptualization and as a standardisation artefact is not new to practitioners of complex systems. In Software Product Line (SPL) development, RAs are utilised as generic artefacts that are instantiated and configured for a particular domain of systems (Derras et al., 2018a).

In software engineering, renowned IT corporations such as IBM have consistently advocated for RAs, considering them exemplary practices in addressing intricate system design challenges (Cloutier et al., 2010b). Similarly, in the realm of international standards, RAs frequently serve as tools to standardise emerging domains. Moreover, the BS ISO/IEC 18384-1 RA for service-oriented architectures (ISO, 2016) demonstrates the utility of RAs in creating standardised frameworks in specific fields.

4.3 Reference Architectures State of the art

Despite the evident advantages of using RAs and their potential to address the complex challenges of BD systems, there is a noticeable development gap in this area. Both academic literature and industry findings suggest a need for increased focus. This observation stems from a systematic review of the current body of knowledge and an exploration of available BD RAs (Ataei & Litchfield, 2020).

A notably comprehensive BD RA is published by the National Institute of Standards and Technology (NIST). The development of the NIST BD RA involved collaboration from the BD Public Working Group (NBD-PWG) with contributions from academia, industry, non-profit organisations, and other sectors. This RA was initiated following a White House directive in March 2012 and saw its release in October 2019 after considerable commitment of resources.

While significant resources have been channelled towards BD RAs, this doesn't immediately validate their inherent value. However, the investment underscores a perceived importance and highlights a potential need for in-depth research. Ambiguity surrounds the term RA. The differentiation between a 'concrete architecture' (a specific architectural solution for a system) and a RA is seldom elaborated upon, with interpretations appearing to vary across different domains.

These differing interpretations are evident in the literature. For instance, Cloutier et al. (2010a) proposes that RAs "capture the essence of existing architectures, and the vision of future needs and evolution to provide guidance in developing new system architectures." This perspective emerges from system engineering insights and a collaborative forum at Steven's Institute of Technology. In contrast, Muller (2008) defines RA as "an artefact that captures the architecture essence of a set of systems." This viewpoint is rooted in the field of product line engineering.

Similarly, Angelov et al. (2009) posits that "a reference architecture is a generic architecture for a category of information systems used as a foundation for designing specific architectures within that category." Bass, Weber and Zhu (2015) asserts that "a reference architecture relates a reference model to software elements and the data flows between them."

While definitions vary, a consistent theme emerges: RAs aim to encapsulate and reuse software engineering knowledge for a specific class of systems, particularly in relation to architecture. Notably, few discussions directly contrast RAs with concrete

architectures. The variability in definitions and the lack of a unified approach to BD RAs underscore the importance of a SLR of the current landscape. This gap in knowledge leads us to the objectives of the SLR.

4.4 Objective of the SLR

Considering the documented failure rate of BD projects, there is a need to explore the potential of RAs to enhance system development and BD architecture for better project outcomes. A previous SLR in this domain was conducted by Ataei and Litchfield (2020). This SLR found that RAs can be pivotal in addressing the complexities of BD system developments, serving as a guide, and promoting the application of software engineering knowledge and patterns.

Building upon the insights of the aformentioned SLR, the objective of the current SLR is to identify and aggregate the BD RAs from the existing body of knowledge, emphasise their architectural commonalities, and delineate their limitations. At the foundation of this SLR lies the broader research direction of the thesis, which concerns the architectural failure modes in BD projects and the role of RAs in addressing these issues, as outlined in Section 2.3.

While the thesis research questions provide a high-level strategic perspective, the research questions of this SLR delve into the operational details of BD RAs, presenting a tactical view that complements the overall research aims. While the thesis research questions offer a broad strategic view, the SLR research questions focus on specific operational aspects of BD RAs. This approach provides a detailed examination that supports the overall research goals. By combining high-level strategic analysis with specific empirical findings, we aim to conduct a thorough study of BD architectures.

The following research questions have been formulated to guide this SLR:

SLR–RQ1 Which BD RAs are currently available in both academic and industrial contexts?

SLR–RQ2 What are the primary architectural components of these BD RAs?

SLR–RQ3 What are the identified limitations of the current BD RAs?

These questions are formulated based on the identified gap in the body of knowledge pertaining to BD RAs. The questions not only strive to encapsulate current best practices in architecting BD systems but also aim to elucidate the limitations and challenges faced in the present-day context of BD RAs.

4.5 Review Methodology

This research adheres to the guidelines of PRISMA (Page et al., 2021). Furthermore, PRISMA-S (Rethlefsen et al., 2021) is incorporated to refine the search strategy. In addition, the guidelines from Kitchenham et al. (2015) for evidence-based software engineering and systematic reviews are also employed. Even though PRISMA offers comprehensive guidelines for conducting SLRs, its origin in the healthcare community brings with it assumptions that may not align perfectly with the needs of software engineering and information system researchers.

In response, Kitchenham et al. (2015) have contextualised many of these assumptions for the domain of software engineering, offering guidance especially for individual researchers and projects with limited resources.

PRISMA serves as the foundation for this research design, complemented by other methods to avoid bias, enhance transparency, and facilitate reproducibility of our systematic approach. An SLR is selected as the methodological approach as it offers a qualitative lens to advance knowledge and understanding around emergent topics and their associated elements. Additionally, SLR delivers a transparent and reproducible

procedure that identifies patterns, relationships, and trends and paints a holistic picture of the subject (Borrego et al., 2014).

The principal aim of this study revolves around assessing the current landscape of BD RAs, pinpointing their primary architectural components, highlighting prevailing theories, and discussing inherent limitations. This aim is methodically pursued across four phases. The initial phase involves stating the research questions, defining the exclusion and inclusion criteria, identifying and pooling relevant literature, and constructing a quality framework.

During the subsequent phase, study titles undergo evaluation against the established inclusion and exclusion criteria, followed by an assessment of the filtered studies' title, abstract, introduction, and conclusion. Thereafter, each study undergoes a comprehensive analysis against the criteria set in the quality framework. In the third phase, the chosen literature is systematically coded as per the research questions. Finally, the findings undergo a thematic synthesis, and the resulting themes are elucidated.

This research extends upon the SLR conducted by Ataei and Litchfield (2020) by encompassing the years 2020 to 2022. Contrasting with the work of Ataei, the present study employs thematic synthesis, intending to provide a more granular examination of BD RAs and their characteristics.

4.5.1 Identification

The first phase of the SLR involves the adoption of PRISMA-S (Rethlefsen et al., 2021) to formulate a comprehensive multi-database search strategy. This extension of PRISMA furnishes a 12-item framework that augments transparency, systematicity, and minimises bias in the search approach. For this study, the following electronic databases are investigated: ScienceDirect, IEEE Explore, SpringerLink, AISeL, JSTOR, and ACM library. To achieve the objective of identifying all available literature on the

topic and ensuring no valuable research is missed, abstract and citation databases and search engines such as Google Scholar and Research Gate are also utilised.

Furthermore, a search of the grey literature is conducted. Grey literature refers to research outputs not formally published in academic books or journal articles. The topic of interest is "big data reference architectures". Using the search string `"big data" AND "reference architecture*"` on Google, the first 40 results are selected for screening. The search is executed in incognito mode' to mitigate the influence of any personalised customisation of the search results. Reference lists of selected studies undergo manual screening to pinpoint additional relevant research, ensuring the crucial component of completeness' for SLRs, as articulated by Kitchenham et al. (2015).

Moreover, academic platform search capabilities may differ, but the search strategy remains largely consistent. For example, when a platform does not support wildcards (like asterisks), the terms are searched in both singular and plural forms. A notable exception is SpringerLink, which does not accommodate bulk downloads of references in BibTex format. The keywords for the databases are:

- `("Document Title":big data) AND ("Document Title":RA) OR ("Document Title":Reference Architecture) OR ("Document Title":big data architecture)`

The inclusion of the term 'architecture' is due to the potential interchangeable usage of *RA* and *architecture*. An architecture at the abstraction level of an RA might be termed merely as an architecture. Therefore, it becomes imperative to distinctly define these terms and categorise studies based on these definitions. These definitions, alongside the findings, are presented in Section 4.6.

The initial search spans the years 2020 to 2022, considering that the work of Ataei and Litchfield (2020) encompasses 2010-2020. However, the years 2010 to 2020 are also included to ensure comprehensiveness. Most databases support year-range

selection, and a language limit is implemented through an advanced search with the aforementioned keywords.

To systematically gather evidence, databases are searched using the specified keywords, followed by bulk downloading of the BibTex files. Only SpringerLink, Google Scholar, and Research Gate deviate from this procedure. For SpringerLink, studies are downloaded in CSV format and then transitioned to BibTex via a custom script. For Google Scholar and ResearchGate, each study's bib file is manually crafted.

Upon creation of all bib files, they are consolidated into a singular bib file and imported into the JabRef software for deduplication. Initially, 172 studies are pooled, but six duplicates are identified and removed. Additionally, the foundational SLR for this study and another uncited paper are excluded. Conversely, 5 white papers and 4 website blogs are incorporated. By the phase's conclusion, 173 studies had been amassed.

4.5.2 Screening and Eligibility

The first stage of screening begins with assessing the title, abstract, and keywords of the pooled studies. For grey literature, only the title is considered. This assessment relies on specific inclusion and exclusion criteria. The inclusion criteria are as follows:

- Primary and secondary studies (including grey literature) between January first, 2010 and August first, 2023 on the topics of BD RA, BD architecture, and BD architectural components.
- Research that indicates the current state of RAs in the field of BD and demonstrates possible outcomes.
- Studies that are scholarly publications, books, book chapters, theses, dissertations, or conference proceedings.

- Grey literature, such as white papers, that includes information on BD RAs.

Studies with the following topics are excluded:

- Informal literature surveys without clearly defined research questions or research processes.
- Duplicate reports of the same study (a conference and journal version of the same paper).
- Short papers (fewer than five pages).
- Studies not written in English.

In the second stage, after excluding papers based on the criteria, and as suggested by Kitchenham et al. (2015), the studies undergo quality assessment. The quality of the evidence collected as a result of this SLR directly influences the quality of the findings, emphasising the importance of quality assessment.

However, assessing quality presents well-known complexities. Among the most fundamental are defining 'quality' and appraising the quality of conference papers, which often lack comprehensive details on research methodology and evaluation. In general, a study's quality relates closely to its research method and the validity of its findings. From this perspective, inspired by the works of Noblit, Hare and Hare (1988) on meta-ethnography and Dybå and Dingsøyr (2008), studies' quality is assessed by the degree to which the conduct, design, and analysis of research are prone to systematic errors or bias (Cumpston et al., 2019). The more bias in the selected literature, the higher the likelihood of misleading conclusions.

Given the diverse nature of software engineering and IS papers, and the challenge of defining quality in studies of various natures, the analysis initially considers several well-established checklists, such as the Critical Appraisal Skills Programme (CASP)

(Critical Appraisal Skills Programme, 2023), and JBI's critical appraisal tool (Munn et al., 2020). However, recognising the need for criteria tailored to software engineering and IS, this research refers to the checklist provided by Runeson, Andersson, Thelin, Andrews and Berling (2006) for software engineering case studies. Similarly, Dybå and Dingsøyr (2008) proposes quality criteria based on the CASP checklist for qualitative studies in software engineering systematic reviews.

Despite these resources, the challenge remains that this research comprises numerous study types that must adhere to a single checklist. To address this concern, a criteria set consisting of seven elements is developed. These criteria are informed by the CASP's recommendations for assessing qualitative research quality (Critical Appraisal Skills Programme, 2023) and by guidelines provided by Kitchenham et al. (2002) on empirical research in software engineering. The seven criteria test literature in four major areas that can significantly impact the studies' quality. These categories and their corresponding criteria are:

1. **Minimum quality threshold:**
    (a) Whether the study reports empirical research or is merely a report based on expert opinion.
    (b) Clear communication of the study's objectives and aims, including the rationale for undertaking the study.
    (c) Provision of adequate information regarding the research context.
2. **Rigour:**
    (a) Appropriateness of the research design to address the research objectives.
    (b) Use and appropriateness of the data collection method.
3. **Credibility:**

(a) Reporting findings in a clear and unbiased manner.

4. **Relevance:**

(a) The study's value for practice or research.

Collectively, these seven criteria measure the extent to which a study's findings might contribute valuably to the review. The criteria serve as a checklist, with each property being dichotomous, that is, yes' or no', and the assessment takes place in two phases. In the initial phase, the assessment focuses solely on the first major area: the minimum quality threshold. If a study surpasses this phase, the next assessment includes credibility, rigour, and relevance.

Another challenge encountered relates to the fact that a PhD thesis typically involves a single researcher, making it infeasible to apply statistical methods like Cohen (1960) κ and Krippendorff (1970) α. Instead, the test-retest approach, as suggested by Kitchenham et al. (2015), is employed.

Following this approach, papers undergo an initial assessment and then a subsequent assessment at a later time. A study's quality is deemed satisfactory if 75% of the responses are positive with at least 75% inter-rater reliability across responses, which encompasses feedback from both the test and retest phases.

It should be noted that this quality framework does not apply to grey literature, which undergoes assessment solely based on inclusion and exclusion criteria. During the identification phase of this SLR, a total of 138 pieces of literature from academia and 24 from grey literature were amassed. Some literature joins the pool through the process of forward and backward searching. For example, upon reviewing the NIST RA, additional references from Oracle, Facebook, and Amazon are incorporated into the literature pool.

In the screening phase, any literature not aligning with the inclusion and exclusion

criteria is discarded. For instance, if a paper is excessively brief and fails to address BD RA, its ecosystem, or its limitations, it gets excluded. This phase results in the exclusion of 50 papers. Subsequently, by evaluating studies against the quality framework, 21 academic studies and 12 from the grey literature pool are excluded. The process is visually represented in Figure 4.1.

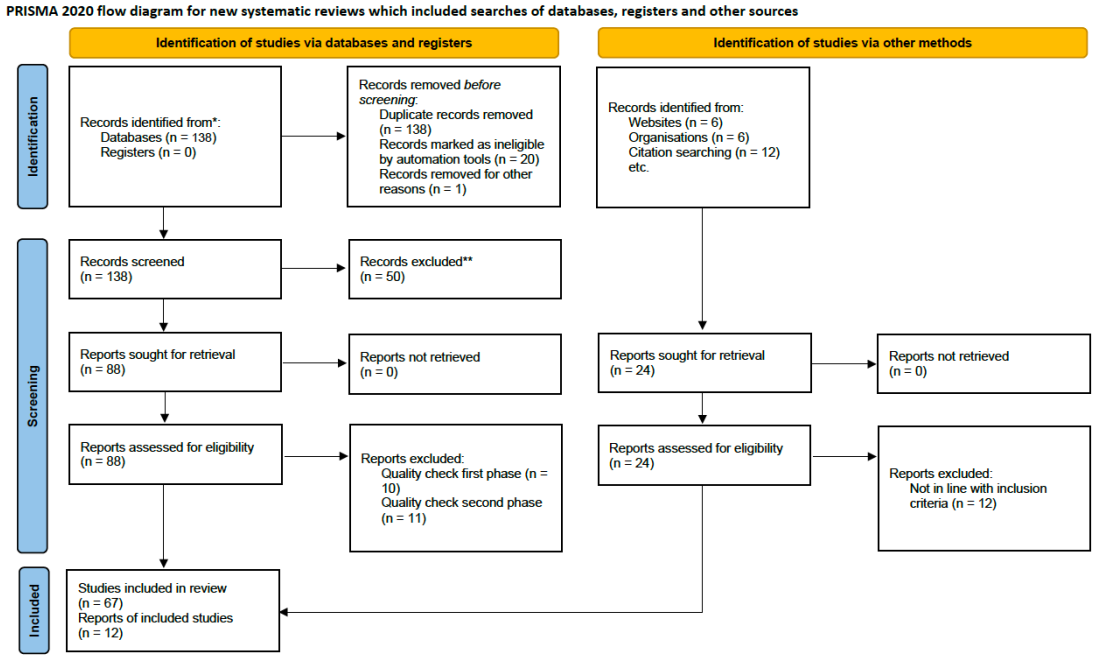


Figure 4.1: PRISMA Flowchart

As an outcome of this effort, 79 studies, consisting of proceedings, journal articles, book chapters, and white papers, were selected. From this collection, 33.3% originate from IEEE Explore, 5.2% from ScienceDirect, 24.5% from SpringerLink, 15.7% from ACM, and 21% from diverse sources like Google Scholar, Research Gate, and grey literature. The breakdown includes 30 journal articles, 29 conference proceedings, 12 book chapters, six white papers, one Master's thesis, and one PhD thesis. A majority of the articles (55%) stem from the period between 2016 and 2022, 33% from 2013–2016, and the remainder from 2010–2013. A visual representation of these statistics is provided in Figure 4.2.

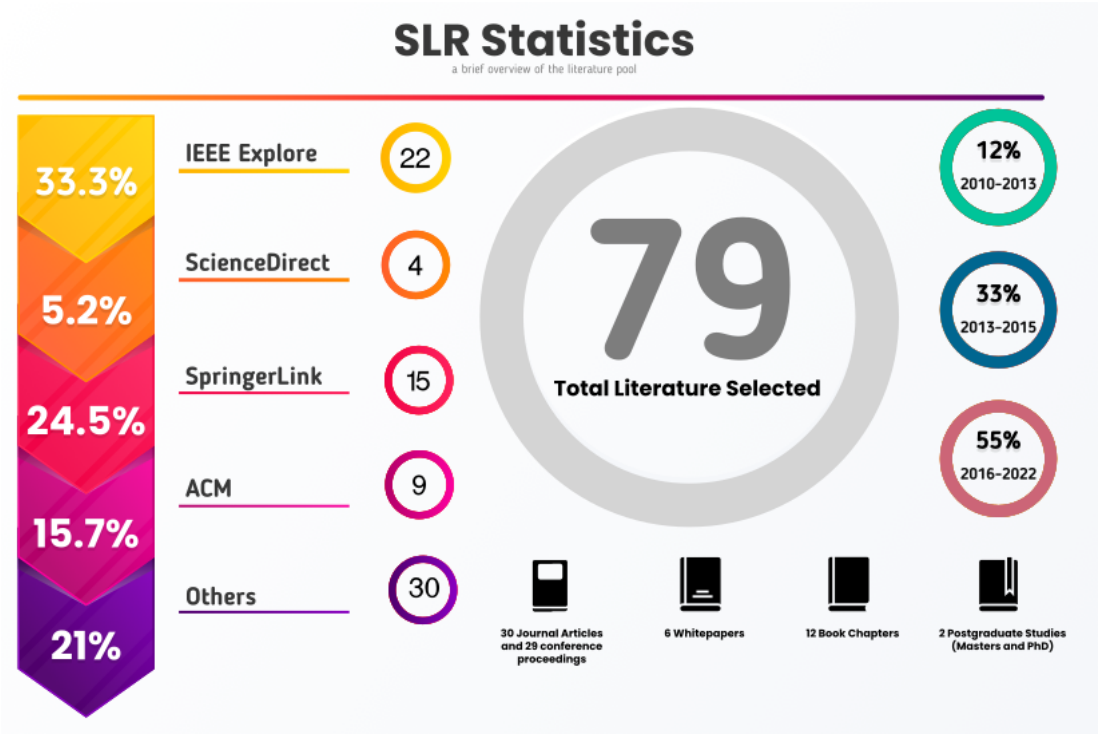


Figure 4.2: SLR Statistics

4.5.3 Data Extraction and Synthesis

At this stage, research questions get established, inclusion and exclusion criteria are defined and applied, the quality assessment framework is developed, and the synthesis of data commences. An essential component of this phase is data extraction, during which the essence of the studies is acquired in an explicit and consistent manner.

Prior to the synthesis of the data, guidelines proposed by Cruzes and Dyba (2011) for data extraction are employed. Data extraction begins by immersing in the entire pool of literature (V. Braun & Clarke, 2006). From there, a structured reading approach ensues, extracting three types of data:

1. Publication Details (author, title, year, etc.),
2. Contextual descriptions (industry, settings, technologies), and
3. Findings (results, the actual RA, events, etc.).

This process poses challenges as some studies fail to adequately describe the method, contextual information often lacks detail, and evaluation methods differ. Upon completing data extraction, the coding process begins. There are several potential approaches: a deductive or a priori approach (Miles & Huberman, 1994) and an inductive or grounded theory approach (Corbin & Strauss, 2014).

Both of these approaches are long-established and can be effective. The software Nvivo organises files, and an initial set of a priori codes based on research questions is created. These codes include:

1. BD RAs (**SLR–RQ1**)
2. BD RAs Architectural components (**SLR–RQ2**)
3. BD RAs limitations (**SLR–RQ3**)

During the coding, it becomes apparent that some fundamental areas might not have been well-established in academia and practice. Despite mentions of these concepts, descriptions are often sparse. Therefore, four additional codes are introduced:

1. Fundamental concepts of RAs
2. Benefits of RAs in BD system development
3. Common strategies for creating BD RAs
4. Challenges in BD RA development

Once all the literature is coded, categorisation by themes ensues. Themes help categorise the data. This process involves the integration of initial codes into higher-order ones, which sometimes requires the rearrangement and reclassification of codes. The end of this process occurs when no new themes emerge, and many of the initial themes get incorporated into higher-order themes.

The final step in data synthesis is the creation of a model based on higher-order themes to delineate relationships and address the research questions of this SLR. The outcome is a theory, alignment with previous theories, and the identification of relationships. A challenge in this phase arises from heterogeneity, attributed to the inclusion of grey literature and the variety of methodologies in software engineering research.

To ensure the robustness of the higher-order themes, the primary sources of variability are identified as follows:

1. Variability of outcomes (some RAs evaluated in practice, while others are merely compared against different RAs),
2. Variability in study designs (methodological diversity in software engineering and specifically in RA creation), and
3. Variability in study settings (often underreported contextual factors).

Finally, to enhance rigour, the trustworthiness of the synthesis is evaluated from three perspectives:

1. Credibility: Does the research align with research questions? Does the thematic synthesis encompass the data adequately?
2. Conformability: Is data extracted and coded correctly? Is there a consensus among researchers? Would readers concur with the approach?
3. Transferability: Are the findings generalisable? Can they apply in different contexts?

The detailed results of this process, including additional codes defined, themes identified, and specific examples illustrating data extraction challenges, are presented in the subsequent Section 4.5 (Findings). This upcoming section provides a comprehensive overview of the outcomes from our systematic review process.

4.6 Findings

This section maps findings against the research questions in several sub-sections. To enhance clarity, these sub-sections align with the research questions and models generated in the prior phase. The section starts by discussing fundamental concepts such as RAs and their significance in BD system development and subsequently delves into specific topics like current BD RAs and their limitations.

4.6.1 What are the Fundamental Concepts of RAs?

With the increasing complexity of systems, the principles and concepts of software architecture become essential tools to address the challenges faced by practitioners (Ataei & Litchfield, 2020). Representing a system using architectural concepts eases the comprehension of the system's essence, the properties it possesses, and its evolution. This representation influences quality attributes such as performance, maintainability, and scalability.

In recent years, IT architectures have been instrumental in the development and evolution of systems. These architectures assist in the maintenance, planning, development, and cost reduction of intricate systems (Martinez-Prieto, Cuesta, Arias & Fernández, 2015). An architecture provides clarity about the fundamental components of the system, guiding development to meet specific requirements (Sievi-Korte, Richardson & Beecham, 2019). This delineation produces manageable components to address various facets of the problem and offers stakeholders an abstract artefact for reflection, contribution, and communication (Kohler & Specht, 2019).

Successful IT artefacts often derive from an effective RA. Notable examples include the Open Systems Interconnection Model (OSI) (Zimmermann, 1980), Open Authentication (OATH) (OATH, 2007), Common Object Request Broker Architecture (CORBA) (Pope, 1998), and workflow management systems (WMS) (Greefhorst, 1999). In fact,

every system incorporates an architecture, which defines its overall qualities.

Definitions of an RA vary, but consistently highlight the significance of patterns. Some studies define RAs as predefined architectural patterns designed for use in specific business and technical contexts, complemented by supporting artefacts (Cloutier et al., 2010a). Within Software Product Line (SPL) development, RAs represent generic schemata adaptable for a certain class of systems (Derras et al., 2018a). In the realm of software engineering, RAs serve as artefacts that encapsulate domain-relevant concepts and qualities, segmenting solutions to foster effective communication among stakeholders (Klein, Buglak, Blockow, Wuttke & Cooper, n.d.).

Drawing upon the model from thematic synthesis, five major concepts of RAs emerge:

1. **RAs operate at the highest level of abstraction**: RAs encapsulate the essence of the practice, portraying elements vital for communication, standardisation, implementation, and maintenance of certain classes of systems. Therefore, RAs focus on high-level architectural patterns and do not delve into details such as specific frameworks or vendors. This positions RAs at a higher level of abstraction than concrete architectures.

2. **RAs focus on architectural qualities**: Due to their higher level of abstraction, RAs cater to a broader audience and context, guiding solution architects in deriving a concrete architecture in specific environments (Angelov et al., 2008; Stricker et al., 2010a). This means RAs prioritise architectural qualities.

3. **Stakeholders in RAs remain ambiguous**: Stakeholders, critical to the creation of the overall product, vary and include developers, designers, product owners, data scientists, and business analysts (Geerdink, 2013). However, the generic nature of RAs makes it challenging to predefine all stakeholders. Since RAs provide generalised solutions rather than context-specific ones, including specific

stakeholders could reduce their efficacy (Ataei & Litchfield, 2020; W. L. Chang & Boyd, 2018).

4. **RAs advocate for common standards**: The design of an RA typically follows existing architectural patterns, established best practices, and literature. Thus, RAs transmit standardised approaches that mitigate known challenges, promote reuse, and simplify complexities.

5. **RAs serve as effective tools for system development and communication**: RAs, encapsulating industry best practices and often comprising architectural descriptions and standards, prove invaluable for system development and communication.

4.6.2 How can RAs Help BD System Development?

Despite the high failure rate of BD projects, IT giants such as Google, Facebook, and Amazon develop exclusive BD systems with intricate data pipelines, data management, data procurement, batch and real-time analysis capabilities (Kohler & Specht, 2019). These companies, with their vast resources, attract top talent globally to handle the complexity inherent in the development of BD systems. However, many organisations strive to benefit from BD analytics without such advantages.

BD systems deviate from traditional small data analytics paradigms, introducing various challenges including rapid technological changes (H.-M. Chen, Kazman, Garbajosa & Gonzalez, 2017), system development, and data architecture challenges (H. V. Jagadish et al., 2014), along with organisational challenges (Rada et al., 2017b). Furthermore, BD systems are distributed by nature and account for multiple types of data processing, typically batch and stream processing. Coupled with the intricacy of maintaining and scaling data quality, metadata management, data catalogues, data dimension modelling, and data evolvability, BD system design proves intricate.

BD encompasses more than merely a large volume of data; characteristics such as velocity, variety, veracity, and variability present substantial challenges. These challenges, although not unique to BD systems, become exacerbated for several reasons:

1. Distributed scaling is required to address batch and stream processing demands.
2. There is a need for real near-time performance (stream processing).
3. Complex technology orchestration is required to create effective communication channels between components and data flow.
4. Continuous delivery is required to continually disseminate patterns and insights into various business domains (DataOps).
5. Two approaches are required for data processing, stream and batch processing; or fast and delayed processing.
6. Managing metadata at scale is crucial.
7. Modelling dimensions for an evolving schema presents difficulties.
8. Engineering data privacy is a priority.

Understanding the core fundamentals of BD systems is essential for addressing these challenges. Various literature describes BD as a combination of methodology (workflow, organisation), software engineering (data engineering, storage, and more), and analysis (mathematics, statistics) (Akhtar, Frynas, Mellahi & Ullah, 2019; Rad & Ataei, 2017a). It becomes evident that technology orchestration and data architecture play a central role in BD system development and maintenance.

RAs, rooted in this understanding and based on the results of the SLR synthesis, are considered effective artefacts. They streamline component delineation, interface

definition, technology orchestration, variability management, data architecture, scalability, and maintenance of BD systems (W. L. Chang & Boyd, 2018; Nadal, Herrero, Romero, Abelló et al., 2017). The role of RAs is to forge an integrated environment where fragmented system processes are optimised, adaptability to change is secured, and the delivery of architectural strategies is upheld.

Most scholars and practitioners concur that challenges surrounding BD software engineering and system development are significant (*Building a High-Performance Data and AI Organization*, 2023). The adoption of RAs for BD systems is warranted. Beginning with a well-established RA ensures that architects can access a pre-designed orchestration of components, interfaces, inter-communications, and variability points. They can then align them with the organisation's capability framework, desired quality attributes, and business drivers, eliminating the need to construct a new architecture from disparate components.

Implementing RAs to tackle intricate challenges has proven successful in Database Management Systems (DBMS) (Piñeiro et al., 2019) and Distributed Database Management Systems (DDBMS) (Rahimi & Haug, 2010).

4.6.3 What are Some Common Approaches to Creating BD RAs?

Findings from this study indicate a limited number of frameworks available for the design and development of RAs. A frequently used approach is the Empirically Grounded Reference Architectures by Galster and Avgeriou (2011a). This research methodology gains recognition for its emphasis on empirical validity and foundation.

This methodology is comprised of six steps which are respectively: 1) Selecting the type of the RA, 2) Selection of the design strategy, 3) Empirical acquisition of data, 4) Construction of the RA, 5) Enabling RA with variability, 6) Evaluation of the RA.

Another notable contribution in this domain is a framework for the analysis and

design of software RAs by Angelov et al. (2012). The framework employs a multi-dimensional classification space to categorise RAs and presents five major types. Developed to support RA analysis concerning their architectural specification/design, goal, and context, the method identifies three main dimensions, each with corresponding sub-dimensions of design, goal, and context.

These dimensions and sub-dimensions stem from interrogatives such as 'why', 'where', 'who', 'when', 'what', and 'how'. The question 'why' focuses on the RA goal, while 'who', 'when', and 'where' address context, and 'how' and 'what' concentrate on design dimensions. This framework divides RAs into two primary categories: facilitation RAs and standardisation RAs.

Volk, Bosse, Bischoff and Turowski (2019) employ the Software Architecture Comparison Analysis Method (SCAM) to evaluate RAs based on their applicability. The outcome of this research is a decision-support process for BD RA selection. Notably, two frequently observed standards are ISO/IEC 25010, used to select quality software products for RAs (Iso, 2011), and ISO/IEC 42010 for architecture description (International Organization for Standardization (ISO/IEC), 2017).

Evidence from this SLR reveals that many researchers and practitioners rely on informal architectural description methods, such as boxes and lines, with a notable exception in Geerdink (2013). Geerdink employs ArchiMate (Josey, Lankhorst, Band, Jonkers & Quartel, 2016), a formal and standardised architectural description language endorsed by ISO/IEC 42010. Informal modelling methods might lead to discrepancies between system design and implementation (Zhu, 2005), lack adherence to established standards, and do not further the development of modelling methodologies.

Thus, emphasising the use of standard architectural description languages for discussing and illustrating ontologies becomes evident. In conclusion, Geerdink (2013) utilises Hevner's IS research framework (A. R. Hevner et al., 2004) to develop an RA, aligning with the perspective that a BD RA is an information system artefact grounded

in existing literature and business requirements.

4.6.4 Challenges of Creating BD RAs

Among the challenges of developing RAs, evaluation stands out as particularly significant (Maier, Serebrenik & Vanderfeesten, 2013). Galster and Avgeriou (2011a) indicate that the fundamental pillars of evaluation are the correctness and the utility of the RA, as well as its adaptability and instantiation efficiency.

RAs and concrete architectures exhibit different levels of abstraction and possess distinct qualities. Although many well-established evaluation methods exist for concrete architectures, such as Architecture Level Modifiability Analysis (Bengtsson & Bosch, n.d.), Scenario-based Architecture Analysis Method (Kazman et al., 1994), Architecture Trade-off Analysis Method (Kazman et al., 1998a), and Performance Assessment of Software Architecture (Williams & Smith, n.d.), none of these methods can be directly applied to RAs.

For example, ATAM depends on stakeholder participation in the early stages for the creation of a utility tree. Given the high level of abstraction inherent to RAs, identifying a clear group of stakeholders at this stage becomes challenging. Moreover, many evaluation methodologies employ scenarios, but due to the abstract nature of RAs and their potential applicability in various contexts, creating valid scenarios proves difficult. This leads to either the development of a few general scenarios covering all aspects or the creation of numerous specific scenarios for various RA facets. Both approaches introduce potential threats to validity.

Given the issues highlighted above, existing architecture analysis methods appear insufficient for evaluating RAs. Various research endeavours aim to address this shortfall. For instance, Angelov et al. (2008) adapted ATAM to better suit RAs by inviting industry representatives to participate in the evaluation process. This revised process entailed

selecting various contexts and defining corresponding scenarios. However, identifying the appropriate candidate and involving them in the process often proves challenging, which in turn threatens the validity and generalisability of the resulting theories.

In a study by Maier et al. (2013) conducted at Eindhoven University of Technology, the RA evaluation involved comparing it against existing reference and concrete architectures described in industry whitepapers and reports. Similarly, Galster and Avgeriou (Galster & Avgeriou, 2011a) recommended reference implementations, prototyping, and an incremental approach for RA validation.

From the findings of this SLR and through examining approaches by Bosch (2000), Avgeriou (2003b), and Derras et al. (2018b), it becomes evident that an RA evaluation framework can involve architectural prototype evaluation. This entails generating a concrete RA architecture and then assessing it using a robust method such as ATAM.

4.6.5 What are Current BD RAs?

To answer **SLR–RQ1**, 22 BD RAs have been identified: 14 from academia and 8 from practice. These RAs are listed in Table 4.1. S22 (NeoMycelia) is the reference architecture developed as part of this research. It is included here for completeness and to show how it compares with existing literature.

Table 4.1: BD RAs

ID	Title	Domain	Year
s1	Lambda architecture (Kiran, Murphy, Monga, Dugan & Baveja, 2015a)	Practice	2011
s2	IBM - Reference architecture for high performance analytics in healthcare and life science (Quintero, Lee et al., 2019)	Practice	2013

Table 4.1 (continued)

ID	Title	Domain	Year
s3	Microsoft - Big Data ecosystem reference architecture (Levin, 2013)	Practice	2013
s4	Oracle - Information Management and Big Data: A Reference Architecture (Cackett, 2013)	Practice	2014
s5	Towards a big Data reference architecture (Maier et al., 2013)	Academia	2013
s6	A reference architecture for Big Data solutions introducing a model to perform predictive analytics using Big Data technology (Geerdink, 2013)	Academia	2013
s7	A proposal for a reference architecture for long-term archiving, preservation, and retrieval of Big Data (Viana & Sato, 2014)	Academia	2014
s8	Questioning the Lambda architecture; Kappa Architecture (Kreps, 2014b)	Academia	2014
s9	Accelerating Secondary Genome Analysis Using Intel Big Data Reference Architecture. (Sikora-Wohlfeld, Basu, Butte & Martinez-Canales, 2014)	Practice	2014
s10	Reference architecture and classification of technologies, products and services for big data systems (Pääkkönen & Pakkala, 2015)	Academia	2015
s11	SAP - NEC Reference Architecture for SAP HANA & Hadoop (SAP, 2016)	Practice	2016
s12	Big data architecture for construction waste analytics (CWA): A conceptual framework (Bilal et al., 2016)	Academia	2016

Table 4.1 (continued)

ID	Title	Domain	Year
s13	A reference architecture for Big Data systems in the national security domain (Klein et al., n.d.)	Academia	2016
s14	Managing Cloud-Based Big Data Platforms: A Reference Architecture and Cost Perspective (Heilig & Voß, 2017)	Academia	2017
s15	A software reference architecture for semantic-aware Big Data systems; Bolster Architecture (Nadal, Herrero, Romero, Abelló et al., 2017)	Academia	2017
s16	Simplifying big data analytics systems with a reference architecture (Sang, Xu & Vrieze, 2017)	Academia	2017
s17	NIST Big Data interoperability framework (W. L. Chang & Boyd, 2018)	Practice	2018
s18	Extending reference architecture of big data systems towards machine learning in edge computing environments (Pääkkönen & Pakkala, 2020)	Academia	2020
s19	A Big Data Reference Architecture for Emergency Management (Iglesias, Favenza & Carrera, 2020)	Academia	2020
s20	ISO/IEC 20547-3:2020 BS ISO/IEC 20547 3:2020 Information technology. Big data reference architecture. Reference architecture (for Standardization (ISO/IEC), 2020)	Practice	2020
s21	Phi: A Generic Microservices-Based Big Data Architecture (Maamouri, Sfaxi & Robbana, 2021)	Academia	2021
s22	NeoMycelia: A software reference architecture for big data systems (Ataei & Litchfield, 2021b)	Academia	2021

Over the past years, the BD domain has garnered significant attention, particularly

in BD system development. Building upon the White House initiative mentioned in Section 4.3, in March 2012, the White House announced an initiative for BD research and development (Kalil, 2012). This initiative aimed to accelerate scientific and engineering discovery, enhance national security, and facilitate knowledge extraction from extensive and complex data sets (W. L. Chang, Grady et al., 2015). This project received support from six federal departments with an investment exceeding 200 million USD.

In June 2013, the NIST launched the NBD-PWG (W. L. Chang et al., 2015). This initiative saw widespread participation from various sectors, including practitioners, researchers, agents, government representatives, and non-profit organisations.

A notable outcome of this project was the NIST BD RA (NBDRA). The US Department of Defense states that the primary objective of NBDRA was to offer an authoritative BD information source to guide and regulate practice. This RA is among the most recent and comprehensive in the BD field. NBDRA consists of two fabrics that define five functional logical components interconnected by several interfaces. These components represent the interconnected nature of security, privacy, and management.

Along these lines, major IT corporations have also released their own BD RAs. This SLR identified eight BD RAs from the industry, primarily sourced from white papers. These papers originate from companies such as IBM, Microsoft, Oracle, SAP, and ISO. Among these industry-developed RAs, the Lambda architecture is one of the most discussed and studied among these RAs. Some BD RAs in practice were omitted as they were deemed outdated or insufficiently detailed. For instance, the RA published by Amazon on AWS' official documentation page (Amazon Web Services, 2024) was omitted as it lacks sufficient detail.

In academia, there have been a few contributions, such as a postgraduate master's dissertation (Maier et al., 2013) and a PhD thesis (Suthakar, 2017). Some universities, including the University of Amsterdam, have introduced their own BD architecture frameworks (W. L. Chang et al., 2015).

Numerous RAs have also been developed for specific domains. These architectures often appear in short journal papers, with many authors intending to elaborate further in subsequent publications. For example, Klein, Buglak, Blockow, Wuttke and Cooper (2016) introduced a BD RA for the national security domain, while Weyrich and Ebert (2015) focused on the IoT domain.

Despite the efforts in this field, a noticeable scarcity of comprehensive BD RAs exists. The aforementioned studies primarily offer brief discussions on RAs in specific domains without delving deeply into quality attributes, data quality, metadata management, security, or privacy.

4.6.6 Major Architectural Components of BD RAs

To address **SLR–RQ2**, RAs listed in Table 4.1 undergo review and comparison to highlight the common architectural components of BD RAs. Some RAs like the works of Klein et al. (2016) present in the form of a short paper, while others, such as NIST, provide comprehensive insight.

Many RAs draw inspiration from or base their structures on other RAs. For instance, the RA ISO ISO20547 (for Standardization (ISO/IEC), 2020) has driven many components from the RA published by NIST (W. L. Chang et al., 2015).This signifies the notion that RAs often derive their effectiveness from existing knowledge rather than original constructs.

In a systematic approach to this inquiry and following data extraction, all the components from the BD RAs listed in the Section 4.6.5 are tabulated in Table 4.2. These components are the exact names of the architectural components used in the diagrams presented in each study.

Table 4.2: BD RAs Components

RA	Components
s1	Streaming layer, batch layer, serving layer
s2	Applicatons, Frameworks and platforms, Software defined infrastructure, Compute and storage servers
s3	Data sources, Data transformation, Data usage
s4	Data sources, Data Integration, Information Management, Information Access
s5	Data sources, Data Acquisition and Recording, Information Extraction and Cleaning, Data Integration, Aggregation and Representation, Query Processing, Data Modeling and Analysis, Interpretation
s6	Import Engine, Processing Engine, Management Engine, Analytics Engine, Visualisation Engine
s7	Big Data Layer, Archive Layer, Storage Layer, Presentation Layer
s8	Data Source, Real-Time Layer, Serving Layer
s9	Access Manager, Intel Big Data Analysis Platform, Data Ingestion, Dat Sources
s10	Data Sources, Data Extraction, Data Loading and Pre-Loading, Data Processing, Data Storage, Data Analysis, Data Loading and Transformation, Interfacing and Visualisation
s11	Data Input sources, Data Processing Platform, Processed Data for Client
s12	Application Layer, Analytics Layer, Storage Layer, Data Sources
s13	Data Providers, Big Data Application Layer, Big Data Framework Provider, Data Consumers
s14	Data Generation, Data Streams, Data Storage, Stream Processing, Data Warehouse, Hadoop Cluster, Machine Learning, Presentation
s15	Batch Layer, Speed Layer, Semantic Layer, Serving Layer

Table 4.2: BD RAs Components

RA	**Components**
s16	Data Source, Data Integration, Data Analysis and Aggregation, Interface/Visualisation
s17	Data Provider, System Orcehstrator, Big Data Application Provider, Big Data Framework Provider, Security and Privacy Fabric, Management Fabric, Data Consumer
s18	Data Sources, Data Extraction, Data Loading and Preprocessing, Data Processing, Data Storage, Model Development and Interface, Data Transformation and Serving, Interacing and Visualisation
s19	Data Provider, Big Data Application Provider, Big Data Framework Provider, System Orchestrator, Management Fabric, Security and Privacy Fabric, Data Consumer
s20	Big Data Application Provider, Big Data Processing Layer, Big Data Platform Layer, Big Data Infrastructure Layer, Integration, Security and Privacy, System Management, Big Data Provider, Big Data Consumer
s21	Acquisition Layer, Refinment Layer, Scrutiny Layer, Training Layer, Insight Layer
s22	Gateway, Stream Processing Service Mesh, Stream Processing Controller, Monitoring, Service Discovery, Query Controller, Batch Processing Controller, Batch Processing Service Mesh, Event Backbone, Data Lake, Query Engine, Event Archive, Semantic Layer, Control Tower, MicroService, Sidecar, Event Queue

Each study opts for different terminology to describe its architectural components. There appears to be no standardised method for modelling BD RAs. Utilisation of architectural definition languages like Archimate remains infrequent, with many studies

opting for specifically defined ontologies depicted using boxes and lines. This non-standard approach complicates the understanding and comparison of these RAs, often necessitating translation between ontologies.

An automated text analysis via Nvivo on the names of these architectural components helps identify commonalities and patterns in word usage. The results from this analysis can be visualised in a word cloud, as seen in Figure 4.3.

Figure 4.3: BD RA Component Names Word Cloud

Among the names used for components, *big data application provider* (five occurrences) and *big data framework provider* (three occurrences) appear most frequently. This prevalence stems from some RAs being based on NIST BD RA (s17) and subsequently adopting its terminology. One term universally used across studies is *data consumer* and *data provider*. Furthermore, most studies prefer the term *layer* to group different components of the RA logically.

To thoroughly address **SLR–RQ2**, attention is directed towards the description of

these components, categorising them based on their functions. These categories include: BD Management and Storage, Data Processing and Application Interfaces, and BD Infrastructure.

BD Management and Storage

A prominent characteristic of BD is variety, necessitating distinct storage solutions. This need sometimes aligns with the term 'polyglot persistence' (P. P. Khine & Wang, 2019). For dynamic data, NoSQL databases such as MongoDB present a suitable option due to their non-tabular nature. When a complex relationship between entities arises, graph databases like Neo4J prove beneficial due to their tree traversal performance.

Selecting the appropriate database or databases constitutes a pivotal architectural decision, encompassing patterns for data access, storage, and caching. Specialists in distributed systems with expertise in micro-services architecture might choose the Command Query Responsibility Segregation (CQRS) pattern for high-performance event-driven applications (Márquez & Astudillo, 2018). Storage type and access pattern stand as two primary architectural components of BD systems.

The prevailing trend in BD RAs focuses on monolithic storage solutions such as data warehouses and data lakes. Traditional methods involving data staging, dimensional modelling, storage in data warehouses, and data marts as bespoke access layers appear less efficient in managing BD loads. Yet, variations of this approach persist.

Another prevalent architectural component in BD RAs is the data lake. A data lake serves as an ingestion framework, accommodating diverse data types, both internal and external. Data within the lake typically undergoes retrieval for transformation following the LET (load, extract, transform) approach, in contrast to the conventional Extract, Transform, Load (ETL) methods.

While Business Intelligence (BI) and BD differ in source data types in terms of granularity and structure, distinctions between data lakes and data warehouses also

exist. For data warehouses, relational databases often reduce flexibility for analysis, potentially incurring substantial costs. Conversely, data lakes allow the storage of varied data without the need for a pre-defined schema, enhancing flexibility. However, excessive flexibility can lead to misuse, with engineers potentially cluttering the data lake. Ensuring data governance and active metadata can mitigate such challenges (Dehghani, 2019).

This synthesis suggests that BD RAs draw from three main paradigms: 1) the enterprise data warehouse paradigm (including the modern data warehouses), 2) the data lake paradigm, and 3) the multi-modal cloud-based paradigm. Some RAs offer higher abstraction levels. In cases like S20, making assumptions about data pipeline nature and storage modality remains challenging. Only the resulting system can offer clarity regarding the paradigm to which the RA aligns.

Regarding BD management, certain cross-cutting concerns often go unnoticed. Many BD RAs do not adequately address security, privacy, metadata management, and data quality. Few RAs focus on security, like S13, while others emphasise metadata management, such as S15. A comprehensive exploration of BD cross-cutting concerns appears lacking.

Data Processing and Application Interfaces

BD systems often encompass two major data processing activities: stream processing and batch processing. Stream processing suits sensitive operations and time-sensitive tasks, like detecting fraudulent credit card activity. Batch processing suits extended data analysis, such as regression analysis.

The processing type needed for a specific architecture relies on the data's characteristics, mainly its variety, volume, and velocity. Different studies offer varied abstraction levels regarding data processing. Some studies, like S19, detail data processing pipeline processes, while others, like S15, abstract them to batch processing' or

stream processing'.

Moreover, two data processing categories emerge. One category employs separate architectural constructs for batch and stream processing, while the other processes both within a single architectural component.

BD interfaces are either presented solely as a 'serving or access layer' or as multiple components tailored to different requirements. Some RAs, like s22, clearly delineate ingress, egress, and inter-node interfaces, while others employ simplistic annotations.

BD Infrastructure

The concept of infrastructure features prominently in RAs. Different approaches exist for its communication. Some adopt a standard architectural description language, like s6, explicitly defining the technology layer. Others, like s21 and s22, imply infrastructure. Some use both infrastructure and platform layers, like s20 and s17.

BD infrastructure appears more as a layer than a component, outlining a potential computing and networking design for a BD system. This becomes essential as BD practitioners frequently architect distributed paradigms and horizontal scaling.

Consequently, issues like the CAP theorem, ACID and BASE transactions, data consistency, and service discovery emerge as potential architectural challenges. Should a BD system adopt an event-driven approach like Kafka, discussed in s22? Or should it adhere to REST-based communication? Addressing context switch overhead and service networking becomes imperative.

In conclusion, a BD infrastructure component emerges as a consistent pattern in various forms and strategies. While some might argue that infrastructure is an inherent architectural component of any system, designing for BD systems becomes more significant due to their distributed nature.

A notable omission concerns DataOps, crucial for automating and delivering data engineering workloads agilely. Many RAs seem not designed on entirely distributed

architectures, although BD systems can leverage this paradigm. An exception includes S22, drawing extensively from event-driven microservice architecture.

4.6.7 What are the Limitations of Current BD RAs?

To address **SLR–RQ3**, RAS collected for this SLR are appraised to identify limitations. This is summarised in Table 4.3.

Table 4.3: BD RAs Limitations

RA	Components
S1	Lambda architecture, developed in the early stages of BD development, lacks a comprehensive approach to data architecture and has several observed limitations. It does not address data quality, changes in the data landscape, or concerns like privacy, security, and metadata. The architecture offers minimal requirements for BD systems.
S2	This RA, designed specifically for healthcare and life sciences, does not consider data quality, security, or privacy. It bears a resemblance to monolithic n-tier architectures and relies on IBM specific solutions. There is no mention of data accountability or interoperability.
S3	This RA addresses a general scenario of BD analytics without attention to metadata or security. It encompasses data sources, data transformation, and data usage, but clarity is missing on data quality assurance and adaptability to changes. Moreover, the heavy influence of MapReduce may not yield optimal performance compared to newer methods like acyclic direct graphs used in Apache Spark and Tez.

Table 4.3 BD RAs Limitations

RA	Components
S4	A traditionally designed RA with three main phases: ingestion, processing, and provision. Underlying mechanisms facilitate 'right-time' data flow. However, the approach to data quality and changes in the data landscape is unclear. The use of data marts may affect modifiability. Governance methods are not evident.
S5	An enhancement of traditional data warehouse architectures, this RA includes components for stream processing. While privacy and metadata are addressed, it lacks security, data provenance, scalability, and modifiability. The approach to data ownership and data quality remains unclear.
S6	This RA seems to be portraying the bare minimum requirements for BD analytics with no mention of security, privacy, metadata, or data quality.. The RA is published as a short paper, thus not much detail is given.
S7	This RA provides with bare minimum components for data analytics, without any clear identification of stream processing. The RA is designed in a reductionist manner, without addressing privacy, security, metadata, and data quality. Data storage seems to be only associated to hardware, and thus it is unclear how data is evolved and scaled.
S8	Kappa is perhaps the predecessor of Lambda, aiming to address some of the limitations of it. The major difference between Kappa and Lambda is that Kappa has a unified processing layer for batch and stream process, which eliminates the complexity of maintaining two separate systems, and reduces costs. Nevertheless, this architecture, does not discuss metadata, security, privacy, data quality, and maintainability in details.

Table 4.3 BD RAs Limitations

RA	Components
S9	This BD RA, designed by Intel for healthcare applications, lacks detail and doesn't address cross-cutting concerns such as security, metadata, privacy, and data quality. The concept of access manager seems to be vague, as I could not understand how the access is managed. The artefact seems to be a simple instance of Hadoop ecosystem with some extra components added
S10	A comprehensive RA that aims to cover many aspects of data engineering. Nevertheless, I did not find any notion of metadata management, neither was there a discussion on security and privacy challenges. While this RA could work successfully for a regular BD workflows, it is unclear how data quality is met and how scalability is achieved.
S11	This RA is made up of three major phases, ingestion, processing and presentation. It is designed around Hadoop ecosystem, and provides with bare minimum necessary to conduct data analytics. I could not find a discussion on metadata management, security, privacy or data quality. The data pipeline is using a data warehouse, and uses it to communicate to the Hadoop side of things. I are not sure how unstructured data is handled, and how data lineage is achieved.
S12	This architecture is specifically designed for waste management, and seems to be using an approach similar to Kappa. The data takes a generic flow from data sources to application, without any clear identification of data quality, privacy and security concerns.

Table 4.3 BD RAs Limitations

RA	Components
S13	This RA is specifically designed for the security domain and seems to have a lot of inspiration from NIST BD RA. The RA is layed out in fabrics just like the NIST one, and unlike many others, does mention cross-cutting concerns such as security explicitly. However the concept of data ownership is not discussed, there's no mention of metadata or privacy, and the artefact evaluation is not extensive. That is, it is unclear how the derived solutions from this RA can scale.
S14	A generic RA that resembles the Lambda architecture, with stream and batch processing being processed in different nodes. This RA utilises data warehouses and Hadoop cluster for data processing. It was unclear how the security, privacy, metadata, and data quality is achieved. Maintainability aspects are not discussed as well.
S15	This RA extends the Lambda architecture by adding a semantic layer. It has a great focus on handling metadata in a right manner, but it does not seem to have any identification of other cross-cutting concerns such as privacy, security or data quality. It also adopts the idea of separate batch and stream layers, which can potentailly affect modifiability negatively and increase cost.
S16	This RA clearly segregates stream data from other data, and defines clear interfaces for ingestion of different data types. It then passes the data directly to a distributed storage (Hadoop's HDFS), and retrieves it later for deduplication and cleaning. While the RA seems to have addressed the minimum requirements of data analytics, it does not seem to address cross-cutting concerns such as metadata, security and privacy. It is also unclear how data quality is achieved.

Table 4.3 BD RAs Limitations

RA	Components
S17	This is perhaps the most comprehensive BD RA found in this SLR, and has been heavily funded by the government of the USA. While this RA is a good tool to facilitate open discussion, design structures, requirements, and operations inherent in BD, it is more of a high-level conceptual model of BD, rather than an effective BD RA. Some of the limitations witnessed in this RA is in its brief mention of metadata management (only discussed in lifecycle management), unclear approaches to attain data quality and data ownership, and potential monolithic coupling of components in BD application provider.
S18	This RA segregate data extraction and data loading phases and tend to adopt the idea of distinct stream and batch processing layers. Nevertheless, I could not find the identification of cross-cutting concerns such as privacy, security, or metadata management. I could not understand how data quality is achieved. There are also three storages designed, but it seems like all data will eventually be stored in one giant storage. This can potentially make modifiability harder and create a choke point.
S19	Derived from S17 (NIST BD RA), this RA is largely identical to S17 but tailored specifically for emergency management. The limitations discussed for S17 apply to this RA as well, so a separate explanation is not provided.
S20	This RA shares all the fundamental components with NIST BD RA, and seems to be very similar. However, the phrase fabrics seems to be changed to multi-layer functions. Therefore this RA, just like NIST is too abstract and leaves many architectural decisions unknown such as data storage, data quality assurance, and data ownership. It is unclear on how storage should be approached, and the overall structure resembles to a monolithic data pipeline architecture.

Table 4.3 BD RAs Limitations

RA	Components
S21	One of the few RAs that tend to absorb the concept of microservices into BD development. Nevertheless, the RA seems to be driven by the idea of one data lake for all data storage, which can be a daunting task to scale and maintain. The concept of metadata does not seem to be discussed, and other concerns such as security, privacy, data quality and data provenance are unclear.
S22	This RA absorbs a lot of patterns from microservices event-driven architectures and reactive systems and tend to absorb them into the BD development. While there's been a clear attention to cross-cutting concerns such as metadata and privacy, security does not seem to be well discussed in the study. The RA also tends to use data lake as the single source of storage which can be challenging to scale.

Except for one case (S22), all the architectures and RAs found as the result of this study, were designed with an with an underlying monolithic data pipeline architecture, with four major components being data consumers, data processing, data infrastructure, and data providers. To discuss the integral facets that embroil these architectures, one must look at the characteristics of these architectures and the ways in which they achieve their ends.

Findings from this SLR and deep analysis of the RAs found highlight 3 generations of BD architectures:

1. **Enterprise Data Warehouse:** this is perhaps one of the oldest approaches to business intelligence and data crunching and has existed even before the term BD was coined (Leonard, 2011). Usually developed as proprietary software, this data architecture pivots on enterprise data warehouses, ETL jobs, and data

visualisation software such as Microsoft Power Business Intelligence (BI). As the data sources and consumers grow, this architecture suffers from hard to main ETL jobs, and visualisations that can be created and understood by a certain group of stakeholders, hindering the positive impact of data on business. This also means, new transformations will take longer to be added to the workload, the system is monolithic and hard to scale, and only a few group of hyper-specialised individuals are able to operate the system. Moreover, data warehouses have been designed with different assumptions that cannot effectively handle the characteristics of BD.

2. **Data Lake**: to address challenges in the first generation of data architectures, a new BD ecosystem emerged. This new ecosystem revolved around a data lake, in a way that there are not as many transformations on the data initially, but rather everything is dumped into the data lake and retrieved when necessary. Although data lake architecture reached a higher level of success in comparison to the first generation of data architectures, it is still far from optimal. As data consumers and data providers grow, data engineers will be immensely challenged to avoid creating a data swamp (Brackenbury et al., 2018), and because there is usually no concept of data owner, the whole stack is usually operated by a group of hyper-specialised data engineers, creating silos and barriers for gradual adoption. This also means various teams' concerns will often go into data engineers backlog through an intermediary such as a business analyst, and they will not be in control of how and when they can consume the data they desire. Furthermore, data engineers are usually oblivious of the semantics and value of the data they are processing; they simply do not know how is that data useful or which domain it belongs to. This will over time decrease the quality of data processing, result in haphazard data management, and make maintenance and data engineering a

complicated task.

3. **Cloud Based Solutions:** Given the cost and complexity of running a data lake on-premise alongside the whole data engineering pipeline, and the substantial talent gap currently faced in the market (Rada et al., 2017b), the third generation of BD architectures tends to revolve around as-a-service or on-demand cloud-based solutions. This generation of architecture tends to be leaning towards stream-processing with architectures such as Kappa (J. Lin, 2017), or frameworks that unify batch and stream processing such as Apache Beam (Foundation, 2021) or Databricks (Inc., 2021). This is usually accompanied by cloud storage such as Amazon S3, and streaming technologies such as Amazon Kinesis. Whereas this generation tends to solve various issues regarding the complexity and cost of data handling and digestion, it still suffers from the same fundamental architectural challenges. It does not have clear data domains, a group of siloed hyper-specialised data engineers are running them, and data storage through a monolithic data pipelines soon becomes a choke-point.

To analyse the integral facets of these architectures, characteristics and methods employed by these architectures require examination. The process of transforming data into actionable insights in these architectures tends to follow a similar lifecycle:

1. **Data Ingestion:** Systems ingest data from various parts of the enterprise, encompassing transactional, operational, and external data. For example, in veterinary practice management software, the platform can ingest and persist transactional data such as interaction with therapeutics, number of animals diagnosed, and quantity of invoices created and medicines dispensed.

2. **Data Transformation:** Data from the preceding step undergoes cleansing for duplication, quality, and privacy policy considerations. This data then undergoes a

comprehensive enrichment process to support data analysis. For instance, a veterinary nurse's journey can be captured at every stage, enriched with demographics and animal breeds for regression analysis and aggregate views.

3. **Data Serving:** At this juncture, the data meets various needs, from machine learning to marketing analytics, business intelligence, product analysis, and customer journey optimisation. In the context of veterinary practice management software, the platform can offer real-time data through event backbone systems such as Kafka about customers who have procured and been dispensed restricted veterinary medicine (RVM) to ensure these transactions meet the conditions of the registration of these products.

This lifecycle represents a high-level abstract view of prevalent BD systems. Nevertheless, it underscores a critical point: these systems all operate on a monolithic data pipeline architecture, which tends to incorporate all data varieties within a single architectural framework. Hence, data from different logical domains become aggregated and processed collectively, complicating maintainability and scalability (Dehghani, 2019).

Architecture and system design following these methodologies can lead to challenging-to-maintain systems with significant costs. While not all such architectures necessarily fail, many face threats to their maintainability and scalability:

- *Data source proliferation*: As the BD system expands and incorporates more data sources, the capacity to ingest, process, and harmonise all this data in one location wanes.

- *Data consumer proliferation*: Organisations employing rapid experimentation methods such as Hypothesis-Driven Development and Continuous Delivery consistently present new use cases for data across different domains. This increases

data variability, and the overall workload for the data engineering team, thus prolonging the data serving process.

A noticeable limitation in many architectures is the insufficient discussion around metadata, which, in the context of BD, data architecture, and data engineering, refers to structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource, encompassing aspects such as data lineage, quality, and lifecycle management (Eryurek et al., 2021). For example, while one study (Nadal, Herrero, Romero, Abelló et al., 2017) highlighted the constraints of metadata management systems, it introduced a layer for metadata management as a discrete component.

Similarly, the NIST BD RA barely touches upon metadata. Except in a few instances, the importance of metadata remains underemphasised. Yet, metadata addresses challenges such as privacy, security, data provenance, data lineage, and linear analysis. Professionals in BD systems are now exploring large-scale metadata systems like metadata lakes.

From these observations, it becomes evident that a well-defined metadata layer can greatly benefit BD systems, aiding in simplifying complexities and potentially fostering the creation of data meshes (Dehghani, 2022). Moreover, white papers from prominent IT corporations like the ones published by Quintero et al. (2019) or Cackett (2013) seem to structure the RA around their offerings, which might limit its universality and influence architectural quality. In these documents, alternative technologies often remain unexplored, limiting architectural choices (Nogueira, Romdhane & Darmont, 2018).

Furthermore, there appears to be inadequate discussion on privacy and security. For instance, architectural components allowing for data scrubbing remain elusive, as do means to ensure security within data pipelines. Particularly with the global trend

towards enhanced privacy, BD architects face the challenge of designing in compliance with regional data privacy regulations such as General Data Protection Regulation (GDPR). These challenges, when juxtaposed with security issues, underscore the need for further research in BD RAs.

The architectures found as the result of this study typically comprise data processing pipelines, each designed for distinct functions within the data lifecycle. For instance, one pipeline might specialise in real-time data analytics, characterised by its ability to process streaming data for immediate insights, whereas another might focus on batch processing for historical data analysis, distinguished by its use of accumulated data to generate comprehensive reports.

Although each pipeline holds specific responsibilities, tight coupling exists. Paired with the *usual lack of contract* between upstream transactional systems and downstream analytical data pipelines, this coupling can increase the time to insight. Underlying this approach, when rapid experimentation is a priority, making feature and value delivery becomes challenging.

Using veterinary practice management software as an illustration, when a new class of animals needs integration for data analysis, the data engineering team must modify and expand the entire ingest, process, and serve pipeline. This implies overarching dependencies affecting various teams, delaying processes, and increasing maintenance complexity.

A significant concern with existing architectural approaches is the isolation of data engineering by teams lacking domain knowledge, reducing their efficiency. In contrast, when various stakeholders collaborate effectively, better decisions emerge, aligning technical and business requirements.

Taking together all the premises discussed in this section, the following summarises the limitations of current BD RAs:

Organisational Challenges: Centralised data ownership often resides with experts lacking comprehensive understanding of data semantics and usage. This centralisation creates a disconnect between those who understand the business context and those managing the data (Dehghani, 2022). Such misalignment can lead to misinterpretation and inefficient use of data resources (Serra, 2024a).

Workload management outside of primary data teams can cause delays and foster tribal knowledge. This siloed approach hinders cross-functional collaboration and impedes data-driven decision making (Hechler, Weihrauch & Wu, 2023). The separation between data producers and consumers creates bottlenecks in data flow and utilization, often resulting in a lack of domain-specific knowledge within the central data team (Company, 2019-2024).

Architectural Challenges: A key architectural challenge in BD RAs is the prevalent use of monolithic data pipelines, where diverse data types are aggregated and processed within a single framework. This centralisation can lead to scalability issues as data volume and variety increase, making it difficult to efficiently process and derive timely insights. Additionally, monolithic architectures are complex and tightly coupled, hindering maintenance and modifications, as changes in one part can ripple through the entire system, leading to increased costs and delays.

Another significant challenge is the isolation of domain knowledge. In monolithic architectures, data is often managed by specialized teams who may not possess a deep understanding of the specific business domains the data relates to. This can result in suboptimal data modeling, transformation, and analysis decisions, as the nuances and context of the data might be overlooked. Furthermore, monolithic architectures lack agility and adaptability to evolving business needs, making implementing new features or modifications a time-consuming and disruptive process.

Technological Challenges: A significant technological challenge in current BD RAs is the prevalence of tools and frameworks built on centralised architectures. While these tools offer a wide range of functionalities for data processing and analysis, they often perpetuate the traditional notion of a single source of truth for data. This centralised approach can lead to bottlenecks in data access and integration, as all data needs to be funneled through a single point. Additionally, it can limit flexibility and scalability, as the central system may not be able to handle the increasing volume and variety of data generated by modern applications.

Operational Challenges: In many organizations, a top-down approach to data governance is common. This centralised control can stifle the agility and responsiveness required for effective data analysis. It can slow down the process of generating insights, as decisions need to go through multiple layers of approval. This can also create friction and resistance among teams, as they may feel their autonomy and ability to experiment are being curtailed. In the fast-paced world of BD, where rapid experimentation and innovation are crucial, such operational constraints can significantly hinder progress.

Principal Challenges: A fundamental challenge in BD RAs is the traditional mindset of treating data as a static asset rather than a dynamic product. This perspective can lead to a focus on data storage and management, neglecting the continuous evolution and improvement of data quality. When data is not viewed as a product with its own lifecycle, it can become stagnant and less valuable over time. This can result in outdated or inaccurate insights, hindering decision-making processes and limiting the potential value that can be derived from data.

Infrastructure Challenges: Current BD architectures often rely on a combination of disparate data sources and technologies, leading to fragmented and non-integrated

data analytics approaches. This fragmentation can make it difficult to gain a holistic view of data, as different systems may have incompatible formats, schemas, or access protocols. As data volume grows, this lack of integration can become a significant bottleneck, hindering the ability to efficiently process, analyse, and derive insights from data. Additionally, managing and maintaining a complex infrastructure with multiple components can be costly and time-consuming, further exacerbating the challenges of scaling and adapting to evolving business needs.

4.7 Discussion

In this section, a detailed summary of the findings from the SLR on BD RAs is provided, highlighting the current landscape and examining the implications of these findings. The research methodology has enabled an investigation into the state of the art in BD RAs, revealing insights into their development, implementation, and maintenance. To the best of available knowledge, this study represents the first comprehensive SLR of BD RAs in the academic domain, addressing a gap in the literature. Despite the important role RAs play in the efficient development and ongoing support of BD systems, the findings indicate a notable shortfall in focused academic attention towards these artefacts. This oversight underscores the need for further research and discussion in this area.

The study most closely related to this study in the domain of comparing and analysing BD architectures is the one conducted by Volk et al. (2019). However, their research does not focus on BD RAs but seeks to craft a decision support system for selecting BD RAs. Its approach to BD architectures is somewhat superficial, not aiming for a systematic collection.

Futhermore, the NIST BD RA (S17) researchers sourced a collection of white papers from the BD Public Working Group as foundational material. These documents, however, do not provide detailed BD RAs. Instead, they serve as conceptual proofs

provided by different group members for the sake of comparison.

The findings of this SLR reveal that advancements in the domain of BD RAs are not consistent. While extensive research exists in data warehousing, Artificial Intelligence (AI), data science, and IOT, the field of data engineering requires further exploration. Notwithstanding the numerous established methodologies for managing vast data volumes or addressing the dimensionality of intricate datasets, the overarching organisation of BD technologies, the architecture, necessitates heightened interest from both academia and industry.

A significant proportion of the assessed BD RAs operate on a monolithic data pipeline with central storage, a framework that poses challenges in scalability and maintenance. Questions arise: How are measures taken to prevent data lakes from devolving into data swamps? How can a cohort of highly specialised data engineers, responsible for the data pipelines, remain cognizant of data consumption and ensure data quality? What methods ensure data interoperability and institutionalised data ownership?

Should a software engineer opt to adjust a field in a specific entity's schema for a novel feature, what repercussions might this hold for the data engineering process, and how would such changes be communicated? The growing availability of data for companies leads to challenges in maintaining this data through monolithic data architectures. As data volumes expand, the ability of monolithic systems to efficiently process and analyse this data diminishes.

In comparison to the evolving demands for more flexible and scalable BD architectures, the current landscape of BD RAs often mirrors traditional data warehousing approaches. This observation is noteworthy, as some architectures have adopted data marts, proposing them as solutions for BD challenges, albeit through the use of modern technologies. Moreover, there is a trend among certain architectures towards utilising data lakes, intended to support data analysts and BI practices.

This adherence to traditional methodologies raises concerns because it may not fully accommodate the dynamic nature and scalability requirements of contemporary BD environments. The reliance on monolithic paradigms could potentially hinder the ability of organisations to effectively manage and derive insights from vast and varied data sources, thereby limiting the agility and responsiveness needed in today's data-driven decision-making processes.

Neither the effort to integrate BD analytics into data warehouses nor the endeavour to bolster business intelligence using data lakes appears sustainable or scalable (Dehghani, 2022). Consequently, the emphasis shifts to the necessity of upcoming research trajectories focusing on decentralised and distributed BD RAs.

In the SLR conducted, significant gaps were identified in the current BD RAs that are pertinent to their effectiveness in industrial applications. Specifically, limitations were observed in architectures such as S1 (Lambda architecture) and S17 (NIST BD interoperability framework), which are critical for the functionality of contemporary data-driven systems. It was noted that S1 lacks a comprehensive approach to data architecture, failing to adequately address essential elements such as data quality, privacy, and metadata management. Similarly, S17 is characterised by its broad conceptual model for BD but exhibits deficiencies in addressing metadata management and the detailed requirements for data quality and ownership. These findings, detailed in Section 4.6.7, highlight the need for the development of BD RAs that more effectively address these identified gaps, thus enhancing their applicability and relevance to the complexities of current and future BD systems.

Subsequently, the analysis conducted in this SLR not only illuminated the limitations within current BD RAs but also led to the identification of significant gaps in the domain, particularly concerning the architectures' ability to address specific failure modes in BD projects. Such observations have precipitated the formulation of the research question: *How can a Reference Architecture be designed to mitigate or address these failure*

modes in Big Data projects? (see RQ2).

In summation, RAs present a viable starting point for the design and evolution of BD systems. These artefacts foster communication, assimilate requirements from diverse stakeholders, and identify design challenges at an early, cost-effective stage. This underpins the argument for amplifying focus on this domain and its foundational methodological requisites.

4.8 Threats to Validity

Aligned with the protocols of the PRISMA and the research methods detailed in Section 4.5, an evaluation of validity threats is important. This evaluation aims to transparently address and articulate any potential biases or limitations encountered in the execution of this study. The threats to the study's validity, as identified, are presented in the ensuing discussion

- **Construct Validity:** The selection of sources, search terms, and criteria for inclusion and exclusion were designed with the intent to align closely with the SLR's objectives. Nonetheless, threats to construct validity may still be present, emanating from possible subjective interpretations of study eligibility, the risk of omitting pertinent studies due to the specificity of search terms, or the limitations inherent in the databases searched. Efforts were made to meticulously craft the search strategy to encompass the domain's state of the art, yet the complete exclusion of the possibility of overlooking relevant studies or misclassifying the selected studies' relevance cannot be guaranteed.

- **Internal Validity:** Consistency in the application of inclusion and exclusion criteria was maintained through a systematic approach, and measures such as snowballing and the inclusion of grey literature were employed to reduce biases.

However, threats to internal validity might emerge from the variability in the methodologies and quality of the included studies. The structured approach to incorporating grey literature aimed to broaden the review's scope, introducing challenges in evaluating the robustness and reliability of these sources in comparison to peer-reviewed academic literature.

- **Conclusion Validity:** The formulation of a quality assessment framework aided in the evaluation of the included literature, ensuring contributions from studies of high quality. Nevertheless, threats to conclusion validity could arise from subjective interpretations of study quality, the potential for confirmation bias in the selection and evaluation of literature, and the variability in evaluators' expertise and perspectives. Though measures to mitigate these threats involved evaluations from multiple individuals, the subjective nature of quality assessment and interpretation of findings necessitates a cautious approach to the generalisation of the SLR's conclusions.

4.9 Conclusion

This chapter sought to find all BD RAs available in practice and academia. The findings revealed an understanding that RAs can be an effective artefact to tackle complex BD system development. RAs, while incorporating established patterns, represent a comprehensive architectural framework that goes beyond just pattern collections. They encompass architectural decisions, design principles, quality attributes, and their complex interactions that together address a class of problems. The emergence of desired behavior and quality attributes requires careful consideration of how these elements work together, along with contextual factors and implementation details. These artefacts direct attention to architectural requirements and solve many of the

prevalent challenges that an architect might face.

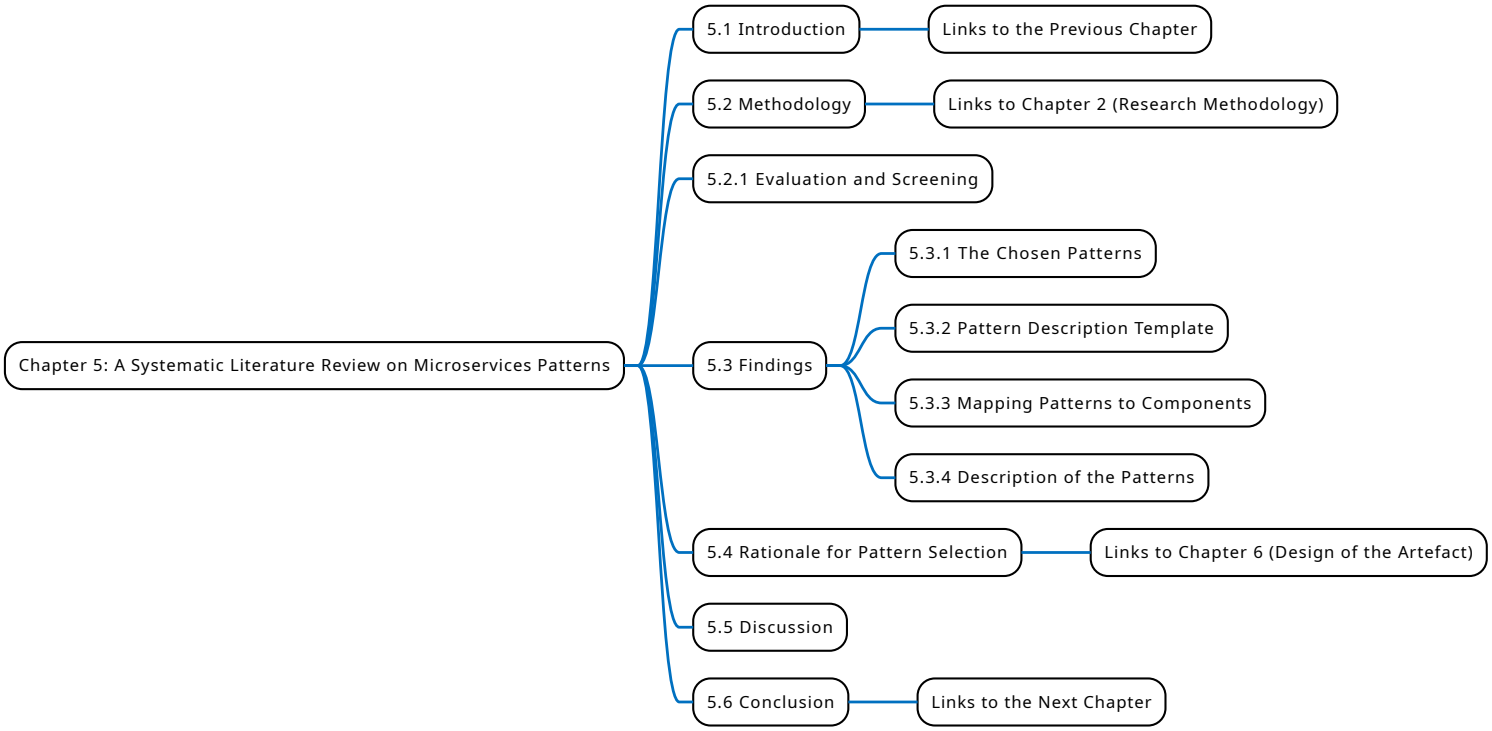
As data proliferates further, there will be more BD systems created. This implies that more technological orchestration is required around data. RAs can be effective in these endeavours. RAs guide the evolution of the system both in terms of functional and non-functional requirements, and pinpoint variability points that can result in more successful BD projects and the avoidance of common pitfalls.

Taking all this into consideration, BD RAs have yet to mature and become ubiquitous in industry, and further research is required in this area. These studies can be done in the areas of micro-services RA for BD systems, event-driven paradigms for BD systems, security and privacy issues in BD systems, and metadata management.

Having synthesised the current state of knowledge in the preceding chapter, the subsequent chapter presents the third SLR of this thesis, diving deeper into the nuances of microservices patterns, addressing the gaps identified in prior studies, and shedding light on overlooked patterns that could be pivotal in the realm of BD architectures.

Chapter 5

A Systematic Literature Review on Microservices Patterns



5.1 Introduction

Following the SLR on BD RAs in the previous chapter, this chapter transitions to focus on microservices patterns. This dedicated SLR aims to address identified gaps and discrepancies in the current academic understanding of microservices patterns.

Microservices patterns have gained significant traction in software engineering due to their ability to tackle issues related to scalability, maintainability, and fault tolerance in distributed systems (Ataei & Staegemann, 2023; Taibi, Lenarduzzi & Pahl, 2018). These patterns provide proven solutions to common architectural problems, enabling the design of loosely coupled, independently deployable, and highly resilient services.

Scalability is addressed by decomposing monolithic applications into smaller, independently scalable services. Maintainability is improved through modular and decoupled design, allowing for independent development, testing, and deployment of services. Fault tolerance is achieved by isolating failures, preventing cascading effects, and enabling graceful degradation.

Given Metamycelium's distributed nature and its emphasis on domain-driven design, leveraging microservices patterns can significantly enhance its architectural robustness and adaptability. Adopting microservices patterns empowers Metamycelium to address the challenges of scalability, maintainability, and fault tolerance inherent in distributed BD systems. The alignment with domain-driven design principles further strengthens the case for leveraging these patterns, enabling Metamycelium to deliver a modular, flexible, and resilient BD architecture that can evolve and adapt to changing business requirements.

While previous works have contributed to cataloguing microservices patterns, a comprehensive and widely accepted collection remains elusive. One notable work in this domain is "Microservices Patterns: With examples in Java" by Richardson (2018). However, Richardson's book, although comprehensive, does not provide an

exhaustive collection of all microservices patterns. Specifically, patterns related to data management, such as the Database per Service pattern, and patterns for ensuring high availability and fault tolerance, like the Circuit Breaker pattern, are not covered in depth.

Additionally, the Azure Architecture Center (*Azure Architecture Center*, 2024) is considered, but its documentation on microservices patterns is limited and primarily tailored towards Microsoft's Azure cloud platform, lacking a broader, platform-agnostic perspective. A critical evaluation of these resources and other relevant literature is necessary to assess the suitability and completeness of the microservices patterns identified, particularly in the context of this study. The research questions guiding this SLR are:

RQ1 What microservices patterns are recognised and described in academic literature?

RQ2 Which of these patterns can be applied or used for a domain-driven distributed BD RA?

These research questions align with the overarching objective of this thesis: to explore the architectural complexities and design considerations in building robust, scalable, and efficient BD systems. By critically analysing microservices patterns and their relevance to BD RAs, this chapter aims to contribute insights into how these patterns can be leveraged to address the challenges of implementing scalable and maintainable BD systems.

The exploration of microservices patterns is considered necessary because the artefact of this study is domain-driven and distributed, and as a result, it can benefit from advancements in the microservices domain. This aligns with the artefact development methodology described in Section 2.9.4.

The methodology for conducting this SLR follows a systematic approach, similar to the previous SLR in Chapter 4. The process includes clearly defined inclusion and exclusion criteria, a comprehensive literature search strategy, and a structured evaluation

and screening process. The findings from this SLR will be analysed and synthesised to identify potential microservices pattern for the development of the artefact.

This chapter is structured as follows: Section 5.2 outlines the SLR Methodology, Section 5.2.1 describes the Evaluation and Screening process, and Section 5.3 presents the Findings and Analysis derived from the SLR. From there on, Section 5.4 provides a rationale for the selection of patterns for the creation of the artefact. Section 5.5 provides a critical discussion on the current state of microservices patterns and how they relate to this study. Lastly, the study concludes by summarising the findings in Section 5.6.

5.2 Methodology

The methodology used in this SLR closely aligns with the research methodology employed in the BD RAs SLR discussed in Chapter 4. However, there are a few minor differences between the two approaches. This methodology follows 14 steps: 1) select data sources, 2) develop a search strategy, 3) develop inclusion and exclusion criteria, 4) develop the quality framework, 5) pool literature based on the search strategy, 6) remove duplicates, 7) scan studies' titles based on inclusion and exclusion criteria, 8) remove studies based on publication types, 9) scan studies abstracts and titles based on inclusion and exclusion criteria, 10) assess studies based on the quality framework (includes three phases), 11) extract data from the remaining papers, 12) code the extracted data, 13) create themes out of codes, 14) present the results.

The first difference between this SLR and the SLR discussed in Chapter 4 is the set of keywords used to search the literature. These keywords used for this SLR are portrayed in Table 5.1. The keywords used are selected to comprehensively identify relevant literature on microservices patterns, architectures, and design strategies. The key concepts targeted include:

1. "microservice*" in the title, abstract, or keywords - to focus the search on microservices-related publications.

2. "pattern*", "architecture*", "design*", "building block*", "best practice*" - to capture a wide range of microservices-specific concepts, techniques, and approaches.

By combining these core microservices terms with the various patterns, architectures, and design elements, the search strategy is aimed at retrieving a diverse set of scholarly works that investigate, describe, or evaluate microservices patterns and related architectural constructs. The rationale behind these search terms is to comprehensively cover the existing research on microservices patterns and design practices, which are the primary focus of this SLR.

Table 5.1: Keywords Used to Search Literature for Microservices Patterns

Database/Register	**Search Term**	**Records**
ACM Digital Library	1) [Title: microservice*] AND [Title: pattern* OR architecture* OR design* OR building block* OR best practice*]	91
	2) [Title: microservice*] AND [Abstract: pattern* OR architecture* OR design* OR building block* OR best practice*]	194
	3) [Title: microservice*] AND [Keywords: pattern* OR architecture* OR design* OR building block* OR best practice*]	65
AISeL	**Title**: microservice OR microservices	10

Table 5.1 continued

Database/Register	**Search Term**	**Records**
IEEE Xplore	"Document Title": microservice* AND ("All Metadata": pattern* OR architecture* OR design* OR building block* OR best practice*)	759
JSTOR	**Title**: microservice OR microservices	0
ScienceDirect	1) **Title, abstract, keywords**: pattern OR architecture OR design OR (building block) OR (best practice) AND **Title**: microservice OR microservices	79
	2) **Title, abstract, keywords**: patterns OR architectures OR designs OR (building blocks) OR (best practices) AND **Title**: microservice OR microservices	76
Scopus	TITLE-ABS-KEY (pattern* OR architecture* OR design* OR "building block" OR "building blocks" OR "best practice" OR "best practices") AND TITLE (microservice*)	1534
SpringerLink	**Title**: microservice* AND **With at least one of the words**: pattern* architecture* design* "building block" "building blocks" "best practice" "best practices"	433
Wiley	**Title**: microservice*	38

5.2.1 Evaluation and Screening

In the initial phase, 1,196 papers are removed due to duplication and publication type. The remaining 1868 papers are filtered by title to evaluate their relevance to the concepts of microservices patterns or architectural constructs related to microservices. From the result of this process, 1699 of the 1868 papers are excluded, leaving 169 items for the next round.

In the second phase, the same approach is followed for abstracts. As a result, 138 papers are excluded. Subsequently, papers not written in English (despite having an English abstract), published before 2012, or with fewer than six pages are removed. 23 papers are then selected for quality assessment against the quality framework. In the next phase, a deeper probing is initiated by evaluating the remaining studies against the quality framework. As a result of this process, 10 studies are yielded.

To further increase the comprehensiveness of the review process, following the recommendation of Webster and Watson (2002), the initial keyword search is amended with a forward and backward search. Here, for the identified ten papers are examined by which papers they cite and which papers cite them.

While the backward search can simply be based on the reference lists given in the papers, the forward search is less unequivocal, because there are several sources with slightly varying information. To account for this, two different ones, namely Google Scholar and ResearchGate are used. However, both searches yield no new results that suffice the criteria applied in the initial search.

Instead, the 538 papers (combined for all papers and both sources, not accounting for duplicates) found in the forward search comprise, inter alia, thesis works, preprints, studies that are not directly related to microservices, papers that are too short and papers that do not meet the quality criteria.

Regarding the backward search, most of the utilised conference papers and journal

articles with a focus on microservices are already captured by the initial search, further highlighting its comprehensiveness. In total for the ten papers, and not accounting for duplicates, there are 16 new entries mentioning microservices in the title that are, however, ultimately not relevant for the focus of this work. Therefore, the final set still consists of the ten contributions shown in Table 5.2.

Table 5.2: Comprehensive final set of literature related to microservices

ID	**Title**	**Year**	**Quality Score**	**Type**	**Source**	**Found in**	**Ref**
S1	Architectural patterns for microservices: A systematic mapping study	2018	11/14	C	CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science	Scopus	(Taibi et al., 2018)

Table 5.2: Comprehensive final set of literature related to microservices

S2	Actual Use of Architectural Patterns in Microservices-Based Open Source Projects	2018	12/14	C	Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC)	Scopus, IEEE Xplore	(Marquez & Astudillo, 2018)
S3	Supporting architectural decision making on data management in microservice architectures	2019	12/14	C	Lecture Notes in Computer Science 11681 Proceedings of the Software Architecture: 13th European Conference, ECSA 2019	Scopus, Springer Link	(Ntentos et al., 2019)

Table 5.2: Comprehensive final set of literature related to microservices

S4	Using architectural modifiability tactics to examine evolution qualities of Service- and Microservice-Based Systems: An approach based on principles and patterns	2019	13/14	J	Software-Intensive Cyber-Physical Systems	Scopus	(Bogner, Wagner & Zimmermann, 2019)
S5	Patterns Related to Microservice Architecture: a Multivocal Literature Review	2020	14/14	J	Programming and Computer Software	Scopus, SpringerLink	(Valdivia, Lora-González, Limón, Cortes-Verdin & Ocharán-Hernández, 2020)
S6	Data management in microservices: State of the practice, challenges, and research directions	2021	12/14	J	Proceedings of the VLDB Endowment 2021	Scopus	(Laigner, Zhou, Salles, Liu & Kalinowski, 2021a)

Table 5.2: Comprehensive final set of literature related to microservices

S7	Deployment and communication patterns in microservice architectures: A systematic literature review	2021	14/14	J	Journal of Systems and Software	Scopus, ScienceDirect	(Aksakalli, Çelik, Can & Tekinerdogan, 2021)
S8	Decision Models for Selecting Patterns and Strategies in Microservices Systems and their Evaluation by Practitioners	2022	13/14	C	Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)	IEEE Xplore	(Waseem et al., 2022)

Table 5.2: Comprehensive final set of literature related to microservices

S9	Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs	2022	13/14	C	IEEE 19th International Conference on Software Architecture (ICSA)	IEEE Xplore	(Vale et al., 2022)
S10	Taxonomical Classification and Systematic Review on Microservices	2022	13/14	J	International Journal of Engineering Trends and Technology	Scopus	(Weerasinghe & Perera, 2022)

Legend for Type column:

C: Conference paper

J: Journal article

5.3 Findings

As a result of this SLR and to answer RQ1, the data synthesis presents 28 microservices patterns. These patterns are categorised based on their primary function and the specific architectural challenges they aim to solve. This categorisation scheme is inspired by the work of Richardson (2022), who proposed a comprehensive pattern language for

microservices. Richardson's classification provides a structured approach to understanding and applying microservices patterns, making it a suitable foundation for this study. The adapted categorisation, presented in Table 5.3, organises the patterns into five main categories: Data Management, Platform and Infrastructure, Communicational, Transactional, Logical, Fault Tolerance, and Observability. This categorisation helps in understanding the role of each pattern within the overall microservices architecture and facilitates the selection and application of appropriate patterns based on the specific requirements and challenges of a given system.

Table 5.3: Microservices Patterns Categories

Category	**Pattern**	**Studies Derived From**
Data Management	1. Database per Service	S3, S6
	2. Shared Database	S3, S6
	3. Event Sourcing	S3, S6
	4. Command and Query Responsibility Segregation	S3, S5, S6, S7
Platform and Infrastructure	5. Multiple Service Instances per Host	S2, S5
	6. External Configuration Store	S5, S7
	7. Sidecar	S2, S5
	8. Static Content Hosting	S2, S5
	9. Computer Resource Consolidation	S2, S5
Communicational, Transactional, Logical	10. API Gateway	S1, S2, S5, S7, S9
	11. Anti-corruption Layer	S5, S7
	12. Self Registration	S2, S5

Continued on next page

Table 5.3 – Microservices Patterns Categories

Category	**Pattern**	**Studies Derived From**
	13. Service Discovery	S2, S5
	14. Competing Consumers	S7
	15. Pipes and Filters	S7
	16. Priority Queue	S2, S5
	17. Ambassador	S2, S5
	18. Gateway Aggregate	S2, S5
	19. Gateway Offloading	S7
	20. Aggregator	S2, S5
	21. Backend for Frontend	S5, S7, S9
	22. API Composition	S2, S5
	23. Saga Transaction Management	S2, S5
	24. Gateway Routing	S2, S5
	25. Leader Election	S2, S5
Fault Tolerance	26. Circuit Breaker	S5, S7, S9
	27. Bulkhead Pattern	S2, S5
Observability	28. Log Aggregation Pattern	S5, S7

5.3.1 The Chosen Patterns

This SLR's objective is not to describe each microservices pattern. Most of these patterns are already defined in the works of Richardson (2018) and on Azure Architecture Center official website (*Azure Architecture Center*, 2024). Therefore, only the patterns used directly or indirectly in the creation of either design theories discussed in Section 6.4 or the architectural constructs in Section 6.5 are explained. As a result of this, 10 patterns are identified as following:

1. API Gateway
2. Gateway Offloading
3. External Configuration Store
4. Competing Consumers
5. Circuit Breaker
6. Log Aggregation
7. Command and Query Responsibility Segregation (CQRS)
8. Anti-Corruption Layer
9. Backend for Frontend (BFF)
10. Pipes and Filters

The selection of these 10 microservices patterns is based on their relevance and potential applicability to the design of Metamycelium. These patterns are identified as particularly useful in addressing common challenges and concerns in BD engineering, such as scalability, resiliency, data management, and communication between distributed components. For example, the *API Gateway* and *Gateway Offloading* patterns can facilitate efficient and secure communication between various components of a BD system, while patterns like *Competing Consumers* and *Circuit Breaker* can enhance fault tolerance and reliability.

Additionally, patterns like *CQRS* and *Anti-Corruption Layer* can aid in effective data management and integration within a distributed architecture. The identification and mapping of these microservices patterns to the context of BD systems was informed by the research published in the paper “Application of microservices patterns to BD

systems" (Ataei & Staegemann, 2023), which provided valuable insights into the relevance and applicability of microservices patterns specifically for BD engineering.These patterns are identified through various methods employed in the respective studies, such as systematic literature reviews (S1, S5, S7, S10), case studies (S2, S9), and empirical analyses (S3, S4, S6, S8).

5.3.2 Pattern Description Template

The microservices patterns are described using a specific format and template proposed by Buschmann, Meunier, Rohnert, Sommerlad and Stal (2008) in their book "Pattern-Oriented Software Architecture: A System of Patterns." This template provides a structured way to document and communicate software patterns effectively.

The pattern template typically includes several elements such as the context in which the pattern is applicable, the problem or challenge the pattern aims to address (often framed as a set of questions), the forces or constraints that influence the solution, variations or alternative implementations of the pattern, examples or known uses of the pattern, the resulting context after applying the pattern, and related patterns that may be relevant.

However, for the purposes of this study, the focus is on presenting the core essence of each pattern in a concise and accessible manner, rather than providing an exhaustive documentation of all pattern elements. Therefore, the decision was made to omit certain elements from the pattern template, such as 'forces', 'variation', 'examples', 'resulting context', 'related patterns', 'known uses', and 'example application'.

Instead, the patterns are presented in a more streamlined format, starting with a description of the context in which the pattern is applicable. This context sets the stage for understanding the problem or challenge that the pattern addresses. The challenges are then framed as a series of questions, highlighting the key concerns and considerations

that the pattern aims to resolve. Finally, the proposed solution is presented in the form of the corresponding pattern, providing a high-level overview of how the pattern addresses the challenges outlined in the context and questions.

5.3.3 Mapping Patterns to Components

The pattern API Gateway is used to inspire *Ingress Gateway*, Gateway Offloading has also affected the design of both *Ingress Gateway* and the *Egress Gateway*, External Configuration Store is implemented as *Data Lichen*, Competing Consumers and Circuit Breaker have indirectly affected the design of *Event Processing Interface*, Log Aggregation is implemented as the *Telemetry Processor*, CQRS has indirectly affected the design of the *Event Backbone* and the *Service Mesh*, Anti-Corruption layer has indirectly affected the design of the *Egress Gateway* and the *Service Mesh*, BFF has affected the design of the *Batch Processing and Stream Processing Controller*, and lastly Pipes and Filters affected the design of the communication in the *Service Mesh*. The mapping of the patterns to architectural components of Metamycelium are displayed in Table 5.4. Additionally, the architectural components mapped to the patterns discussed here are explain in detail in Section 6.5.

5.3.4 Description of the Patterns

In this section, the 10 microservices patterns that are identified as particularly relevant and applicable to the design of the Metamycelium are described in detail. These patterns are selected based on their potential to address common challenges and concerns in BD engineering, such as scalability, resiliency, data management, and communication between distributed components. As discussed hereinabove, these patterns are mapped against the Metamycelium components, and their usage is further elaborated in the explanation of the respective components in Section 6.5.

Table 5.4: Microservices Patterns to Architectural Components Mapping

Pattern	**Architectural Components**
API Gateway	Ingress Gateway
Gateway Offloading	Ingress Gateway, Egress Gateway
External Configuration Store	Data Lichen
Competing Consumers	Event Processing Interface
Pipes and Filters	Service Mesh
Backend for Frontend	Batch Processing, Stream Processing Controller
Anti-Corruption Layer	Egress Gateway, Service Mesh
Command and Query Responsibility Segregation (CQRS)	Event Backbone, Service Mesh
Circuit Breaker	Event Processing Interface
Log Aggregation	Telemetry Processor

The descriptions aim to provide a clear understanding of each pattern's context, the challenges it addresses, and the proposed solution it offers. By presenting these patterns and their relevance to BD systems, this section lays the foundation for understanding how they influenced and shaped the design decisions behind various components of the Metamycelium architecture.

It's important to note that while these patterns are well-established in the microservices domain, their application and adaptation to the specific context of BD systems may require unique considerations and tailoring. Therefore, the descriptions will also highlight any specific nuances or interpretations relevant to the BD engineering domain.

Through these detailed pattern descriptions, readers will gain insights into some of the rationale behind the architectural choices made in the Metamycelium RA and how microservices patterns can be effectively leveraged to address the complexities and challenges inherent in designing scalable, resilient, and efficient BD systems. This

section is necessary to familiarise with the patterns that have directly and indirectly affected the design of the Metamycelium artefact, providing the necessary background and context for understanding the architectural decisions made.

API Gateway

Context: In a Software as a Service (SaaS) application, various microservices are responsible for distinct functionalities such as financial management, medical records, client information, and data related to different domains like agriculture or healthcare. The system consists of multiple frontend applications, each requiring specific data from the backend microservices to fulfill its purpose.

For example, an agricultural frontend application may require access to data from microservices dealing with crop management, livestock records, weather data, soil analysis, and financial information. Similarly, a healthcare frontend may need to aggregate data from patient records, laboratory tests, insurance claims, and billing microservices to provide a comprehensive view.

Problem: How does the financial micro-frontend retrieve the data it needs from various backends? Should it make separate Representational State Transfer (REST) requests to different APIs and then combine the data representation required? REST is an architectural style for distributed hypermedia systems, commonly used in web services for its simplicity and statelessness. As microservices evolve over time, how does this point-to-point approach adapt to changes? If the financial microservice changes its API, how does it impact the data composition logic in the frontend? How does the frontend get notified of new endpoints? How does it handle authentication with each individual microservice?

Should these configurations be hard-coded and frequently updated? If a microservice changes the implementation of certain functionality, would it break

the frontend in production? Addressing these questions reveals a central issue: The uncertainty and complexity surrounding data retrieval and integration for the financial micro-frontend from diverse backend sources. Handling changes to microservice APIs, discovering new endpoints, managing authentication across services, configuring services, and ensuring stable operations are all intertwined challenges that need robust solutions.

Solution: The solution is to have one gateway that resolves different data necessary for various micro-frontends (see Figure 5.1). The API gateway can act as a single entry for all clients, handling version changes, reducing the network requests, and addressing cross-cutting concerns. In addition, the API gateway can help with load balancing. The gateway can either proxy/route requests to appropriate services or it can fan out a request to multiple services. Underlying this approach, the communication pattern is streamlined and micro-frontends are only required to know about the gateway.

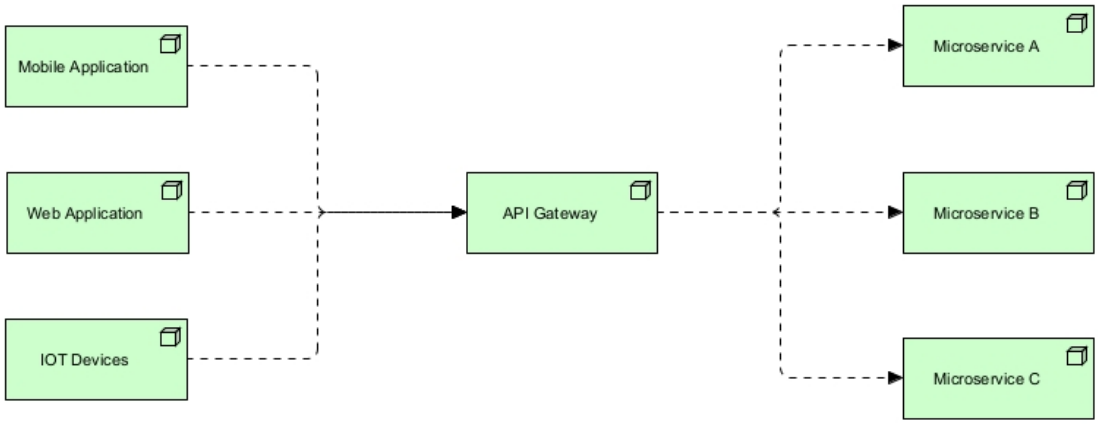


Figure 5.1: API Gateway Pattern

Gateway Offloading

Context: In the context of the SaaS practice management system, various microservices possess shared features. These features necessitate maintenance, configuration, and management. Such features include token validation, feature flag management, SSL certificate management, encryption, and environment variable management.

Problem: How does one go about handling these shared features? Should each team write their own feature for their own services? If a feature is updated, should each team then update their own implementation? How do we ensure that these features conform to the same interface and standards? If a new feature is added, should we communicate with three different teams to update their implementation? What happens if an implementation of one team does not respect the specification?

Handling shared features across microservices presents challenges. Determining whether each team should independently develop and maintain their own feature poses difficulties. Ensuring consistent interfaces and standards across features is paramount, especially when updates or new features are introduced. Communication becomes complex when multiple teams are involved, and there is a risk of discrepancies if a team's implementation diverges from the specified standards.

Solution: Common features and cross-cutting concerns can be offloaded into a gateway (see Figure 5.2). This includes but is not limited to: SSL termination, certificate management, feature flag management, environment variables management, secret management, monitoring, logging configurations, throttling, and protocol translation. This approach simplifies the development of services, and improves the maintainability of the system. In addition, features that require special skills (privacy and security) can be developed by experts and propagated

to teams, eliminating the risk that non-expert developers may introduce. This pattern also introduces more consistency, and standardised interfaces, which helps with communication, agility and productivity of development teams.

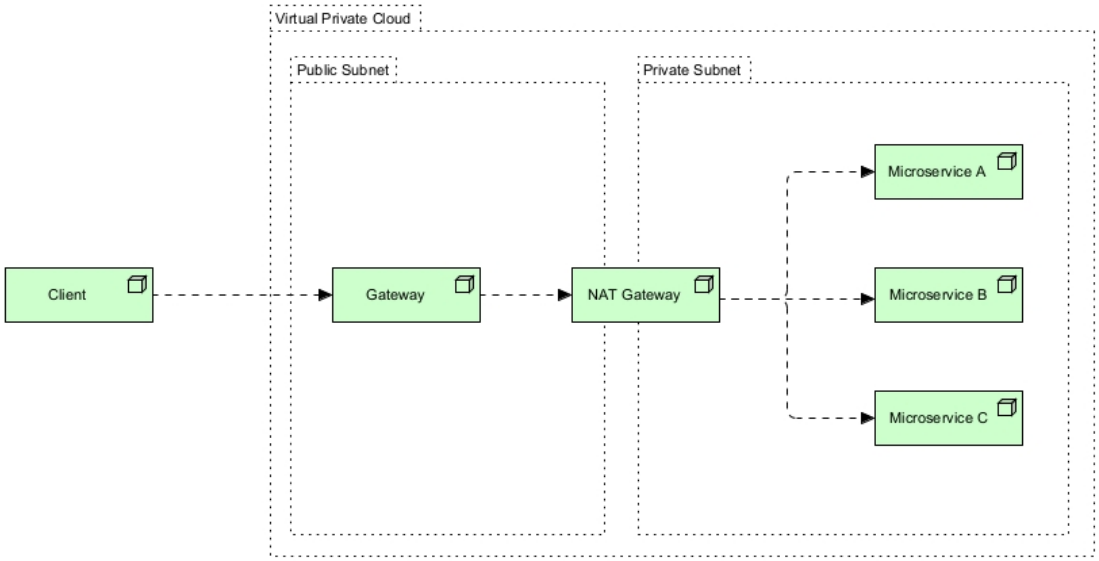


Figure 5.2: Gateway Offloading Pattern

External Configuration Store

Context: Software applications are usually deployed to various environments for different purposes. This is part of the continuous integration, continous delivery (CI/CD) approach that creates pipelines to capture bugs and issues. For instance, there are testing, integration, pre-production, and production environments. Each environment is tailored for a different purpose. In a development environment, several feature flags may be deactivated, some infrastructure may be configured to reduce costs, and end to end tests may run. Therefore, an application needs to keep a list of configuration for internal and third-party infrastructure it needs. In addition, various classes of services require radically different configurations to meet their ends. These configurations could be a buffer size setup on stream

processing services or it could be the timeout time set on batch processing services.

Problem: Should each application have its configuration written separately? As the number of applications grows, how does one scale and maintain this? If a configuration should be uploaded for a class of similar services, should each service update its configuration separately? How can configurations be shared across several applications?

Managing configurations for multiple applications poses challenges. As the number of applications increases, scalability and maintenance of configurations become problematic. Determining whether to centralise configurations for a class of similar services or have each service update its own configuration independently is complex. Furthermore, establishing a method to share configurations across several applications is essential.

Solution: Store all application configurations in an external store (see Figure 5.3). This can include package versions, database credentials, network locations and APIs. On startup, an application can request for the corresponding configuration from the external configuration store.

Competing Consumers

Context: An enterprise application, specially a data-intensive one is expected to handle a large number of requests. Handling these requests synchronously would be challenging. A common approach is for applications to send these requests through a messaging system to another application that handles them asynchronously. This ensures that one blocking service is not going to have a ripple effect on the system. Requests loads vary at different times. During peak hours there might be

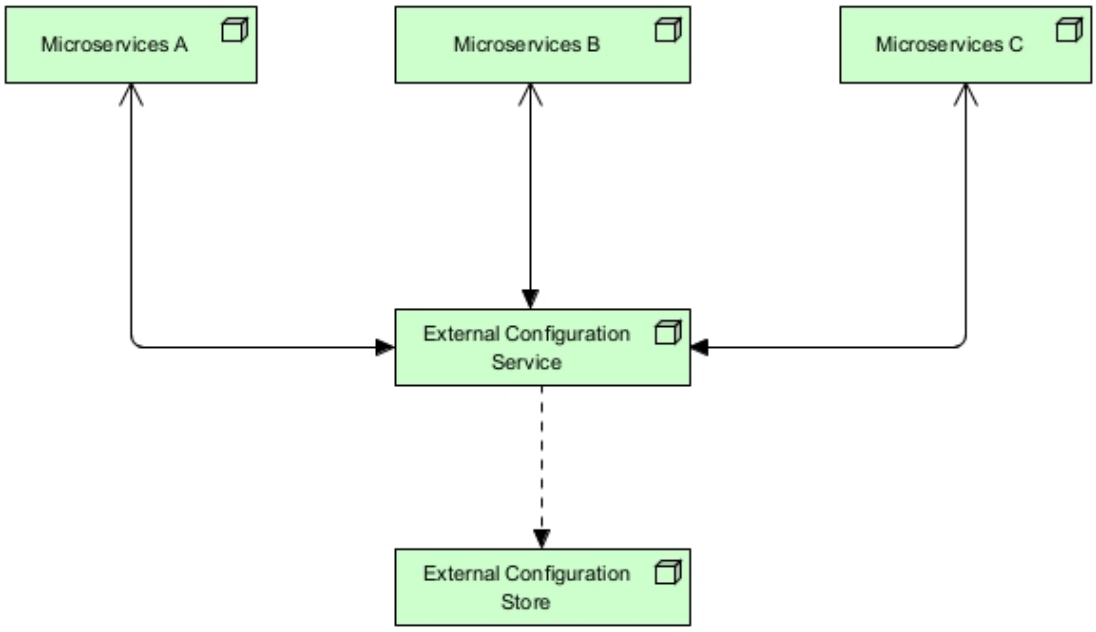


Figure 5.3: External Configuration Store Pattern

many requests coming from various sources. In addition, the processing required for different requests varies, and while some may be quite cheap, others might be compute intensive.

Problem: Should only one consumer instance be responsible for incoming requests? What happens if that consumer instance does not have the computing resources available? What happens if that consumer instance fails? Relying on a single consumer instance for handling incoming requests introduces vulnerabilities. There is uncertainty about whether the single consumer has adequate computing resources. Additionally, the potential failure of this lone consumer instance could lead to a complete halt in processing requests.

Solution: A message queue system can be used to load balance requests to different consuming services based on their availability (see Figure 5.4). In this case, a group of consumer applications is created, which allows for timely processing of incoming requests during peak time. This can be achieved either by a push model

(message queue pushing to available consumer nodes), or a pull model (consumer nodes pull requests based on their state and process it).

This increases the elasticity, availability and reliability of the system. The queue can act as a buffer between the producer and consumer instance, and help with minimising the impact of consumer service's unavailability. The message can also be enhanced with fault tolerant mechanisms in case of node failures. Furthermore, scalability is improved as new data consumers can be dynamically added. For instance, in Amazon Web Services (AWS), auto scaling groups can be set for Elastic Compute Cloud (EC2) instances, which are virtual servers in the cloud.

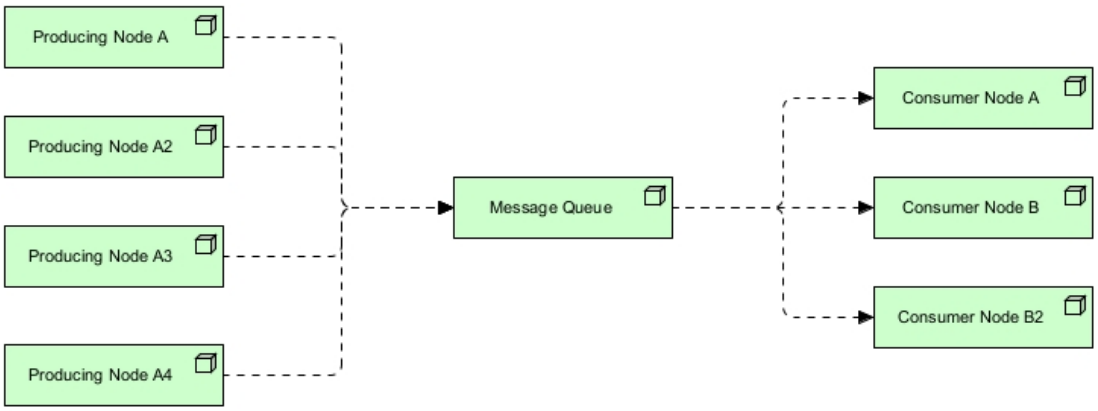


Figure 5.4: Competing Consumers Pattern

Circuit Breaker

Context: Suppose a company is using a microservices architecture. There are various services that communicate with each other to process requests. If one service synchronously calls another service through a REST API, there is a chance that the other service may not be available or is exhibiting a high latency. As the number of services grow, there will be an increased requirement for communication between services. Therefore, the failure of one service can introduce a bottleneck to the whole system.

Problem: How does one handle the failing service? How should the failed service be handled to avoid a ripple effect? Addressing service failures is crucial to prevent a cascading impact on interconnected services. The challenge is to implement a mechanism that effectively manages a failing service without causing disruptions to the larger system.

Solution: An architect can employ the circuit breaker pattern. The circuit breaker pattern prevents services from repeatedly calling the failing service. This allows for the system to operate in spite of a failing node, which helps with saving CPU cycles, improving availability, improving reliability and decreasing the chance of faulty data. In addition, circuit breaker signals the fault resolution, which allows system to get back to its default state.

In a common scenario, circuit breaker acts as a proxy between the source and destination services, and monitors the destination service. If the number of failing requests reaches a certain threshold, the circuit breaker trips, blocking subsequent requests to the destination. The circuit breaker then probes the failing service to identify its health. Once the service becomes healthy again, the circuit breaker allows requests to be passed to the destination.

Circuit breaker can be implemented on frontend, backend, or as a standalone service. This pattern is usually implemented as a state machine that mimics the functionality of an electrical circuit breaker. This is often designed in three states:

1. Closed: the default state, where the circuit breaker listens on the number of incoming requests
2. Open: if the number of failing requests reaches a certain threshold, the circuit breaker trips, immediately returning an exception
3. Half-open: a limited number of requests are passed, if these requests are

passed, it is assumed that the service is healthy, and the circuit breaker switches to closed state. If any requests fail, the circuit breaker assumes the fault is still present, so it reverts back to open state

This pattern is displayed in Figure 5.5.

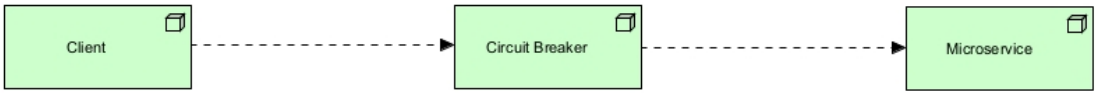


Figure 5.5: Circuit Breaker Pattern

Log Aggregation

Context: Microservices architectures often comprise a large set of services, each having its own domain and responsibility. A request usually spans multiple services and in the process something might go wrong, and bugs may occur. Each system writes logs in a standardised format about errors, warning and access requests.

Problem: How to understand the root cause of an issue if it is spanning across multiple services? Should one read the logs of one service, and then the logs of the other and the next to try to make sense of the problem? Identifying the root cause of issues in a multi-service environment presents a challenge. Without a unified system for log analysis, tracing issues across interconnected services becomes complex and time-consuming.

Solution: A centralised logging service can be implemented that retrieves logs from different services and composes them together (see Figure 5.6.). The developers can then search and analyse these logs to make sense of the root cause. This eliminates the tedious task of going to each service, extracting logs and aggregating them manually.

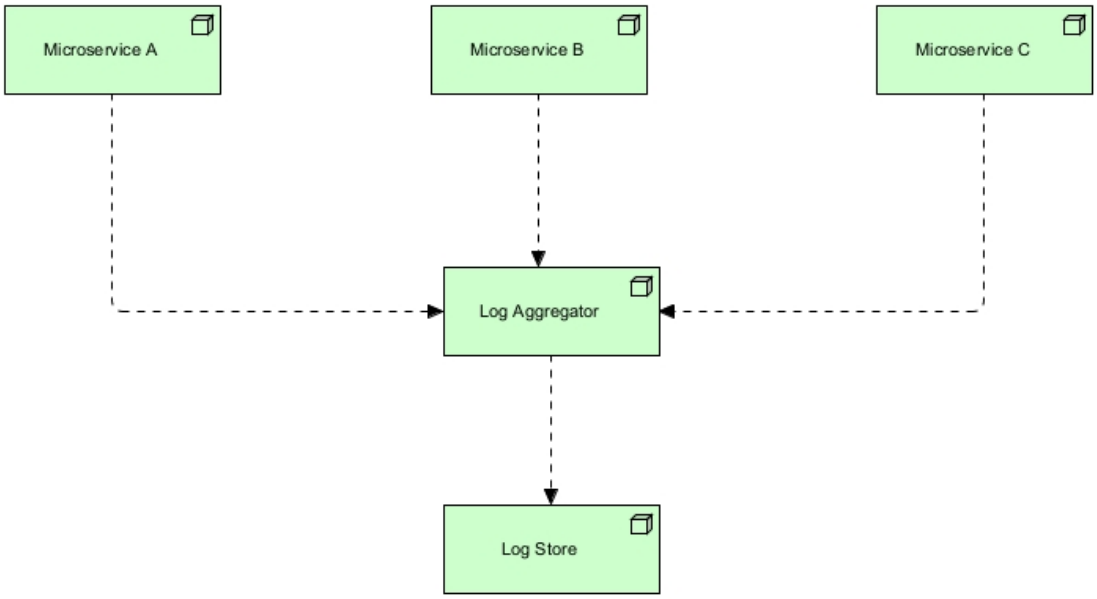


Figure 5.6: Log Aggregation Pattern

Command and Query Responsibility Segregation (CQRS)

Context: Suppose that a team is working on a data heavy service. This service needs to scale and process a lot of data. Following the traditional approach, often the same data model is used to query and update the database. Underlying this approach, the read and write workloads both go to the same datastore.

Problem: How should the team optimise for read workloads? How should the team optimise for the write workloads? Can the team optimise for both read and write workloads? How does the team handle the missmatch between the read and write representations of the data? How does the team ensure a certain performance objective is met on read workloads?

The team faces a challenge in optimising data handling for both read and write workloads. Striking a balance between these two operations can lead to discrepancies in the representations of the data. Additionally, ensuring that specific performance objectives are met for read workloads presents further complexities.

Solution: Implement CQRS pattern to separate read and write workloads, using commands to update the data and queries to read the data (see Figure 5.7). This is usually achieved through a message queue asynchronously. Having the command and query separated simplifies modeling, development, and maintenance of data stores. In addition, the system will be able to support multiple denormalised views that are optimised for a specific workload.

CQRS is commonly implemented in two distinct data stores. This allows for the read database to optimise for read queries. For instance, it can store a materialised view of the data, and avoid expensive joints or complex ORM mappings. The read database can be a different type of data store. One might choose to use a graph database such as Neo4J for relationship heavy datasets, or a NoSQL database such as MongoDB for highly dynamic data. On the other hand, CQRS can potentially increase complexity, introduce code-duplication and increase latency.

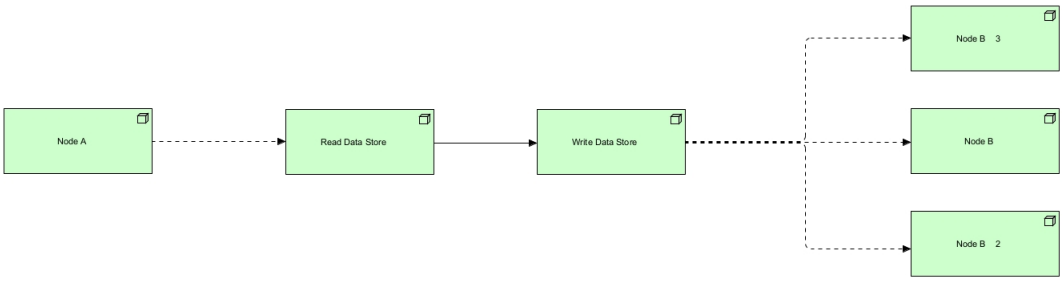


Figure 5.7: Command and Query Responsibility Segregation Pattern

Anti-Corruption Layer

Context: Most services rely on some other services for data or functionality. Each service has its own domain model. Some of these services can be external services, some of these services can be internal legacy services, and some of them can be bleeding edge services. For these services to interoperate, there is a need for a standard interface, protocol, data model or APIs.

Problem: How does one maintain access between legacy internal systems and bleeding edge internal systems? How does one enable interoperability between legacy internal services and external services? Should the bleeding edge service be modified to account for legacy service's interface or API? Should the internal services support the API requirements of external services even if they are sub-optimal? Should the semantics of legacy and external services be imposed to the bleeding edge service? Should services be corrupted by the requirements of other services?

Maintaining access and ensuring interoperability between legacy and cutting-edge internal systems, as well as external services, poses a challenge. Decisions need to be made regarding adapting the cutting-edge service to legacy interfaces or accommodating potentially sub-optimal external API requirements. This leads to the dilemma of whether to compromise the integrity and semantics of newer services based on the demands of older or external services.

Solution: Define an anti-corruption layer that translates semantics between different services' domains. This enables services to be unaffected by external entities, avoiding compromises on interface, design and the technological approach. The anti-corruption layer can be a module, a class inside the application or it can be an independent service. This pattern is displayed in Figure 5.8.

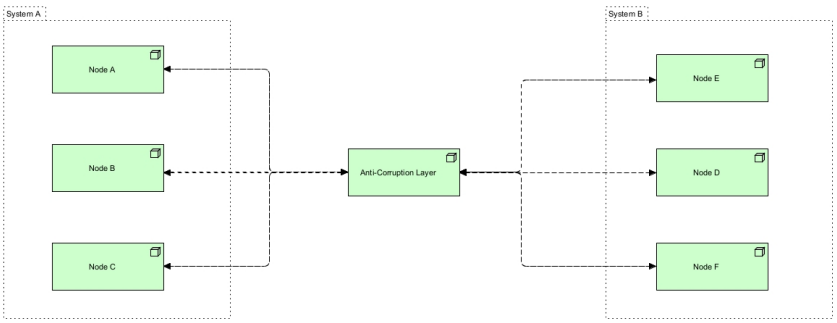


Figure 5.8: Anti-Corruption Layer Pattern

Backend for Frontend

Context: In a large scale system, a backend service needs to provide the necessary APIs for various clients. A client can be the user's browser, a mobile phone, or an IoT device. As the number of clients grows, the traffic grows, and new requirements emerge. As a result, the backend service needs to account for higher level of abstraction to serve the requirements of different clients.

Problem: Should the backend service account for various clients? If the backend service tries to account for all clients, how hard will it be to maintain this service? Can a general-purpose highly abstract backend service be scaled and maintained easily? If the web development team has a conflicting requirement with the mobile development team, how does the backend service account for that? How does the backend service provide optimised data for each client? How can the backend service be optimised for various clients?

The challenge arises when determining whether a backend service should cater to diverse clients. Accommodating every client type might complicate the service's maintainability. Striking a balance between a general-purpose, highly abstract backend and the specific needs of web and mobile clients becomes crucial. Additionally, optimising the service for each client type and managing conflicting requirements between development teams further complicates the situation.

Solution: A dedicated backend that accounts for a specific client (frontend) can be created. This introduces opportunities for optimising performance of each backend to best match the needs of the frontend, without worrying much about introducing side-effects to other frontends. In addition, the backend will be smaller, better abstracted, less complex, and therefore easier to maintain and scale. Furthermore, this enables horizontal teams to work without side-effects and conflicting

requirements. This pattern is depicted in Figure 5.9.

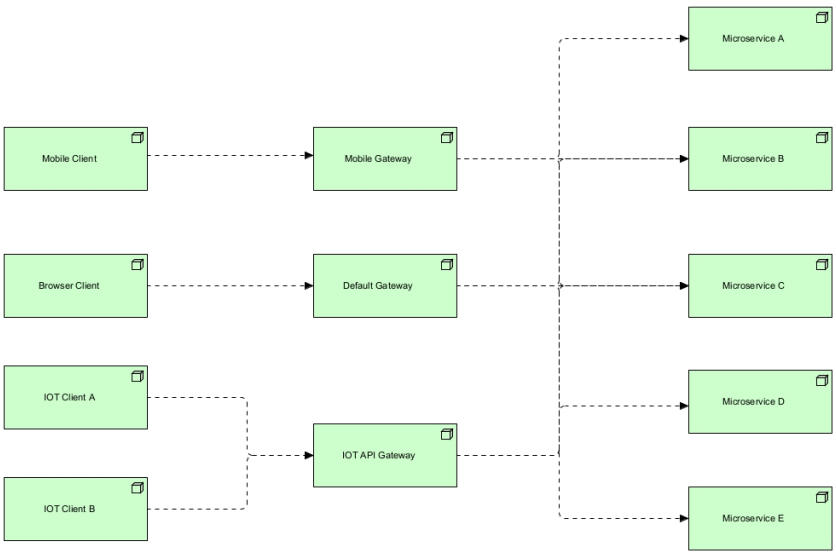


Figure 5.9: Backend for Frontend Pattern

Pipes and Filters

Context: A large scale application is usually required to do numerous processes with varying complexity. For instance, a complex business logic may require several transformations to be done on the data. These transformations can be sophisticated and require many lines of code to be written.

Problem: Should all these processes be performed in one monolithic module? How flexible is that approach? In light of emerging requirements how can one maintain and scale the monolithic module? Is that the right level of abstraction? Does this approach provide with much opportunity to optimise or reuse parts of the module?

The dilemma revolves around the use of a monolithic module for executing various processes. Concerns arise regarding its flexibility, maintainability, scalability, and the level of abstraction it provides. Moreover, there's uncertainty about the

module's potential for optimisation and component reuse in response to evolving requirements.

Solution: Different processes can be broken down into their own components (filters), each taking a single responsibility. This provides clean and modular components that can be extended and modified with ease. This pattern is ubiquitous in Unix like operating system; for example it is common for system engineers to pipe the result of the command 'ls' (list) into the command 'grep' (global search for regular expression) or command 'sed' (stream editor). By standardising the interface for data input and output, these filters can be easily combined to create a more powerful whole. Composition then becomes natural, and the maintainability increases. This pattern is portrayed at Figure 5.10.

Figure 5.10: Pipes and filters Pattern

5.4 Rationale for Pattern Selection

This section presents the rationale for choosing the 10 microservices patterns discussed hereinabove. A clear mapping between the components, requirements, theories and principles of Metamycelium and the chosen microservices patterns are made. While the requirements, artefacts and the principles of the artefact are discussed later in Chapter 6, this rational is deemed necessary as it justifies the need for this SLR. Moreover, Metamycelium's software and system requirements are mentioned in this section with their alphanumeric identifier which are described in Section 6.3.

The selection of the 10 microservices patterns for the artefact was driven by their ability to address key challenges and concerns in BD engineering, while aligning with the architectural principles, styles, and design decisions outlined in Section 6.4.

The API Gateway and Gateway Offloading patterns are chosen to facilitate efficient communication and load balancing between components of the BD system and external clients or data consumers within Metamycelium. These patterns directly influenced the design of the Ingress Gateway and Egress Gateway components, serving as entry and exit points. By consolidating cross-cutting concerns such as authentication, SSL termination, and protocol translation at these gateways, Metamycelium promotes secure and optimised data flow, addressing requirements like Vol-1, Var-1, Var-3, Var-4, Val-1, Val-3, Val-4, SaP-1, and SaP-2.

The Competing Consumers and Circuit Breaker patterns are selected to enhance fault tolerance, reliability, and resilience within Metamycelium's distributed architecture. These patterns indirectly influenced the design of the Event Processing Interface component, which acts as an intermediary for handling events and managing service communication. By enabling load balancing of requests across multiple consumer instances and implementing circuit breaking mechanisms, Metamycelium aims to mitigate the impact of service failures and ensure continued operation, addressing requirements like Val-1 and Ver-1.

The Log Aggregation pattern is adopted to simplify troubleshooting and observability across Metamycelium's components. This pattern directly influenced the design of the Telemetry Processor component, which centralises log collection and analysis from various domains. By providing a unified view of logs and metrics, Metamycelium facilitates root cause analysis and performance monitoring, indirectly addressing requirements such as Vol-1, Vel-1, Val-1, and Ver-1.

The CQRS pattern is chosen to optimise data management and processing within Metamycelium's domains. By separating read and write workloads, this pattern influenced the design of the Event Backbone and the Service Mesh components, enabling domains to optimise their data stores and processing pipelines for specific workloads. This pattern indirectly addresses requirements related to data processing, such as Vel-1,

Vel-2, Vel-3, Vel-4, Vel-5, Val-1, Val-2, Ver-1, Ver-2, and Ver-3.

The Anti-Corruption Layer pattern is selected to facilitate the integration of legacy systems and external services with Metamycelium's components. This pattern indirectly influenced the design of the Egress Gateway and the Service Mesh, enabling domains to communicate with external entities while maintaining their independence and avoiding compromises on their interfaces, design, and technological approaches.

The BFF pattern influenced the design of the Batch Processing and Stream Processing Controller components, dedicated to handling batch and streaming events respectively. By separating concerns and optimising processing pipelines for specific workloads, these components address requirements like Vel-1, Val-1, Val-2, Vel-2, Vel-4, and Vel-5.

Finally, the Pipes and Filters pattern is adopted to facilitate modular and composable data processing pipelines within each domain's Service Mesh. This pattern promotes separation of concerns, code reusability, and maintainability, aligning with Metamycelium's principles of loose coupling and domain-driven decentralisation.

The selection of these 10 patterns is guided by the overarching architectural principles and styles defined for Metamycelium, such as domain-driven decentralisation (AS1, AR1, AR2, AP1), event-driven communication (AP8), federated governance (AP5, AP6, AR6), and the treatment of data as a first-class citizen (AP4). While other patterns may have been considered, the chosen set aimed to strike a balance between addressing key challenges in BD engineering while adhering to Metamycelium's architectural vision, promoting scalability, resilience, and maintainability within a distributed, domain-driven landscape.

5.5 Discussion

This SLR aimed to identify and analyse prevalent microservices patterns described in academic literature and industry sources. Through a comprehensive search and evaluation process, 28 distinct patterns that address various architectural challenges in microservices-based systems are identified. These patterns are categorised into groups based on their primary function and the specific architectural challenges they aim to solve. The categorisation scheme was inspired by the work of Richardson (2018), who proposed a comprehensive pattern language for microservices.

The adapted categorisation organised the patterns into five main categories: Data Management, Platform and Infrastructure, Communicational, Transactional, Logical, Fault Tolerance, and Observability. This categorisation facilitates understanding the role of each pattern within the overall microservices architecture and the selection and application of appropriate patterns based on the specific requirements and challenges of a given system.

The identified patterns serve as a valuable resource for architects and developers working on microservices-based BD systems like Metamycelium. For the Metamycelium RA, which aims to enable the development of robust, scalable, and efficient BD systems, 10 specific patterns are deemed particularly relevant and described in detail. These patterns include API Gateway, Gateway Offloading, External Configuration Store, Competing Consumers, Circuit Breaker, Log Aggregation, CQRS, Anti-Corruption Layer, Backend for Frontend, and Pipes and Filters. These patterns address critical concerns such as efficient communication, resilience, data management, and integration within a distributed BD architecture.

The mapping of these patterns to specific components of the Metamycelium architecture highlighted their potential applicability and relevance in addressing real-world challenges in BD engineering. For instance, the API Gateway and Gateway Offloading

patterns influenced the design of the Ingress Gateway and Egress Gateway components, facilitating efficient and secure communication between microservices and external clients within Metamycelium. The nuances and interpretations relevant to the BD engineering domain are highlighted in the pattern descriptions, providing insights into leveraging these patterns effectively in the context of scalable, distributed and resilient BD architectures like Metamycelium.

This SLR contributes to the growing body of knowledge on microservices patterns and their application in BD systems, specifically informing the design and development of the Metamycelium RA. By consolidating and analysing relevant literature, a comprehensive overview of patterns that can guide architects and developers in designing and implementing scalable, and maintainable microservices-based BD solutions like Metamycelium is provided.

However, it is essential to acknowledge the limitations of this study. While the search strategy aimed to be comprehensive, some relevant literature may have been inadvertently excluded. Additionally, the selection and mapping of patterns to the Metamycelium architecture components involved subjective judgments based on the researchers' expertise and understanding of the domain. The threats to validity discussed in Section 4.8 apply to this SLR as well.

Future research could explore the practical implementation and evaluation of these patterns within the Metamycelium RA and other real-world BD systems, investigating their efficacy, trade-offs, and potential for integration with emerging technologies and approaches in the field of BD engineering. Furthermore, as the microservices and BD landscapes continue to evolve, regular updates and extensions to the identified pattern catalogue may be necessary to ensure its relevance and comprehensiveness for the development of Metamycelium and other BD RAs.

5.6 Conclusion

This SLR comprehensively captured microservices patterns emerging from academic and industry sources. The identified patterns, categorised based on their primary functions and architectural challenges addressed, serve as foundational pillars influencing the design of the Metamycelium RA for BD systems.

From the 28 distinct microservices patterns identified, 10 were deemed particularly relevant and described in detail: API Gateway, Gateway Offloading, External Configuration Store, Competing Consumers, Circuit Breaker, Log Aggregation, CQRS, Anti-Corruption Layer, Backend for Frontend, and Pipes and Filters. These patterns address critical concerns like communication, resilience, data management, and integration within distributed BD architectures.

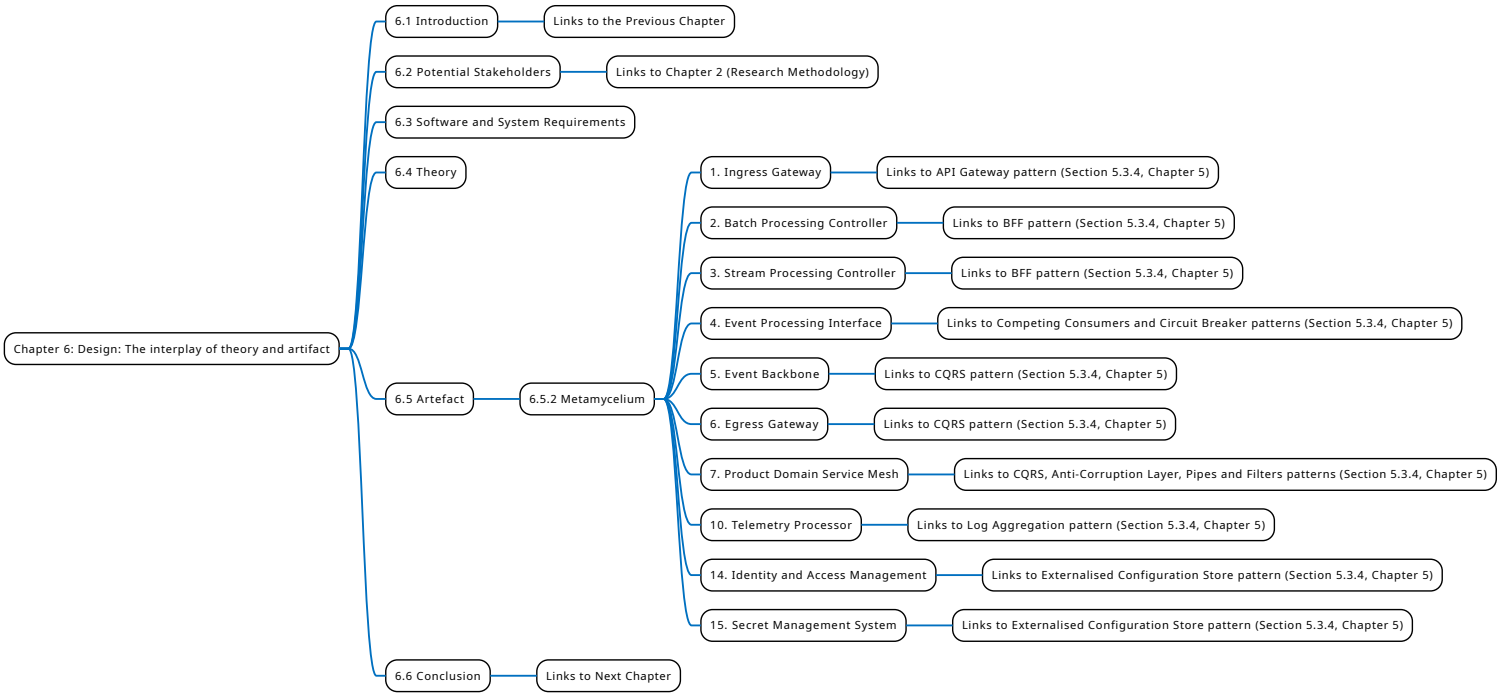
The mapping of these patterns to Metamycelium's components highlighted their applicability in addressing real-world challenges encountered during development. Their nuances and interpretations relevant to BD engineering are highlighted, providing insights for leveraging them effectively in scalable, resilient, and efficient BD architectures.

While comprehensive, the study's limitations include the potential inadvertent exclusion of some literature and the subjective judgments involved in pattern selection and mapping to Metamycelium's components.

This review contributes to the growing knowledge on microservices patterns and their application in BD systems, informing the design and development of Metamycelium. In the next chapter, the artefact of this study, Metamycelium is introduced in detail.

Chapter 6

Design: The Interplay of Theory and Artefact



6.1 Introduction

In the preceding chapter, microservices patterns have been studied and their correlation to the artefact are discussed. This chapter is dedicated to the design and development of a treatment, here referred to as the artefact. Artefact refers to physical or digital objects created during the design process, while theories are abstract concepts that guide the design. The relationship between artefact and theory is iterative, with each informing and shaping the other. DSR can help bridge the gap between theory and practice and lead to innovative solutions.

This chapter is made up of the following integral elements: 1) discussion on potential stakeholders (Section 6.2), 2) the requirements specified for the artefact (Section 6.3), 3) the theories that underpin the design and development of the artefact (Section 6.4), and 4) the artefact itself (Section 6.5).

6.2 Potential Stakeholders

Stakeholders in DSR are individuals or groups who can influence or be affected by the design, development, implementation, or use of the artefact being created (A. R. Hevner et al., 2010). The identification of potential stakeholders for Metamycelium is based on the analysis of industrial reports and surveys discussed in Section 2.9.1 of the research methodology chapter.

The following roles are identified as potential stakeholders for Metamycelium:

1. **Data Engineers**: These professionals are tasked with designing, building, and maintaining the data infrastructure that supports an organisation's BD initiatives. The introduction of a BD RA can influence data engineers in several ways, including the need to familiarise with new tools and technologies and the requirement to modify existing data pipelines to align with the new architecture.

2. **Data Architects**: These individuals are responsible for designing the comprehensive data architecture that meets an organisation's business needs. The introduction of a BD RA can influence data architects, necessitating modifications to their existing data architecture to fit the new reference model or designing new data models to support the new structure.

3. **Data Stewards**: Data stewards have a crucial role in ensuring the accuracy, completeness, and security of an organisation's data. They collaborate with data engineers and data architects to manage data in a way that meets the organisation's data governance policies. The adoption of a BD RA can impact data stewards in several ways, including modifying existing data governance policies to align with the new architecture or working closely with data engineers and architects to ensure consistent data management.

4. **Data Scientists**: Data scientists are responsible for analysing and interpreting data to gain insights into business operations, customer behaviour, and other key areas. The adoption of a BD RA can impact data scientists in multiple ways, including the need to learn new tools and technologies to work with the new architecture, and the requirement to modify their existing data models to align with the new architecture.

5. **Business Analysts**: Business analysts are responsible for translating business requirements into technical specifications to develop software applications and other solutions. The adoption of a BD RA can impact business analysts in several ways, such as working closely with data scientists and data engineers to understand the organisation's data requirements and modify their approach to requirements gathering and analysis.

6. **Platform Engineers**: Platform engineers are responsible for managing and

maintaining the platform that hosts an organisation's BD infrastructure. The adoption of a BD RA can impact platform engineers in several ways, including the need to learn new tools and technologies to manage the new infrastructure, and the requirement to modify existing platform components to align with the new architecture.

7. **IT Executives**: IT executives are responsible for setting the strategic direction of an organisation's IT initiatives, including BD initiatives. The adoption of a BD RA can impact IT executives in several ways, including the need to align the new architecture with the organisation's overall IT strategy, and the requirement to ensure that the new architecture meets the organisation's business needs.

6.3 Software and System Requirements

As the result of the processes conducted (see Section 2.9.3), and by carefully evaluating similar approaches to requirement specification, a set of requirements for the development of artefact is tailored. These requirements are presented in terms of BD characteristics in Table 6.1. The definition of each BD characteristic is defined in Section 3.8.

Table 6.1: Metamycelium software and system requirements

Volume	**Vol-1)** System needs to support asynchronous, streaming, and batch processing to collect data from centralised, distributed, and other sources **Vol-2)** System needs to provide scalable storage for massive data sets
Velocity	**Vel-1)** System needs to support slow, bursty, and high throughput data transmission between data sources **Vel-2)** System needs to stream data to data consumers in a timely manner **Vel-3)** System needs to be able to ingest multiple, continuous, time-varying data streams **Vel-4)** System shall support fast search from streaming and processed data with high accuracy and relevancy **Vel-5)** System should be able to process data in a real-time or near real-time manner
Variety	**Var-1)** System needs to support data in various formats ranging from structured to semi-structured and unstructured data **Var-2)** System needs to support aggregation, standardisation, and normalisation of data from disparate sources **Var-3)** System shall support adaptations mechanisms for schema evolution **Var-4)** System can provide mechanisms to automatically include new data sources

Table 6.1: Metamycelium software and system requirements

Volume	**Vol-1)** System needs to support asynchronous, streaming, and batch processing to collect data from centralised, distributed, and other sources **Vol-2)** System needs to provide scalable storage for massive data sets
Value	**Val-1)** System needs to able to handle compute-intensive analytical processing and machine learning techniques **Val-2)** System needs to support two types of analytical processing: batch and streaming **Val-3)** System needs to support different output file formats for different purposes **Val-4)** System needs to support streaming results to the consumers
Security & Privacy	**SaP-1)** System needs to protect and retain the privacy and security of sensitive data **SaP-2)** System needs to have access control, and multi-level, policy-driven authentication on protected data and processing nodes.
Veracity	**Ver-1)** System needs to support data quality curation including classification, pre-processing, format, reduction, and transformation **Ver-2)** System needs to support data provenance including data life cycle management and long-term preservation.

6.4 Theory

In mathematical terms, an inflection point is a significant juncture where the curvature of a curve changes direction, signifying the transition from one behaviour to another. This pivotal point is marked by a dissolution of the prior shape and the emergence of a novel form. Today, there are empirical signals and derivers that point in the direction of

a seismic shift (*Building a High-Performance Data and AI Organization*, 2023).

A glimpse of the New Vantage Partners report in 2023 (Partners, 2023) communicates that despite potential economic headwinds, the investments in data are growing and remain strong. Nevertheless, the report also communicates that only 23.9% of companies identify themselves as data-driven.

In Chapter 4, major architectural challenges facing companies in the realm of BD are outlined. This chapter shifts its focus to the current state of big data architectures and the specific approaches proposed to address these challenges. Primarily, this chapter concentrates on the design and development of Metamycelium, a novel RA that aims to tackle the architectural challenges associated with big data adoption.

Before delving into the design and development of Metamycelium, it is essential to establish a clear communication method for architectural constructs. This study employs the architecture constructs definition from ISO/IEC 42010 (International Organization for Standardization (ISO/IEC), 2017), described as "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution".

To comprehensively understand Metamycelium, it is vital to outline its components. For this purpose, Richards and Ford (2020) approach of thinking about software architectures is utilised. In this approach, the software architecture consists of the following elements:

- Structure: refers to the architectural styles (distributed, layered, monolith, among others)
- Architecture Characteristics: refers to the success criteria or non-functional requirements of the system (availability, maintainability)
- Architecture Decisions: refers to rules on how the system should be constructed.

- Design Principles: refers to guidelines on how the system should be constructed.

Consistent with Archimate and TOGAF's categorisation of architectural elements into structures and behaviours, this research adapts these definitions for Metamycelium. Herein, structure is termed architectural style, architectural decisions are referred to as Architectural Rules (ARs), and design principles are denoted as Architectural Principles (APs). Architectural characteristics are sourced from ISO/EIC 25010 (Estdale & Georgiadou, 2018), while architectural styles draw from the work of Richards (Richards & Ford, 2020). Neither architectural characteristics nor styles are redefined in this work.

6.4.1 The monolith

While the tools and technologies of data engineering have reached a scale and diversity reminiscent of the Cambrian explosion (Figure 6.1), the underlying assumptions that govern the data architectures have not been much challenged.

According to Richards and Ford (2020) there are two major categories of architectures; 1) monolithic (deployed as a single unit) and 2) distributed (the system is made up of sub-components that are deployed separately). Today, most data architectures are in the first category.

While starting as a monolith can be a simple and good approach for building a data-intensive system, it falls short as the solution scales. While this assumption is challenged in the software engineering world, data engineering seems to still be driven by monolithic designs (Ataei & Litchfield, 2022). These designs are enforced by enabler technologies such as data warehouses and data lakes. In addition, many organisations and books adopt the idea of a *single source of truth*.

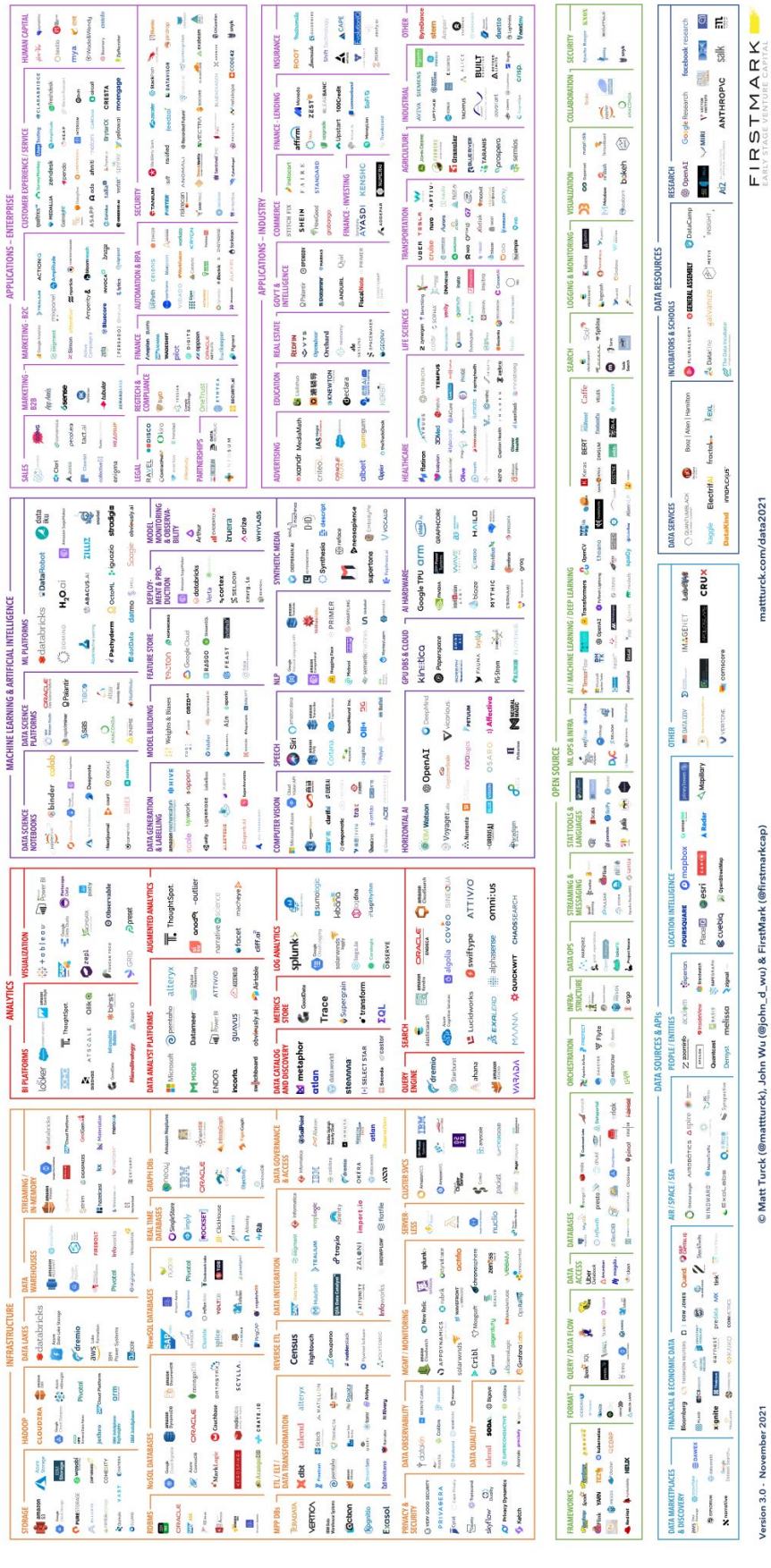


Figure 6.1: Cambrian Explosion of Big Data Tools and Technologies

The data chasm

Analytical data and operational data are two distinct types of data used in businesses. Operational data is used to manage day-to-day business operations, while analytical data is used to support strategic decision-making by identifying patterns and trends in historical data.

Many of the challenges of current BD architectures rely on their fundamental assumption of dividing operational and analytical data. While operational and analytical data have different properties and are processed differently, bringing operational data far from its original source can affect its integrity negatively, create organisational silos, and produce data quality issues.

These two planes of data are usually operated underlying different organisational hierarchies. Data scientists, business intelligence analysts, machine learning engineers, data stewards, and data engineers are usually under the leadership of the Chief Data and Analytics Officer (CDAO) and are heavily involved in creating business value out of data. On the other hand, software engineers, product owners, and quality assurance engineers are usually working with the Chief Technology Officer (CTO).

This has resulted in two segregated technology stacks and heavy investments in bridging the two. This chasm has resulted in two different topologies and fragile integration architectures through ETLs (Figure 6.2). This is usually achieved by some sort of batch ETL job that aims to extract data from operational databases. These ETLs usually do not have any clearly defined contracts with the operational database and are sheer consumers of its data. This highlights the fragility of this architecture, as the upstream changes from operational databases can affect downstream analytical applications. Over time, ETL job complexity increases, maintainability becomes harder, and data quality decreases.

Many of the technologies created over the years are developed underlying this very

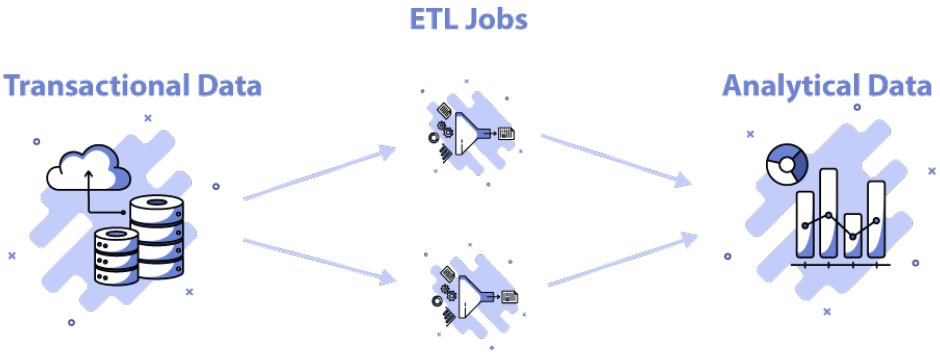


Figure 6.2: The great divide of data

assumption. While these technologies are effective in handling the volume, velocity, and variety of data, today's data challenges are about the proliferation of origins, data quality and data architecture.

These challenges include the proliferation of data origins, ensuring data quality across diverse sources, and designing data architectures that can seamlessly integrate and process data from various systems in real-time or near-real-time (Manyika et al., 2011; Davenport, Barth & Bean, 2012).

Data are usually collected and consolidated through several systems. Some of these data may even go beyond the perimeters of the organisation. Herefore, based on the premises discussed hereinabove and the challenges explicated in Section 4.6.7, it is posited that today's data architectures need a shift from the centralisation of data in one big analytical database to connecting analytical data wherever it is.

Based on this, the artefact designed for this study learns from past solutions and addresses their shortcomings. This artefact aims to walk away from overly centralised and inflexible data architectures that act as a coordination bottleneck. Therefore, one of the objectives of this artefact is to bridge the gap between the point where data is generated and the point in which it is used, thus simplifying the process. This artefact

aims to increase agility in the face of growth and respond effectively to organisational changes. Therefore the architectural style of Metamycelium is as follows:

> Architectural Style (AS): Metamycelium adopts the principles of microservices architecture

Localised autonomy through domain-driven decentralisation

Today's businesses are dealing with a great deal of complexity. A typical business is made up of various domains with different structures. These domains change at different rates and tend to be quite isolated from each other. The overall synergy of the business is dictated by the relationship between these domains and the ways in which these domains evolve.

At the heart of these synergies sits volatility and rapid change in the market and an ever-increasing number of regulations. How do businesses today manage the impact of these changes to their data? Should they constantly modify ETL jobs, create new ETL backlogs, and consolidate data into operational stores? How can businesses create quality and trustworthy data without slowing down? The answer lies in embracing and adapting to the changes in today's data landscape.

One way to tackle this complexity is to align technology with business. Businesses break down their problem into smaller problems that are handled in each domain; technology and data can be incorporated into those domains too. This approach is well-established in microservices architectures (Ataei & Staegemann, 2023). Therefore, the first and foremost rule of Metamycelium is:

> Architectural Rule 1 (AR1): In order to optimise performance and efficiency, it is important that analytical and operational data be located as close to each other as possible. "Close to each other" refers to minimizing network latency and data transfer times between storage and processing nodes.

Domain-driven design

Integral to Metamycelium, is the distribution and decentralisation of services into domains that have clear bounded context. Perhaps one the most challenging things one might face when it comes to architecting a distributed system is: based on what architectural quanta should the system be broken down? This issue has been repeatedly discussed for example among adopters of microservices architecture (Dehghani, 2022). Metamycelium, inspired by the concept of domain-drive design (DDD), tends to sit data close to the product domain that relates to it (Evans, 2004). This implies that data inheres in the product domain and as a facet of it (Laigner, Zhou, Salles, Liu & Kalinowski, 2021b).

This is mainly driven by the fact that most organisations today are decomposed based on their products. These products are the capabilities of the business that are segregated into various domains. These domains usually define their bounded context, evolve at different rates, and are operated by cross-functional teams (Skelton & Pais, 2019). Incorporating data into these bounded contexts can result in a synergy that can improve the management of continuous change.

This can be micro, such as application developers communicating with data engineers about collecting user data in nested data structures, or macro, such as application developers thinking about redesigning their Graphql schema in an intermediary layer that may affect the downstream analytical services. Therefore, the concept of DDD is absorbed into this study to facilitate communication and increase the adoption, rigour and relevance of Metamycelium.

DDD is an approach to software development that focuses on understanding and modeling the problem domain of a software application. The goal of DDD is to create software that reflects the language, concepts, and behaviours of the problem domain, rather than being based solely on technical considerations.

DDD can help Metamycelium by providing a systematic approach to modeling and managing data that is closely aligned with the problem domain of the application. By focusing on the language, concepts, and behaviours of the problem domain, DDD can help data architects gain a deeper understanding of the data that is needed and how it should be structured (Eridaputra et al., 2014; Dehghani, 2022)

Furthermore, communication is a key component of any software development endeavour (Sudhakar, 2012), and without it, essential knowledge sharing can be compromised. Often data engineers and business stakeholders have no direct interaction with one another. Instead, the domain knowledge is translated through intermediaries such as business analysts or project managers to a series of tasks to be done (Khrononov, 2021). This implies at least two translations from two different ontologies.

In each translation, information is lost, which is the essential domain knowledge, and this implies a risk to the overall data quality. In such a data engineering process, the requirement often gets distorted, and the data engineer has no awareness of the actual business domain and the problem being addressed.

Often, problems being solved through data engineering, are not simple mathematical problems or a riddle but rather have broader scopes. An organisation may decide to optimise workflows and processes through continuous data-driven decision-making, and a data architecture that is overly centralised and not flexible can risk a project failure. Based on these premises, the first principle of Metamycelium is as follows:

Architectural Principle 1 (AP1): The domain that generates the data, should own that data.

Complex adaptive systems

Architectures like Metamycelium share properties with complex adaptive systems (Holland, 1992). The artefact designed for this thesis is especially inspired by the idea that powerful groups can emerge from an array of simple rules governing local

agents. In one study, Reynolds (1987) analysed a synchronised flock of starling birds in autumn. This study presented the fact that every starling bird follows three simple rules: 1) alignment (following flockmates that are close by), 2) separation (so birds do not collide with each other), and 3) cohesion (keeping the same pace as the neighbouring flockmates). These rules can be mathematically expressed as follows:

Alignment:

$$v_i(t+1) = v_i(t) + \frac{1}{k}\sum_{j=1}^{N}(v_j(t) - v_i(t)) \tag{6.1}$$

where $v_i(t)$ is the velocity vector of bird i at time t, k is a normalisation factor, and N is the number of neighbouring birds.

Cohesion:

$$v_i(t+1) = v_i(t) + \frac{1}{k}(c_i(t) - p_i(t)) \tag{6.2}$$

where $c_i(t)$ is the center of mass of the neighbouring birds, $p_i(t)$ is the position of bird i at time t, and k is a normalisation factor.

Separation:

$$v_i(t+1) = v_i(t) + \sum_{j=1}^{N}\frac{(p_i(t) - p_j(t))}{d_{ij}^2} \tag{6.3}$$

where $p_i(t)$ is the position of bird i at time t, $p_j(t)$ is the position of bird j at time t, and d_{ij} is the distance between birds i and j.

Starling birds do not need a centralised orchestrator to create this complex adaptive system. In Metamycelium the aim is to promote a domain-driven distribution of data ownership. This architecture is modeled in a way that a domain does not provide only operational data through a standard interface, but does provide analytical data too. For instance, in practice management software for veterinaries, the animal domain provides operational APIs for updating animal attributes, but it can also provide analytical interfaces for retrieving animal data within a window of time. Every domain owns its

data.

In this fashion, the domain can also choose to retrieve data from other domains with some sort of discovery mechanism, process the data and enrich its data. In some cases, there can be a creation of aggregate domains with the main concern of aggregating data from various domains and providing it for a specific use case.

This is to remove vertical dependency and allow teams to have their local autonomy while being empowered with the right level of discovery and APIs. This architecture promotes the idea of coequal nodes consolidated to achieve the overall goal of the system rather than a centralised database of all data owned by people who do not have domain knowledge. This concept is inspired by DDD in the book presented by Evans (2004), data mesh (Dehghani, 2020), and microservices architecture (S. Newman, 2015b). Thus the next rule is as follows:

Architectural Rule 2 (AR2): Systems should be distributed and decentralised into domains, promoting the concept of local autonomy.

The graphical representation of this concept can be seen in Figure 6.3.

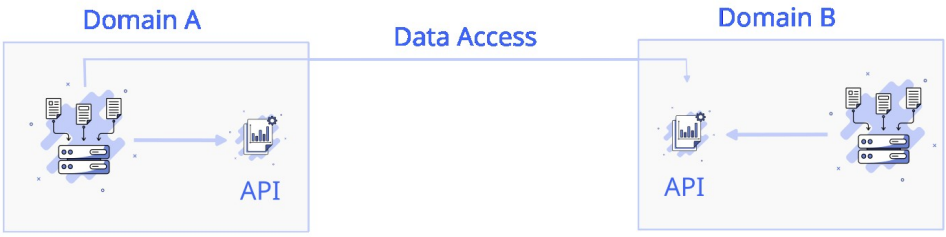


Figure 6.3: Localised autonomy through domain-driven decentralisation

6.4.2 A New Architectural Quantum

According to Ford, Parsons and Kua (2022), an architectural quantum is the smallest unit of architecture that has high cohesion, includes all the structural elements to meet its ends and is independently deployable. In the case of Metamycelium, the architectural quantum is the data domain. This domain should encompass all the structural elements required to achieve its functionality, should be able to have a separate lifecycle, and should be able to evolve without introducing side effects to other domains.

The transition from monolithic n-tier architectures to SOA presented challenges, notably in maintaining the Enterprise Service Bus (ESB), which became a bottleneck due to the bloated transformation and logic operations. The emergence of microservices architecture addressed these issues by promoting a shift from complex pipelines to simpler ones, paired with smarter, self-contained services, often designed following domain-driven principles (Ataei & Litchfield, 2023).

Although microservices bring their own complexities, they represent an evolutionary step in software engineering akin to advancements in data engineering, where the focus shifts from the tightly-coupled nature of traditional ETL processes to more agile and decomposed approaches.

Metamycelium scales up by the addition of new data domains. That is, data domains are the axis over which the system scales. This is in striking contrast with current centralised architectures that chose their architectural quantum based on technology. Based on that premise, the next principle is as follows:

> Architectural Principle 2 (AP2): The architectural quantum should align with the business, not technology.

To create an effective data product, it is important to include not only the data itself but also the necessary infrastructure and systems to ensure that the data is easily accessible, understandable, and secure. Additionally, the product should be designed to

operate independently while following all relevant regulations and policies.

At a minimum, a data domain should include four structural components: data, metadata, code, and configurations (including infrastructure dependencies). Figure 6.4 provides a graphical representation of these structures.

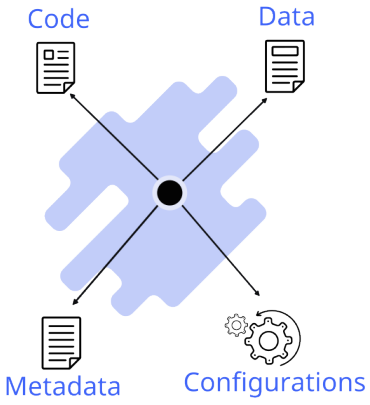


Figure 6.4: Dota domain as the architectural quantum

Data

Data is the core of what a data domain provides through a standard interface. This implies that the domain owns and stands accountable for the lifecycle of its data. Depending on the nature of the contexts and technology stacks, the data domain can provide its data in various formats including columnar, tabular, and files. The domain can even choose to provide its data through storage systems. The domain must adhere to a set of data contracts and global policies.

Metadata

Integral to the usability of data, is the metadata. Metadata is important because it describes the dataset to potential users without the effort of digging through the dataset.

In addition, it plays a key role in maintaining the data, its quality and its lifecycle. For instance, a data domain may include metadata about data semantics, data types, data structure, statistical characteristics and service level objectives (SLOs).

As opposed to current BD architectures where metadata is usually centralised into some sort of metadata management system, the data domain itself is responsible for extracting, extrapolating and providing metadata.

Code

In this architectural approach, domains independently own their business logic, analytical data, version history, and access management. They should own their code, executing within their bounded context. Various code type, transformation code, API code, and discovery and metadata handling code are included within a domain. Currently, BD architectures typically externalise this as a separate artefact known as a data pipeline. Metamycelium dispenses with external pipelines, favoring internal data transformation codes. Thus, the next architectural rule is:

> Architectural Rule 3 (AR3): The domain should entail all the necessary structures to provide its data. These structures are data, metadata, code, and configuration.

This transformation code aligns with the domain's business logic and particularities. The data model is shaped by the team's objectives for analytical and transactional data. The domain is responsible for data cleansing and quality, ensuring metrics like timeliness, integrity, accuracy, completeness, and consistency. Thus, the next architectural principle is:

> Architectural Principle 3 (AP3): The domain should adhere to a set of data quality metrics. These metrics should be set from the governance layer.

Every domain should be enhanced with discoverability mechanisms, SLOs, metadata and usability information. The API should follow a contract defined by the domain,

reflecting the system's context and architecture. This approach allows data engineers and developers to build against a stable contract, preventing negative impacts on downstream consumers. Additionally, the domain should have its input data API, ensuring a clear ingress for upstream data reads

In essence, the domain should maintain three fundamental APIs: 1) ingress APIs, 2) egress APIs, and 3) metadata APIs. Figure 6.5 illustrates this concept.

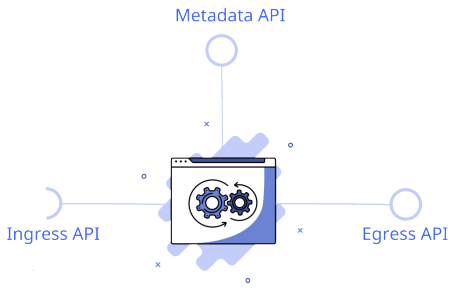


Figure 6.5: Data Domain APIs

Therefore, the next architectural rule is as follows:

Architectural Rule 4 (AR4): The domain should be discoverable, providing three main APIs: ingress API, egress API, and metadata API.

Configuration

The domain should encapsulate codes that are responsible for executing and configuring various structural and behavioural policies such as run-time environments, access control, compliance, encryption and privacy. Policies executed by the domain contain two major facets 1) domain-driven policies, and 2) federated governance policies. Domain-driven policies are the policies that the team generate based on the particularities of its product. Federated governance policies are the high-level standard rules that come from the federated computational governance layer. This layer is discussed in detail in Section 6.4.5.

While domains are usually constituents of transactional and operational services, a domain can be spawned purely for analytical purposes, in this case, the domain is an *aggregation domain* that is purely made for data analysis. A data domain can also be made for the requirements of an external party or integration.

6.4.3 Data as a First-class Citizen

The Metamycelium architecture prioritises treating data as a valuable asset, in contrast to traditional data management approaches where data is often considered a byproduct of systems. In this approach, data is managed by dedicated teams, treated as long-lived products rather than one-off projects, and aligned with business capabilities. The goal is to promote a data-driven culture and enable organisations to effectively leverage their data for business outcomes (Dehghani, 2022).

In the context of Metamycelium, treating data as a first-class citizen means:

- **Data is managed by business domains**: Data are managed by dedicated teams that have full ownership and accountability for the data product's end-to-end lifecycle. These teams are responsible for defining and publishing the data product's schema, quality standards, and access policies, as well as ensuring that the data product is discoverable and accessible by other teams across the organisation.

- **Data are treated as products, not projects**: Data are made discoverable to other teams across the organisation via a centralised data catalogue or metadata repository, termed Data Lichen in Metamycelium, as discussed in Section 6.4.9. This simplifies access to data products without navigating complex silos or relying on central IT.

- **Data products are aligned with business capabilities**: Data products are aligned

with business capabilities, rather than technical domains or systems. This means that data product teams are responsible for understanding the needs of the business and designing data products that support those needs.

By treating data as a first-class citizen, Metamycelium aims to foster a culture of data-driven decision-making and enable organisations to leverage their data assets to drive business outcomes. Therefore the fourth principle is as follows:

Architectural Principle 4 (AP4): Data should be treated as a first-class citizen and not just a byproduct of transactional systems.

6.4.4 Data Platform as a Service

As product domains increase, the complexity and volume of infrastructure-related tasks, such as building, deploying, executing, and monitoring services, along with the necessary data pipelines also escalate. These platform skills, predominantly found in DevOps and site reliability engineers, are not commonly possessed by application developers and data engineers at a local level. Consequently, there is a pressing need for abstract, reusable infrastructure components that can be utilised with ease, ensuring that teams are supported by infrastructure as a service tailored to BD requirements (Dehghani, 2022).

Infrastructure as a service can be effectively implemented using infrastructure-as-code tools such as Terraform (HashiCorp, 2023a) coupled with GitOps principles (*GitOps*, 2023). However, extending this infrastructure to accommodate the BD ecosystem, which is growing rapidly and often involves disparate systems such as Amazon's EKS clusters and Databricks, presents challenges in creating a cohesive, cost-effective, and interoperable setup.

Moreover, prevailing BD architectures tend to favor a centralised platform model, as exemplified by the NIST BD RA framework provider. This model suggests a unified

platform that could inadvertently lead to over-centralisation, either through a single vendor or a patchwork of tools from multiple vendors. While initially simpler, these models can introduce issues such as vendor lock-in, restrictions on data hosting, and limitations on functionality for bespoke needs. Furthermore, integrating products from different vendors can result in a lack of standardisation, complicating maintenance and scaling efforts.

Metamycelium learns from these limitations and embraces the fact that organisational complexity can create a multi-platform environment. To enable teams with strong APIs and to promote the concept of *platform as a service*, platform engineers must first look into designing an effective experience for day-to-day data work. This implies that the platform engineers are not only concerned with the technicality of what they're doing, but they are ultimately engineering an *experience*. Therefore the next principle is as follows:

> Architectural Principle 5 (AP5): One of the platform's first objectives shall be on creating an effective experience for day to day operation of data engineers.

This process can include the study of a data developer or data engineer's journey and how they interact with platform APIs. This relationship is depicted as an 'x-as-a-service' facilitation relationship in Team Topologies presented by Skelton and Pais (2019). Based on these premises the next architectural rule is as follows:

> Architectural Rule 5 (AR5): The platform architecturally should not be overly centralised and inflexible. This architecture should embrace the dynamics of today and manifest itself as facilitating 'as-a-service' with a standard and easy-to-use interface.

The three layered architecture

Metamycelium proposes a three-tiered platform architecture to manage data complexity, consisting of resources, infrastructures, and composites. Resource modules are the

fundamental units, each representing a single infrastructure component like an AWS S3 bucket. Infrastructure modules group related resources to form complete components, such as a database cluster, encompassing elements like security groups and virtual private clouds. Composite modules, the highest level of abstraction, combine these infrastructure modules to create comprehensive environments for production or development, offering scalable and maintainable structures.

The three-layered platform architecture is inspired by Terraform's official documentation (HashiCorp, 2021), the 'Terraform Up and Running' book by Brikman (2017), and the works by Morris (2016). Figure 6.6 provides a graphical representation of the three layered platform architecture.

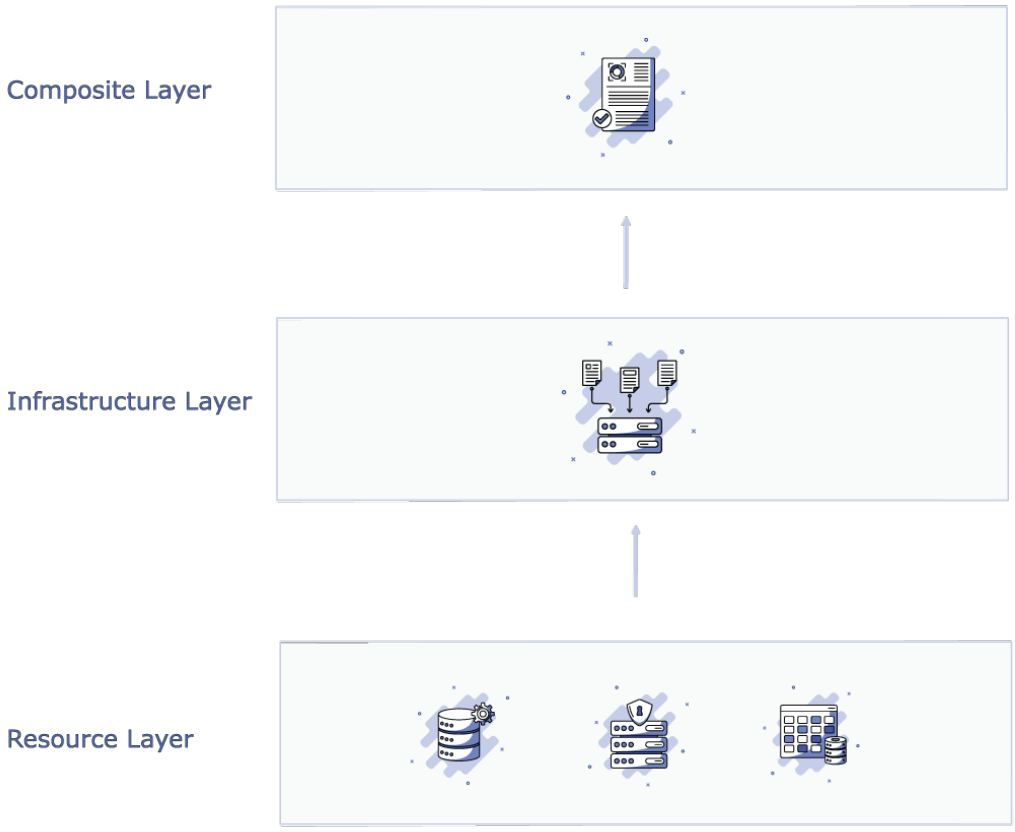


Figure 6.6: Three Layered Platform Architecture

This platform architecture should be applied to two major areas: 1) the domains and

2) the *data lichen*. Data lichen is a word devised to name an architectural component of Metamycelium. Data lichen is a major part of Metamycelium as it is a platform in which data engineers can get access to data products available from the company. It serves as a discovery mechanism. It supports various operations such as searching for data sets, traversing the lineage, and providing metadata and data quality metrics. Data lichen relies on interfaces provided by the domains as it collates and aggregates them to a standard view. This construct is described in Section 6.4.9.

The platform plane is inspired by the concept of *affordances* by Norman (2013). In his works, Norman defines affordance as a relationship between the properties of an object and the capabilities of an agent. Therefore, the platform should afford experiences such as easy-to-use and standard platform interfaces, and discovery and addressability of data products.

6.4.5 Federated Computational Governance

Metamycelium is designed to be a mesh of decentralised and distributed co-equal nodes that constantly evolve. This is a complex adaptive system that reacts to change by either adding, altering or removing nodes. Domains independently own and serve their analytical data via interfaces. These co-equal nodes utilise a set of self-server data infrastructures to achieve their means. This promotes the concept of local autonomy, reduces side effects, removes centralisation and improves the maintainability and scalability scores of the architecture. Nevertheless, that's not enough.

Metamycelium acknowledges that as systems expand and become more interconnected, cross-functional teams face greater challenges in scaling and maintaining services that must efficiently interoperate and process data. This complexity can lead to entropy at interaction points, which, without proper governance, may cause confusion, faults, and security vulnerabilities, as highlighted by the Open Web Application Security

Project (OWASP) (Open Web Application Security Project, 2017), or compliance issues with regulations like the GDPR, California Consumer Privacy Act (CCPA), or Personal Data Protection Act (PDPA). Additionally, the absence of data architects can result in flawed data models and the selection of suboptimal tools for data management.

Hence, Metamycelium advocates for a governance model that ensures data across the organisation is managed with an emphasis on quality, security, consistency, compliance, performance, privacy, and usability. This approach necessitates a shift away from traditional data engineering practices, which often rely on a central canonical data model and comprehensive processes for data validation and certification, roles typically filled by data stewards and custodians. However, such centralised models can impede the agility of an organisation, slowing down the time to insight and constraining experiment-driven development, which are essential for maintaining a competitive edge in a rapidly evolving market (Baum, Münch & Ramler, 2016).

Metamycelium embraces constant changes in the data landscape and does not aim to kill the creativity and innovation of the teams. In this architecture, a representative from each domain joins the governance group, crystalizing the concept of *federated governance*. This results in the next architectural principle follow:

> Architectural Principle 6 (AP6): Governance policies should be created through a federation of domain representatives.

These representatives are then accompanied by platform developers, specialists such as lawyers, security engineers, and organisational executives. The aim is to capture the cross-cutting concerns across all aspects of the company. This is to capture the wisdom of the crowd and project it through a set of well-defined policies. This also results in the creation of the *operation model* that provides consistent experiences across domains. Therefore the next rule is as follows:

> Architectural Rule 6 (AR6): Domains should function autonomously but in the river of global standards, policies and operational models that are laid down by the federated governance team.

This concept is inspired by *Systems thinking* presented by Meadows (2008), data mesh depicted by Dehghani (2020), the works of Eryurek et al. (2021) on data governance, and TOGAF's ADM architectural principles (Josey, 2016).

Application of Policies Through Automation

As discussed above, it is imperative for Metamycelium to operate with the presence of federated computational governance. Nevertheless, the governance layer should not slow down the agility and productivity of the domains. The governance layer policies and standards should be applied to domains in an automated way. This results in the next architectural principle as follows:

> Architectural Principle 7 (AP7): Governance policies should be applied to domains in an automated manner.

This automation is achieved by four logical components; 1) the sidecar, 2) the data plane, 3) the communication plane, and 4) the configurations and policies. The sidecar is a lightweight proxy attached to every domain's services. Sidecar run in the same execution scope as the domain's services; be it the Docker container the app is running on or an EC2 instance. The sidecar implements the policy execution and other cross-cutting concerns such as authentication, authorization, token handling and revalidation, runtime configuration, and others. The sidecar can intercept all incoming and outgoing traffic, controlling the communication between domains. The side car is deployed with the domain's code itself.

As these proxies are deployed to each service, they intercept traffic and are capable of communicating with each other too. This creates a data mesh that governs all traffic

among all domains. Proxies are like the dots in the general patterns of communication. These proxies mediate and control communication. In addition, they collect and report telemetry data.

The next logical component is the control plane. The control plane is responsible to leverage standards and policies that come from the governance layer. This includes the configuration codes that have been discussed in Section 6.4.2. The control plane is like a control tower that looks over processes and enforces policies through special high-level leverage.

The concept of a service mesh is to provide a dedicated infrastructure layer for managing services in the domain in an automated way. This layer is responsible for handling cross-cutting concerns so data engineers do not have to rewrite privacy, security and runtime mechanisms. Furthermore, this layer serves as a mechanism for automatically applying global policies that have been generated from the federated governance layer.

Service mesh increases visibility and control over domain-to-domain communication, make it easier to understand and diagnose issues, and allows domains to perform autonomously without having to create backlog tickets for platform engineers. In turn, this increases the security, privacy, resiliency and scalability of the Metamycelium. The concept of service mesh, as depicted in Figure 6.7, originated with Istio, a joint project by Google, IBM, and Lyft (Istio, 2018).

6.4.6 Access Control, Identity, and Encryption

It is imperative for the policies to be standardised and consistent. This enables the domain to avoid mismatching interfaces or getting into faulty states. Having a standard policy for access control, identity and encryption reduces complexity, increases interoperability, reduces maintainability costs, and helps with scalability. Given that

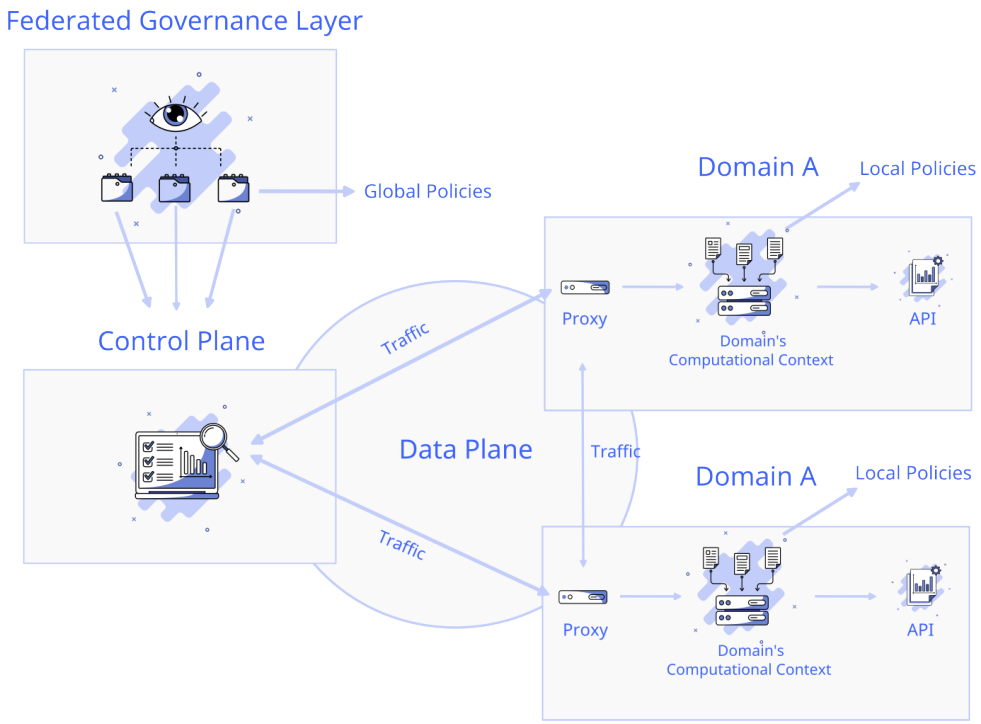


Figure 6.7: Federated Computational Governance

the standard chosen is an open standard, the system can easily integrate with other third-party systems outside the perimeter of the organisation, and this in turn increases the interoperability of the architecture.

For instance, authentication and authorization are two key components of any distributed system. Nodes need to communicate with each other safely through a secure protocol and port. This issue is further highlighted when the domain has to serve its data to external consumers. An effective system must be able to confidently identify users or systems whether internal or external.

If there's no standard way of accessing and sharing data in Metamycelium, it would be impossible to enable data sharing among domains. The more diversified the approach to authentication and authorization, the higher the cost of maintenance, and chances of friction. While this sounds intuitive and obvious, surprisingly, there's no industry-wide

adopted protocol for identity and access management for data management systems. This has been hardly discussed in the RAs depicted in Chapter 4, Section 4.1 as well. Based on this, the next architectural principle is as follows:

> Architectural Principle 8 (AP8): Cross-cutting concerns such as authentication, authorization and encryption should be standardised, ideally through an international open standard.

Just like identity and access control, the system should be governed by an array of standards associated with encryption. These standards should include data in motion, in transit, and at rest. For instance, the organisation may opt to use confidential computing techniques such as Trusted Execution Environments (TEEs) and secure enclaves (Sabt, Achemlal & Bouabdallah, 2015; Arnautov et al., 2016).

Taken all together, the next rule is as follows:

> Architectural Rule 7 (AR7): There should be an ideally one and one standard way of addressing cross-cutting concerns.

AR7 aligns with the principle of having lower entropy in software architecture. Lower entropy refers to a state of order, consistency, and predictability within the system.

6.4.7 Event Driven Services

Metamycelium's decentralised and distributed architecture introduces challenges in service communication as the network grows. Initially, simple point-to-point communication via REST API calls suffices, but this method proves inefficient with system expansion. Such synchronous interactions can lead to a *distribution tax*, where one service's blocking state, often due to intensive processes, causes delays in dependent services (Montesi & Weber, 2016).

The heavy network demands of distributed systems may further complicate matters, potentially causing *tail latency*, *context switching*, and *gridlocks* (Sriraman & Wenisch, 2018; Gan et al., 2019; Kakivaya et al., 2018; Ataei & Litchfield, 2021b). This tight coupling is at odds with the distributed system's goals of autonomy and resilience.

To overcome these issues, Metamycelium adopts asynchronous event-driven communication. This model enables services to publish and respond to events, thus decoupling their interactions. In this *publish and forget* framework, services announce events to specific topics and move forward without awaiting direct responses. This is similar to restaurant staff responding to environmental cues instead of direct commands, promoting a smooth operational flow.

While event-driven architectures typically offer eventual consistency, which might not be suitable for certain real-time stream processing scenarios requiring immediate consistency, it is a safe assumption that the majority of data engineering workloads can efficiently operate within an event-driven paradigm . Based on this the next principle is as follows:

> Architectural Principle 9 (AP9): Services should avoid point-to-point communications if possible and instead opt for an asynchronous event-based and reactive style of communication.

According to Richards and Ford (2020), event-driven architectures come in two major topologies: 1) broker topology and 2) mediator topology. Additionally, in the works of Stopford (2018) the concept of streaming platforms is elaborated. Since Metamycelium is a BD architecture that aims to process analytical data, a lot of challenges of achieving ACID transactions are eliminated. Therefore Metamycelium adopts the architectural concepts of *distributed asynchronous event-driven systems* using a hybrid topology. That is, Metamycelium is absorbing some elements of the broker topology, some elements of the mediator topology and many concepts from streaming platforms. Therefore in Metamycelium's CAP theorem (E. A. Brewer, 2000)

partition tolerance and availability are top guarantees.

There are five primary architectural components within this hybrid topology: the event, the event consumer, the event producer, the event backbone and the eventing interface. Events are initiated by the event producer and are dispatched to the topic of interest. This event then goes through the event broker and is stored there for retrieval in a queue-like indexed data structure. The event consumers that are interested in this topic will then listen to the topic of interest using the eventing interface. The event backbone itself is internally a distributed system that is made up of an arbitrary number of event brokers.

Event brokers are services that are spawned and provisioned to facilitate event communication through Metamycelium. These brokers are coordinated with a distributed service coordinator. Brokers do also allow for the replication of topics. Furthermore, to allow for fault tolerance and recoverability, the event backbone has a dedicated event archive. This archive aims to store all events that are going through the brokers so they can be restored to a correct state if a failure occurs.

These components work together to create a distributed, fault-tolerant, and scalable data system that can handle both batch and stream processing as portrayed in Figure 6.8.

6.4.8 Principles of Serving Data

In Metamycelium, data is a first-class citizen. This implies that data should be treated as a focal and essential component of the architecture. Data is increasingly dynamic, comes in various forms, evolves rapidly and even changes semantics. Therefore to address data in a maintainable way, some principles and architectural decisions should be taken into consideration.

Each domain needs to serve its data as a product, adhering to the integral principles that data must be discoverable, trustworthy, immutable, bitemporal, and accessible with

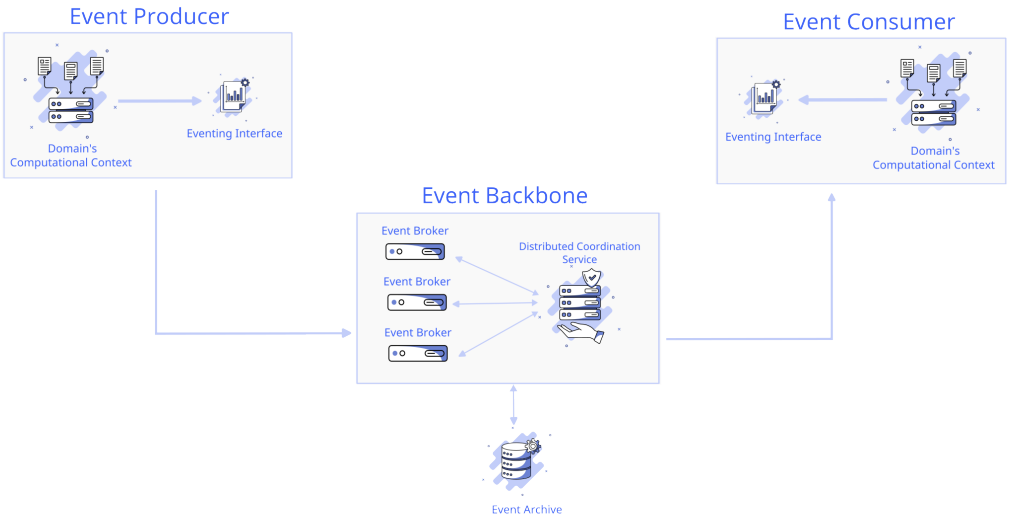


Figure 6.8: Metamycelium Event-driven Architecture

read-only permissions. This grounds the next architectural rule for Metamycelium as follows:

> Architectural Rule 8 (AR8): All domains that serve data should make their data discoverable through a standard interface.

Data discovery

Contemporary BD architectures commonly adopt two approaches to data discovery. The first involves retrospective curation, with data stewards amalgamating previously isolated data sets to establish a unified discovery platform. The second employs analytical algorithms to retrospectively extract metadata from data, thereby facilitating enriched discoverability (Dehghani, 2022).

Metamycelium deviates from these trends by centering discovery around the data's origin within each domain. This is achieved by domains exposing their data and semantics via APIs, coordinated by a central component, *Data Lichen*, detailed in Section 6.4.9.

Domain registration with Data Lichen is mandatory for participation in the data discovery process, and this is typically integrated into the domain's initialisation sequence. This includes provisioning a globally accessible URI that references the domain's metadata, data, and operational status as described in Section 6.4.5.

To render its data semantically transparent, a domain translates its semantics into standardised abstractions, which are then accessible in various formats, a process elucidated in Section 6.4.11. Data Lichen acts as a directory, providing initial metadata to consumers, who can then directly interface with the domain's URI for further data access, incorporating privacy safeguards like differential privacy when necessary (Dwork, 2006).

For exploratory and interpretive purposes, domains may offer computational notebooks to facilitate an interactive engagement with data, as outlined by (Wolfram Research, Inc., 2021). For automated interactions, data semantics and format choices are communicated through the domain's API, which, upon request, results in the data being stored in a distributed storage system. The domain then informs the consumer of the data's location for subsequent retrieval.

This decentralised storage model offers several advantages, such as reduced event backbone load and the efficient utilisation of distributed storage systems for handling large datasets, thereby enabling a more flexible and maintainable architecture (Figure 6.9).

However, this approach is not without limitations. To address diverse processing requirements, Metamycelium endorses a stream-first approach to inter-domain communication, highlighting streams' attributes of real-time processing, low latency, scalability, and immutability, which are paramount for contemporary data-intensive operations, as characterised by Urquhart (2020) in his discussions on streaming architectures.

These features render streams a pivotal element within Metamycelium, enabling not only real-time data dissemination but also ensuring data persistence for historical

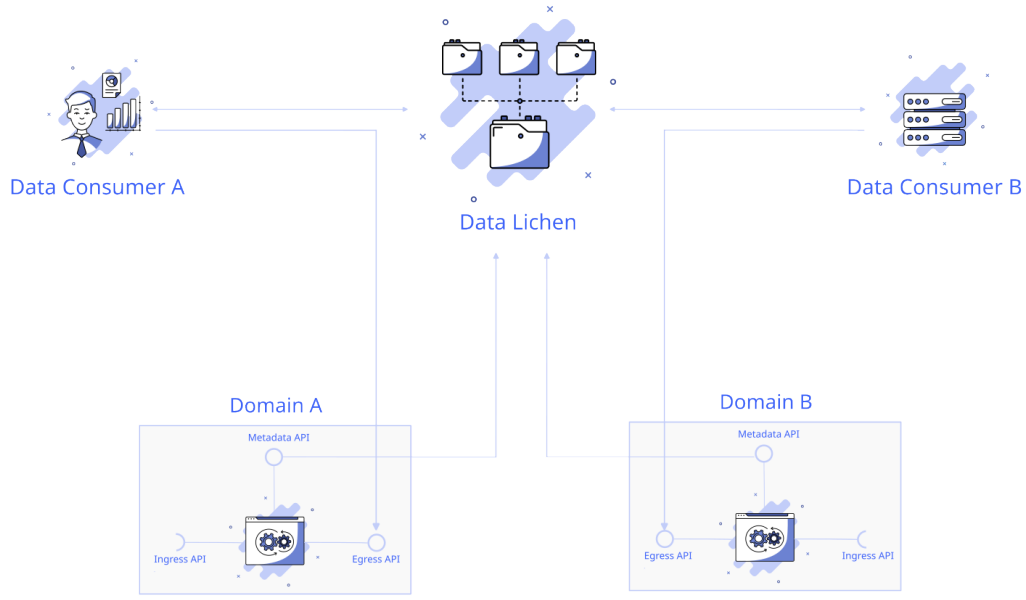


Figure 6.9: Data discovery through Data Lichen

analysis and governance. Therefore the next architectural principle is as following:

Architectural Principle 10 (AP10): Favour streams as the main data structure for inter-domain communication.

It is essential for data domains to publish either stream or batch-processing data with backwards-compatible schemas.

6.4.9 Data Lichen

Data Lichen operates as a centralised architectural quantum within Metamycelium, designed to streamline the discovery of data by accumulating and presenting metadata from various data domains. It facilitates the identification and access to data through uniquely addressable URIs, thereby addressing the challenges associated with data

discovery in distributed environments. The platform enhances observability, ensures data integrity, and integrates automated feedback mechanisms, which are in line with the systemic principles of feedback loops and leverage points (Meadows, 2008).

The primary function of Data Lichen is the consolidation of metadata from across the Metamycelium landscape to aid in data discovery, a concept reminiscent of network routing paradigms (Tanenbaum & Wetherall, 2011). Its functionality is predicated upon the robust infrastructure services outlined in Section 6.4.4 and leverages the metadata interfaces provided by the data domains, as expounded in Section 6.4.2.

> Architectural Rule 9 (AR9): A centralised architectural construct shall exist within Metamycelium to facilitate the discovery of data across domains.

6.4.10 Data Quality for Establishing Trust

For the seamless evolution of Metamycelium, it is incumbent upon domains to proactively and assiduously assess their data products through a suite of well-defined artefacts known as metrics. These metrics illuminate various dimensions of the domain's data corpus, including but not limited to data quality metrics, temporality metrics, and bespoke domain-driven metrics.

Primarily, a data domain must embrace the data quality metrics delineated by the governance stratum, striving to fulfil the minimum stipulated Service Level Objectives (SLOs). While these data quality metrics are context-sensitive and are not precisely delineated herein, it is posited, as elucidated by Moses, Gavish and Vorwerck (2022), that measurement is a prerequisite for rectification. In Metamycelium, the exigency for quality data underpins daily operations and pivotal decision-making processes, heralding the following architectural mandate:

Architectural Rule 10 (AR10): All domains must comply with a prescribed set of data quality benchmarks.

The ensuing data quality dimensions are posited as foundational:

- Completeness: The extent of data inclusiveness, encompassing all requisite elements.
- Validity: The extent to which data is congruent with established standards and criteria, affirming its intended utility.
- Accuracy: The extent of data precision, devoid of errors or distortions.

An additional suite of metrics to be contemplated pertains to temporality. Given the dynamic nature of data across the temporal spectrum, temporal metrics provide critical insights into the contemporaneity of data vis-à-vis specific applications.

- Last processing time: The juncture of the latest successful data procession.
- Timeliness: The degree of data recency and pertinence relative to its intended application.

In conclusion, domains may elect to proffer custom metrics tailored to the data consumer, such as the latest data access timestamp, deprecated fields, and datasets currently under revision. The aforementioned metrics draw inspiration from two laudable W3C open-source standards devised for the standardisation of consistent data metrics: 1) Data on the Web Best Practices: Data Quality Vocabulary (World Wide Web Consortium, 2017) and 2) Data Catalog Vocabulary (DCAT) - Version 2 (World Wide Web Consortium, 2014). These standards are endorsed and promulgated within Metamycelium to catalyse a homogenised lexicon and best practices, fostering an open data ecosystem on the web.

6.4.11 Data Modeling

Beyond the principles of data provision discussed in Section 6.4.8, the domain is obliged to construct a coherent representation of its data, which involves conveying the intrinsic semantics of the business elements. The choice of representation, be it an entity-relationship model, a columnar model, or a document model, should align with the domain's business logic intricacies. Based on this the next principle is as follows:

> Architectural Principle 11 (AP11): Domains must rigorously model their data in close alignment with the reality of their business contexts.

For the Metamycelium to achieve heightened interoperability, domains ought to offer semantics that are both human-readable and machine-readable, enabling systematic validation. A domain, depending on its data type and requirements, may utilise JSON Schema (*JSON Schema*, 2019) for general purposes.

Such adherence to standardised semantic definitions, as endorsed by the computational governance layer, is conducive to a broad spectrum of automated verification processes, including contract-driven testing, static code analysis, integration testing, and comprehensive data quality assurance, thereby fostering enhanced interoperability both internally and externally to the enterprise.

6.4.12 Data Composition

Analytical and machine learning workloads necessitate the interconnection and consolidation of data across domains. The prevalent BD architecture frameworks, as elucidated in Chapter 4, typically employ explicit relationships for data composability. Typically, this involves extracting data from transactional systems or data lakes, followed by cleansing and modelling using dimensional schemes, such as star or snowflake schemas (Kimball & Ross, 2013). However, such centralised methods of data composability

prove suboptimal for decentralised architectures, as they can become fragile and make schema evolution onerous.

In contrast, the Apollo GraphQL framework (Apollo GraphQL, 2023) facilitates data composition through a distributed type system, utilising sub-graphs and super-graphs, which are independent services with their own schemas and resolvers that can interlink and extend across the network. Alternatively, data composability can also be achieved using hyperlinks and centralised type systems like Schema.org (*schema.org*, 2011), popular within the semantic web community, deploying linked data to interconnect related datasets.

Metamycelium draws inspiration from these models, particularly adopting the concept of a distributed type system, moving away from conventional *master dataset management*. Domains within Metamycelium are self-contained, each defining their schemas and lifecycle, which are addressable through a unique URI as part of the metadata API, ensuring schemas are always accessible and current.

> Architectural Principle 12 (AP12): Metamycelium must implement a distributed type system, accessible via a globally unique identifier, maintained to be constantly available and updated.

With well-defined domain boundaries, this system permits schemas such as those for animals within a veterinary application to be uniquely addressable. These URIs enable various domains to reference and extend schemas for their specific requirements.

Furthermore, schema metadata should include temporal aspects like ‘last processing time’ and annotations for deprecated fields. Such a system is illustrated in Figure 6.10.

6.4.13 Immutability and Bitemporality

Immutability means that data once created, doesn’t change. This concept is fundamental to functional programming and is widely discussed in academic literature (Haller &

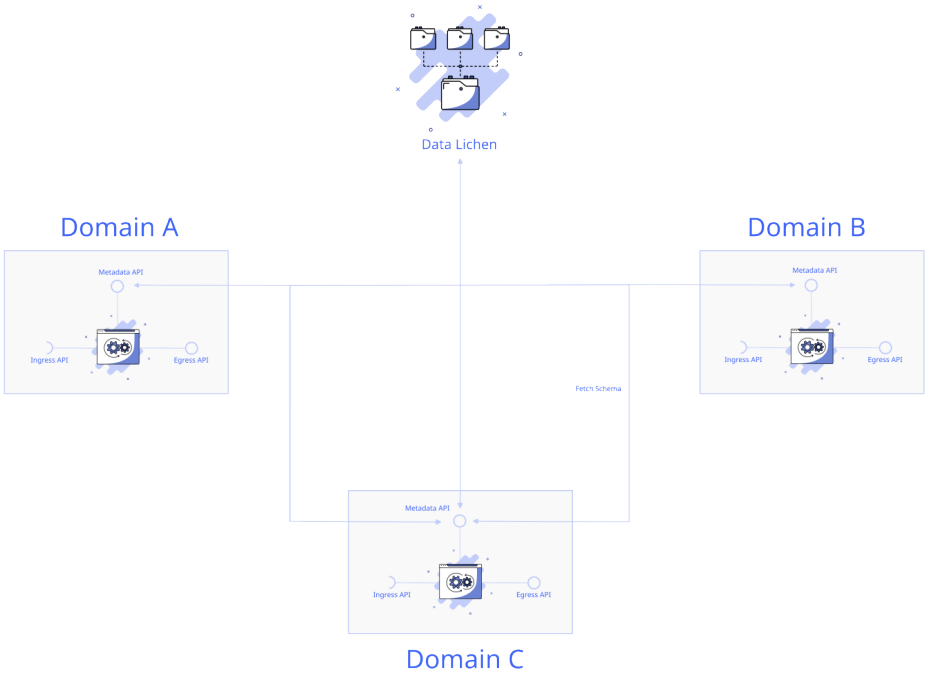


Figure 6.10: The Distributed Type System of Metamycelium

Axelsson, 2017; Coblenz et al., 2016; Ainsworth & Jones, 2020).

According to Bird and Wadler (1988), immutability is a key characteristic of pure functional programming, which requires that functions always produce the same output given the same input. This property is known as referential transparency, and it can only be achieved with immutable data.

While immutability is mostly discussed in relation to software engineering, it is particularly relevant to data engineering and Metamycelium. Applying the principles of immutable data implies that data users can get repeatable analytics query results at a particular point in time.

Repeatability is desired because often data analytics usually results in the emergence of patterns, which in turn results in a deeper analysis of data. Therefore if the data scientist cannot reproduce the same pattern, this may be perceived as a mistake or a

programming error. On the other hand, if there was a bug in the analysis query code, it becomes difficult to track down what went wrong at that point in time.

This issue is exacerbated when it comes to a distributed BD architecture like Metamycelium. If upstream data is used by several downstream data consumers, and given that these downstream data consumers themselves become the provider of data to other downstream data consumers and considering that each domain keeps a slice of that data, the end service or the most downstream service may end up getting two versions of the same data, results in mismatching truth.

Backtracking this data upstream is feasible, but undesirable as this would be a tedious and time-consuming task. Therefore immutability is integral to Metamycelium to provide two important guarantees: 1) once data is processed at a point in time, it will never change and consumers can reliability repeat the data reads, 2) data that has been read is kept consistent across different downstream consumers. Based on this then, the next rule is as follows:

> Architectural Rule 11 (AR11): Analytics data created from the domains for a given point in time, should be immutable. That is, the data that has been produced for a given point in time should never change.

The immutability of data reduces side effects, diminishes chances of incidental complexity, makes debugging and issue finding easier, and increases the overall reliability of the system. This concept is inspired by the principles of functional programming (Bird & Wadler, 1988).

Bitemporality

Another aspect that is integral to data in Metamycelium is data bitemporality. Bitemporal data refers to an approach to modelling data so that events are captured with two timestamps. These two timestamps are referred to by Snodgrass (1999) as the *valid*

date, indicating the actual occurrence of an event or fact, and the *effective date*, denoting the time when the event or fact is considered true.

Bitemporal data modelling enhances the capacity for temporal analysis and prognostication by enabling meticulous examination of historical data patterns and facilitating predictions of forthcoming phenomena. Variability in update frequencies among distinct domains may engender inconsistencies, thereby affecting the integrity of predictive models. Such variability can be effectively neutralised by instituting a uniform processing time across data domains.

To illustrate, a data scientist might standardise the data selection process by extracting records across all domains that have processing times within the delineated interval from 2023-05-01 to 2023-05-02. Crucially, all domains must embed two temporal properties into their datasets: actual time and processing time, as mandated by the following architectural rule:

> Architectural Rule 12 (AR12): All domains should associate two properties to their data: 1) actual time, and 2) processing time.

Operational systems typically concentrate on the current state of data, whereas analytical systems are predicated upon the analysis of historical data to identify patterns and support decision-making processes. Actual time refers to the moment an event occurs, and processing time indicates when an event's data is captured and integrated into the system.

The significance of processing time is derived from the possibility of initial misinterpretations of data. For example, in the animal domain, a recorded count of animals may later be corrected upon identifying an error. The credibility of the animal count for a particular date hinges on the chosen processing time, which enables an accurate analysis based on the corrected data.

The divergence between actual and processing time is a critical aspect of ensuring

accuracy within an analytical framework. Processing time is indicative of the temporal evolution of data, facilitating data immutability and the asynchronous processing between different domains, as outlined in the following architectural rule:

> Architectural Rule 13 (AR13): Processing time shall be adopted as the definitive versioning mechanism for data and metadata.

Additionally, data processing should be scheduled at intervals that are suitable for the context, allowing for data correction, quality enhancement, and minimisation of data retractions. For instance, a financial application that processes a high volume of transactions would necessitate a processing interval that is appropriately timed to match the production and aggregation rate of the transactional data.

6.4.14 Architectural Styles

An architectural style, in the context of software or system architecture, refers to a set of design principles, patterns, and conventions that guide the organisation, structure, and behaviour of a software or system. It provides a high-level blueprint or framework for designing and implementing software or systems, guiding decisions about how different components, modules, and subsystems interact and work together (Richards & Ford, 2020).

Metamycelium is an innovative architectural approach that deviates from centralised BD architectures and embraces the principles of microservices-style architecture. This shift aims to enhance scalability, flexibility, and agility in data-intensive systems. However, the adoption of a microservices-style architecture poses several challenges. First, managing inter-service communication and ensuring data consistency and integrity across distributed services becomes complex. Additionally, handling service discovery, load balancing, and fault tolerance in a dynamic microservices environment requires careful design and implementation.

To address these challenges, an event-driven architecture style can be leveraged. By using events as a means of communication, microservices can operate in a loosely coupled manner, enabling scalability and seamless integration of new services. Furthermore, the event-driven approach facilitates the handling of asynchronous and real-time data processing scenarios.

In the context of Metamycelium, the absorption of the event-driven microservices style aligns with its preference for asynchronous communication without relying on a centralised orchestrator. This architecture empowers Metamycelium to achieve high scalability, resilience, and responsiveness while effectively managing and processing large volumes of data in a distributed environment.

6.4.15 Architectural Characteristics

The architectural characteristics of the Metamycelium architecture are notable for their focus on maintainability, scalability, fault tolerance, elasticity, and deployability. It aligns with modern engineering practices such as automated deployment and continous integration. The architecture emphasises the use of microservices, which are independently deployable and maintainable components.

Maintainability is a high-scoring characteristic of this architecture. The use of event-driven microservices architecture allows for modular development and independent scaling, making it easier to maintain and update individual components without affecting the entire system. Additionally, the architecture supports automated deployment practices, facilitating efficient updates and reducing manual intervention.

Scalability and elasticity are also prominent features. The architecture enables horizontal scalability, allowing for the addition or removal of services based on demand. This flexibility ensures that the system can handle varying workloads effectively.

Fault tolerance is another strength of the architecture. While interservice communication can impact fault tolerance, redundant and scalable design of Metamycelium, along with service discovery mechanisms, mitigate this issue. The independent, single-purpose nature of microservices generally leads to high fault tolerance.

Deployability is emphasised, thanks to the small deployment units and decoupled nature of Metamycelium. The architecture supports evolutionary change, aligning with modern business practices that require agility and adaptability. Small, independently deployable units allow for faster updates and iterations, keeping pace with the dynamic nature of business requirements.

However, the architecture may receive a lower score in terms of cost and simplicity. The distributed nature of Metamycelium and the potential for increased communication overhead can introduce complexities in managing and optimising costs. Strategies such as intelligent data caching and replication can address performance challenges associated with network calls, but cost management remains an ongoing consideration.

Overall, the Metamycelium architecture embraces the strengths of microservices, prioritising maintainability, scalability, fault tolerance, elasticity, and deployability. It acknowledges the challenges inherent in distributed architectures and offers strategies to mitigate them. Architects must understand the rules of architecture to intelligently navigate and leverage its benefits effectively. An overview of architectural characteristics is portrayed in Table 6.2.

The scores for each architectural characteristic are directly adopted from the rating schemes proposed in the book "Fundamentals of Software Architecture" by Mark Richards and Neal Ford (Richards & Ford, 2020), and the book "Software Architecture in Practice" by Rick Kazman et al. (Len Bass, 2021). The rating scale used is as follows:

- ★ - The architectural characteristic exhibits low quality or inadequate support in the Metamycelium architecture.

- ★★ - The architectural characteristic has moderate quality or support in the Metamycelium architecture.
- ★★★ - The architectural characteristic demonstrates good quality or support in the Metamycelium architecture.
- ★★★★ - The architectural characteristic exhibits excellent quality or support in the Metamycelium architecture.

While the rating may not be entirely objective and depends on the researcher's interpretation, it is guided by well-established industry patterns and best practices. For instance, event-driven architectures are commonly highlighted in "Patterns of Enterprise Application Architecture" by Fowler (2012) as promoting high maintainability due to loose coupling and asynchronous communication. Similarly, "Software Architecture: The Hard Parts" by Ford, Richards, Sadalage and Dehghani (2021) discusses how microservices architectures are often rated favorably for scalability due to their modularity and ability to scale individual services independently. The book "Solutions Architect's Handbook" by Shrivastava, Srivastav, Sheth, Karmarkar and Arora (2022) provided insights on evaluating maintainability based on factors like modularity of the architectural constructs.

The evaluation of each characteristic was performed by thoroughly analysing the Metamycelium architecture, its principles, rules, and components, as described in this chapter. The rating for each characteristic was then determined by carefully considering the guidelines, criteria, and examples provided in the aforementioned books, which offer industry-recognised frameworks for evaluating software architecture qualities. Any characteristics that aligned well with the recommended practices and exhibited strong support in the Metamycelium architecture were assigned higher ratings, while those with potential weaknesses or limited support received lower ratings.

Table 6.2: Metamycelium Architecture Characteristics

Characteristic	**Score**
Maintainability	★★★
Scalability	★★★★
Fault Tolerance	★★★
Elasticity	★★★★
Deployability	★★★★
Cost	★★
Simplicity	★
Performance	★★★
Support for Modern Engineering Practices	★★★★

6.5 Artefact

After having discussed many kernel and design theories, the necessary theoretical foundation is created for the design and development of the artefact. Metamycelium is created with Archimate and displays the RA mostly in the technology layer. Displaying these services in the technology layer means that it is up to the architect to decide what flow and application should exist in each node. For the sake of completion, and as every software is designed to account for a business need, a very simple BD business process is assumed. While this business layer could vary in different contexts, Metamycelium should be able to have the elasticity required to account for various business models.

It should be noted that the BD RA does not represent the architecture of any specific BD system. Instead, it serves as a versatile tool for describing, discussing, and developing system-specific architectures using standardised principles. By offering comprehensive and high-level perspectives, Metamycelium facilitates productive discussions regarding the requirements, structures, and operations inherent in BD systems. Notably, it remains vendor-neutral, allowing flexibility in selecting products or services, and does not impose rigid solutions that limit innovation.

6.5.1 Metamycelium

Metamycelium is made up of 15 main components and 5 variable components as depicted in Figure 6.11. The lowercase *a* at the top left corner of the diagram entities stands for *auxiliary view* while the letter *m* stands for *master view*. If the same entity is used in different models, then the auxiliary view is used. This is communicating that this entity already exist and is being reused.

While this business layer could vary in different contexts, Metamycelium should be able to have the elasticity required to account for various business models. To ease understanding of the RA, we sub-diagrammed the product domain in Figure 6.12.

These components are;

1. **Ingress Gateway:** The ingress gateway plays a crucial role in Metamycelium by serving as the primary entry point for all incoming requests from external parties. Adopting the API Gateway pattern discussed in Section 5.3.4, the responsibilities of this component include exposing the necessary port and endpoint to facilitate the flow of data into the system. Depending on the nature of the request, the ingress gateway intelligently load balances the traffic, directing it either to the batch processing controller or the stream processing controller. This load-balancing capability contributes to the architecture's scalability and performance, as requests are efficiently distributed and processed based on their specific requirements.

   Beyond load balancing, the ingress gateway offers several architectural advantages. Firstly, it enhances security by preventing port proliferation and ensuring that services are not directly accessed. Acting as a central entry point, it enables fine-grained control and enforcement of security policies, such as SSL termination, authentication, and potentially name-based virtual hosting. This consolidation of security measures at the ingress gateway safeguards the system from

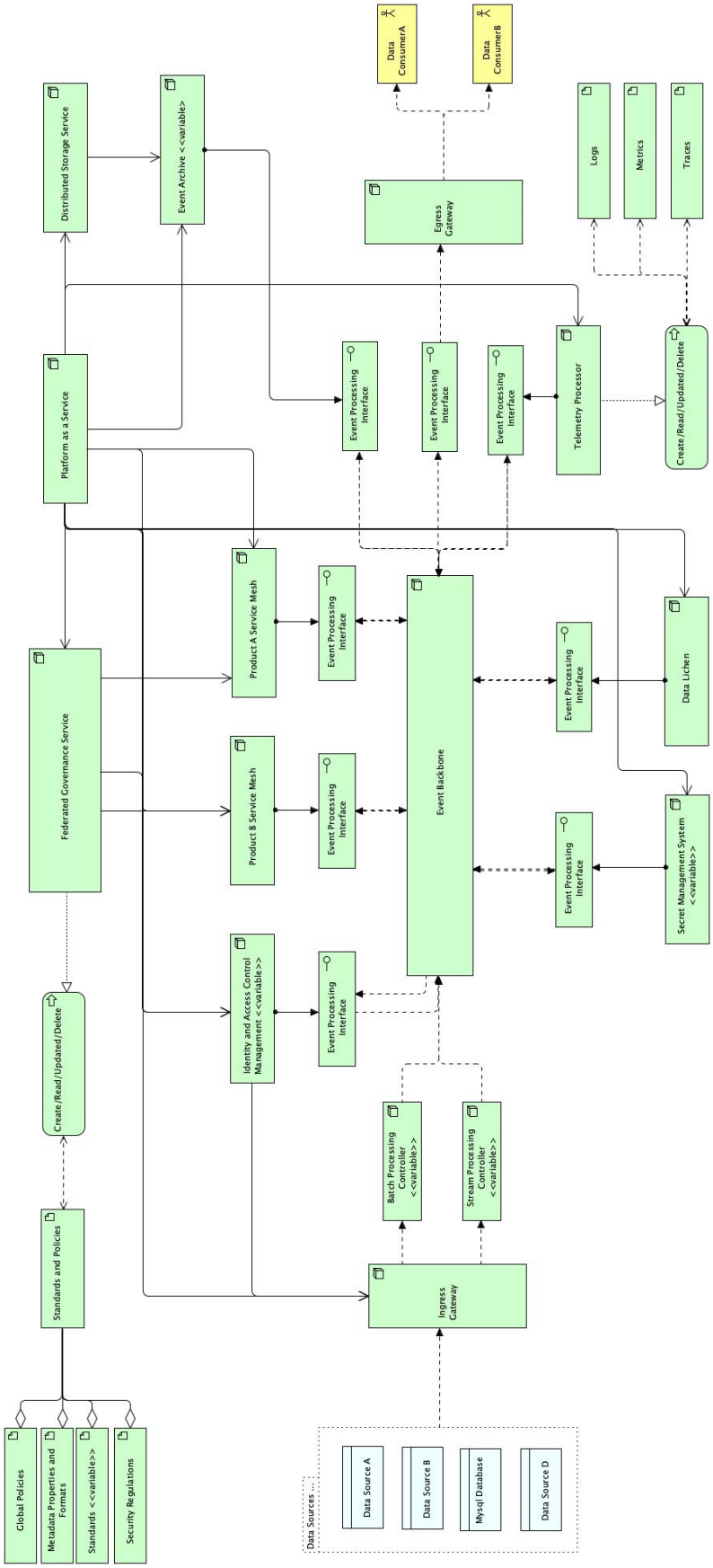


Figure 6.11: Metamycelium

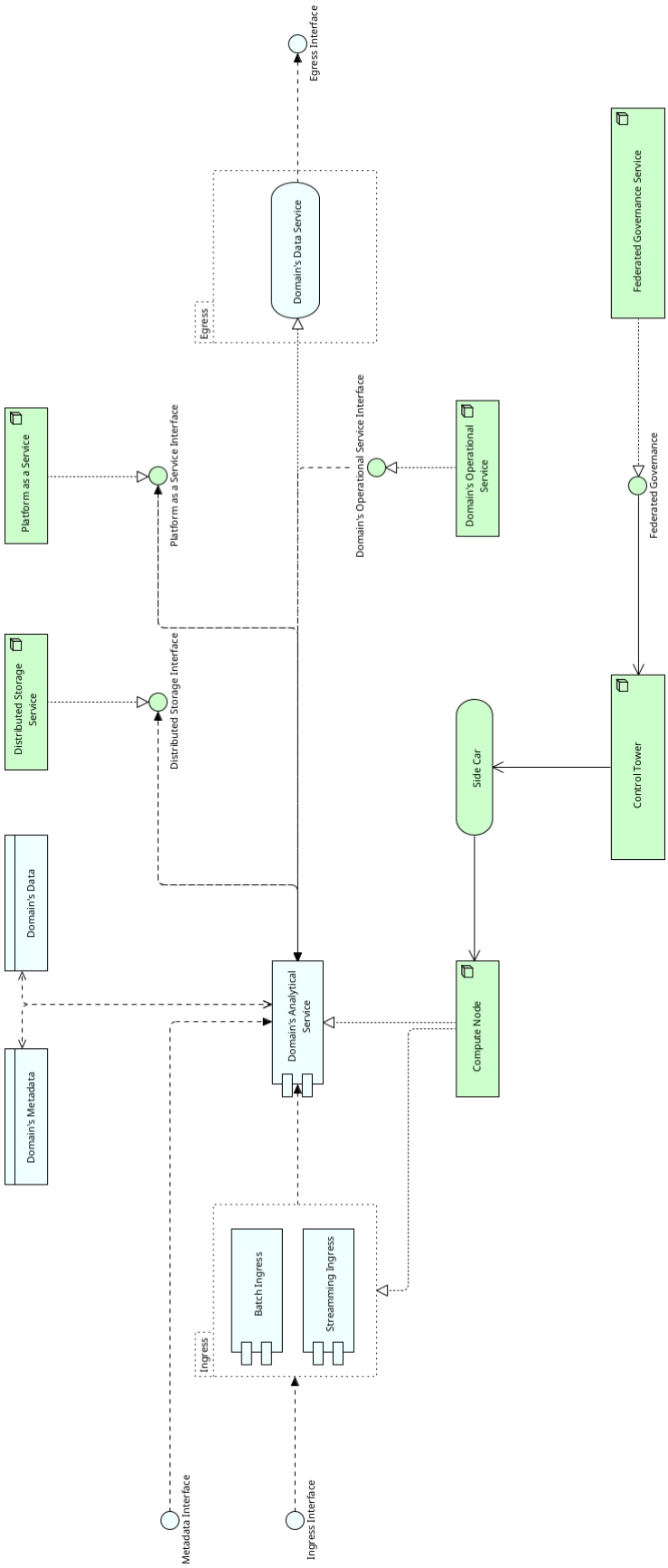


Figure 6.12: Service Mesh

unauthorised access and ensures secure communication with external parties.

Secondly, the ingress gateway improves performance by efficiently handling the SSL termination process, relieving downstream services from this computational burden. Furthermore, it facilitates object mutation through a proxy, allowing for potential modifications or enrichments of incoming data before it reaches the processing controllers. This flexibility enables the architecture to accommodate customisations and integrations with other systems, enhancing its extensibility.

Additionally, the ingress gateway brings benefits in terms of monitoring and observability. With a clear point of entry, monitoring the incoming requests becomes easier, and metrics, logging, and tracing can be focused on this central component. This heightened visibility enables efficient troubleshooting, performance analysis, and compliance monitoring.

Moreover, the ingress gateway supports network segmentation by separating public-facing components from private network resources. It establishes an architectural boundary that isolates internal services and infrastructure, further enhancing security and ensuring better control over access and data flow.

By encompassing all these advantages, the ingress gateway addresses several architectural requirements, such as Vol-1, Vol-2, Var-1, Var-3, Var-4, Val-1, Val-3, Val-4, SaP-1, and SaP-2. Its role in the architecture contributes to its overall robustness, security, scalability, performance, flexibility, and ease of monitoring.

2. **Batch Processing Controller:** Inspired by the BFF pattern discussed in Section 5.3.4, the batch processing controller's main job is to handles batch events by dispatching them to the event backbone. This dedicated service, which can be a small service or a Lambda function, receives batch-processing requests and transforms them into events for the event broker. Its distinct nature allows it to process batch requests in bulk and asynchronously, setting it apart from stream

processing controllers.

The batch-processing controller brings several architectural benefits. Firstly, it significantly enhances monitoring capabilities, providing better visibility into the progress and performance of batch events. This streamlined monitoring facilitates troubleshooting and performance analysis, ensuring that batch processing meets specified requirements and performance expectations.

Additionally, the batch processing controller enables customisation and flexibility in producing batch events. It can perform additional tasks beyond compute-intensive operations, such as data scrubbing or adding custom headers. This customisation capability allows businesses to tailor batch event production to their specific needs and requirements.

Having a specific controller for batch processing acknowledges the unique requirements and characteristics of batch events, providing dedicated functionality and optimisation. This component addresses the requirements Vel-1, Val-1, and Val-2.

3. **Stream Processing Controller:** The stream processing controller plays an important role in the architecture by handling streaming events and dispatching them to the event backbone via the event handling interface. This component is distinguished from the batch processing service as it caters specifically to the unique characteristics of streaming events. Unlike batch processing, streams are synchronous and often require high throughput processing capabilities.

   The stream processing controller, although also a small service, focuses on non-heavy computations that are optimised for stream processing requirements. It can enable stream provenance, which tracks the lineage and history of streaming events, providing valuable insights for data governance and traceability. Additionally, the stream processing controller can leverage one-pass algorithms, which

optimise memory and processing requirements for stream analytics tasks.

One of the key advantages of having a dedicated stream processing controller is the ability to associate custom attributes with stream events. This allows for the contextual enrichment of streaming data and enables differentiated treatment based on the nature of the system. For example, specific rules or transformations can be applied to certain streams, facilitating real-time decision-making or targeted actions based on the content of the events.

The stream processing controller also simplifies monitoring and discovery within the architecture. By having a separate component dedicated to stream processing, it becomes easier to track and analyse the performance, latency, and throughput of streaming events. Additionally, it enables focused monitoring of stream-specific metrics, providing valuable insights into the behaviour and efficiency of the streaming data pipeline.

Overall, the stream processing controller brings architectural value by catering specifically to the unique characteristics of streaming events. Its ability to handle high throughput, apply custom attributes, optimise computations, and simplify monitoring and discovery makes it an important component in architectures involving streaming data. Industry use cases further highlight the importance and applicability of stream processing controllers in various domains. This component is inspired by the BFF pattern discussed in Section 5.3.4 and addresses the requirements Vol-1, Vel-1, Vel-2, Vel-4, Vel-5, and Val-2.

4. **Event Processing Interface:** Event brokers are designed to achieve *inversion of control*. As the company evolves and requirements emerge, the number of nodes or services increases, new regions of operations may be added, and new events might need to be dispatched. As each service has to communicate with the rest through the event backbone, each service will be required to implement

its event-handling module. This can easily turn into a spaghetti of incompatible implementations by various teams, and can even cause bugs and unexpected behaviours.

To overcome this challenge, an event broker is introduced to each service of the architecture. Each service connects to its local event broker and publishes and subscribes to events through that broker. One of the key success criteria of the event broker is a unified interface that sits at the right level of abstraction to account for all services of the architecture. Event brokers, being environmentally agnostic can be deployed to any on-premise, private or public infrastructure. This frees up engineers from having to think about the event interface they have to implement and how it should behave.

Event brokers can also account for more dynamism by learning which events should be routed to which consumer applications. Moreover, event brokers do also implement circuit breaking, which means if the service they have to break to is not available and does not respond for a certain amount of time, the broker establishes unavailability of the service to the rest of the services, so no further requests come through. This is essential to preventing a ripple effect over the whole system if one system fails. This component is driven by the principles of Competing Consumers and Circuit Breaker patterns discussed in Section 5.3.4 and 5.3.4 and indirectly addresses the requirements Val-1, and Ver-1.

5. **Event Backbone:** This is the heart of the Metamycelium, facilitating communication among all the nodes. The event backbone in itself should be distributed and ideally clustered to account for the ever-increasing scale of the system. Communication occurs as choreographed events from services analogous to a dance troupe. In a dance troupe, the members respond to the rhythm of the music by moving according to their specific roles.

Here, each service (dancer) listens and reacts to the event backbone (music) and takes the required action. This means services are only responsible for dispatching events in a *dispatch and forget* model, and subscribe to the topics that are necessary to achieve their ends. Event backbone thus ensures a continuous flow of data among services so that all systems are in the correct state at all times. Event backbone can be used to mix several streams of events, cache events, archive events, and other manipulation of events, so long as it is not too smart! or does not become an ESB of SOA architectures.

Ideally, an architect should perceive the event backbone as a series of coherent nodes that aim to handle various topics of interest. Over time, the event backbone can be monitored for access patterns and can be tuned to facilitate communication in an efficient manner. The pattern CQRS described in Section 5.3.4 has affected the design of this component. This component addresses the requirements Vel-1, Vel-2, Vel-3, Vel-4, Vel-5, Val-1, Val-2, Ver-1, Ver-2, and Ver-3.

6. **Egress Gateway:** This component is inspired by the CQRS pattern described in Section 5.3.4 and offers numerous advantages, especially for external data consumers. In this architecture, data consumers first interact with the discovery component, known as Data Lichen, which serves as a central hub for accessing available data domains. Once connected to Data Lichen, data consumers can navigate to the relevant data domain to retrieve the desired data.

   Furthermore, all external data consumers in the system go through a centralised secret management and centralised authentication and authorization component. This centralised approach brings several benefits to the architecture. Firstly, it ensures a consistent and secure management of secrets, such as API keys, access tokens, or credentials, which are essential for data access and security. This centralised secret management enhances the overall system's security posture by

reducing the chances of vulnerabilities or misconfigurations in secret handling.

Secondly, the centralised authentication and authorization component streamlines the authentication and access control processes for external data consumers. By enforcing a unified authentication mechanism, it ensures that all users are properly authenticated and authorised before accessing the system's data resources. This centralised approach simplifies the management of user access rights, improves security, and provides granular control over data access permissions.

Thirdly, the centralised components simplify the maintenance and scalability of the system. With a single point for managing secrets, authentication, and authorization, it becomes easier to update, monitor, and audit these components. Additionally, this architectural pattern allows for easier scaling and expansion as new data domains or data sources can be seamlessly integrated into the system with consistent authentication and authorization mechanisms.

Overall, the inclusion of an egress gateway in Metamycelium offers a robust and efficient approach for external data consumers. It ensures standardised data access, enhances security, simplifies maintenance, and enables scalability, making it a highly favourable and beneficial architectural design. This component addresses the requirements Vel-2, Vel-4, Val-3, Val-4, SaP-1, and SaP-2.

7. **Product Domain Service Mesh:** In the architectural context, the implementation of a service mesh as a fundamental component of each product's domain is an effective approach. This service mesh comprises various key components that collectively enable efficient data processing, storage, governance, and intercommunication within the domain. These components include batch and streaming ingress, the domain's analytical service, domain's operational service, API access to distributed storage services (such as AWS S3), an infrastructure-providing API for platform-as-a-service modules (e.g., Terraform), containers hosting the

domain's analytical service, a control tower (e.g., Istio), and integration with a federated governance service's API for policy enforcement through sidecars.

The effectiveness of the service mesh stems from its architectural design and its ability to address critical requirements. By encapsulating the domain's capabilities within a service mesh, the coupling between teams is eliminated, allowing for enhanced team autonomy. This architectural approach empowers individuals across teams by granting them the computational resources, tools, and autonomy necessary to operate independently and scale without being negatively affected by other teams or encountering friction with platform teams or siloed data engineering teams.

From an architectural viewpoint, the service mesh's components work in harmony to deliver its benefits. The batch and streaming ingress components facilitate the intake of data into the domain's analytical service, enabling efficient data processing. The analytical service itself leverages domain-specific algorithms and analytical techniques to derive meaningful insights and outputs from the ingested data. API access to distributed storage services provides seamless integration with scalable and reliable storage solutions like AWS S3, ensuring efficient data persistence.

The platform-as-a-service module's API offers infrastructure provisioning capabilities, streamlining the deployment and management of the service mesh. Containers hosting the domain's analytical service provide a standardised and isolated execution environment, promoting modularity, scalability, and ease of deployment. The control tower, represented by popular frameworks like Istio, manages and orchestrates the communication and traffic flow within the service mesh, enabling features such as load balancing, routing, and policy enforcement.

Additionally, the service mesh integrates with a federated governance service,

using its API to retrieve and enforce policies through sidecars. This ensures adherence to governance and compliance requirements, supporting centralised control and management across the service mesh.

Moreover, the service mesh architecture includes a noteworthy element: the analytical service's access to operational data in a native and accessible manner. This aspect bridges the gap between data analytics and operational systems, effectively removing the *great divide of data* and bringing analytics closer to the source.

By enabling direct access to operational data within the service mesh, the analytical service gains real-time insights and a holistic view of the system's operations. This seamless integration of operational data eliminates the need for manual data extraction and transformation processes, reducing latency and enabling timely decision-making.

Moreover, having native access to operational data enhances the analytical service's effectiveness and accuracy. It eliminates potential data discrepancies or inconsistencies that may arise when analytics are performed on ETLed data. By accessing the operational data directly, the analytical service can provide up-to-date and reliable insights, contributing to more accurate analysis and informed decision-making.

This native and accessible access to operational data within the service mesh brings analytics closer to the source. It promotes a data-driven culture by empowering the analytical service to work in tandem with operational systems, enabling a feedback loop where insights from analytics can drive optimisations and improvements in real time. This closer integration fosters a more agile and responsive approach to decision-making, leading to enhanced operational efficiency, innovation, and business value. This also eliminates accidental data quality issues

that can arise from upstream operational systems.

By eliminating the divide between analytics and operational data, the service mesh architecture supports a unified view of the system, facilitating seamless collaboration between analytics and operations teams. This convergence promotes a holistic understanding of the business and operational dynamics, enabling data-driven insights to be readily incorporated into the operational workflows.

The service mesh's effectiveness lies in its ability to address key architectural concerns. It promotes scalability, allowing the domain to handle large volumes of data and increasing computational resources as needed (Vol-1). It facilitates rapid development and deployment of analytical capabilities (Vel-3, Vel-4, Vel-5). The service mesh architecture accommodates variability in business contexts, supporting the diverse needs and requirements of different product domains (Var-1, Var-2, Var-3). It ensures data validation, quality, and integrity by leveraging advanced analytics and processing techniques (Val-1, Val-2, Val-3, Val-4). Security and privacy requirements are fulfilled through policy enforcement, secure communication, and data governance mechanisms (Sap-1, SaP-2). Finally, the service mesh architecture allows for the verification of system behaviour, enabling efficient testing, monitoring, and verification of the domain's analytical outputs (Ver-1, Ver-2, Ver-3). This component design is affected by the patterns CQRS (Section 5.3.4), Anti-Corruption Layer (Section 5.3.4), and Pipes and Filters (Section 5.3.4).

8. **Federated Governance Service:** Evidently, Metamycelium is a distributed architecture that encompasses a variety of independent services with an independent lifecycle that are built and deployed by independent teams. Whereas teams have their autonomy established, in order to avoid haphazard, out-of-control and conflicting interfaces, there should be a global federated governance that aims to

standardise these services. This will facilitate the interoperability between services, communication, and aggregates, and even allows for a smoother exchange of members across teams. This also means the most experienced people at a company such as technical leads and lead architects will prevent potential pitfalls that more novice engineers may fall into. However, the aim of this service is not to centralise control in any way, as that would be going a step backwards into the data warehouse era.

This service aims to allow autonomous flow in the river of standards and policies that tend to protect the company from external harm. For instance, failing to comply with GDPR while operating in Europe can set forth fines of up to 10 million euros, and this may not be something that novice data engineers or application developers are fully aware of (Voigt & Von dem Bussche, 2017). The real challenge of the governance team is then to figure out the necessary abstraction of the standards to the governance layer and the level of autonomy given to the teams. The federated governance service is made up of various components such as global policies, metadata elements and formats, standards and security regulations. These components are briefly discussed below;

(a) **Global Policies:** general policy that governs the organisational practice. This could be influenced by internal and external factors. For instance, complying with GDPR could be a company's policy and should be governed through the federated governance service.

(b) **Metadata Properties and Formats:** This is an overarching metadata standard defining the required elements that should be captured as metadata by any service within the organisation; it can also include the shape of metadata and its properties of it. For instance, the governance team may decide that each geographic metadata should conform to ISO 19115-1 (ISO, 2019).

(c) **Standards:** overall standards for APIs (for instance Open API), versioning (for instance SemVer), interpolation, documentation (for instance Swagger), data formats, languages supported, tools supported, technologies that are accepted and others.

(d) **Security Regulations:** company-wide regulations on what's considered secured, what software is allowed, how interfaces should be conducted and how the data should be secured. For instance, the company may choose to alleviate risks associated with OWASP's top 10 application security risks.

While the above-mentioned components are promoted as a bare minimum, an architect may decide to omit or add a few more components to the federated governance service. This component can indirectly affect all requirements.

9. **Data Lichen:** As the number of products increases, more data become available to be served to consumers, interoperability increases, and maintenance becomes more challenging. If then, there is no automatic way for various teams to have access to the data they desire, a rather coupled and slow BD culture will evolve. To avoid these challenges and to increase discoverability, collaboration, and guided navigation, the Data Lichen should be implemented. Data discovery mechanisms like Data lichen are listed as a must-have by Gartner (Ehtisham Zaidi, 2019) and introduce better communication dynamics, easier data serve by services and intelligent collaboration between services. This component addresses the requirements Vel-4, Var-1, Var-3, and Var-4. This component is portrayed in Figure 6.13.

10. **Telemetry Processor:** If all services employ the idea of localised logging, and simply generate and store logs in their own respective environments, debugging, issue finding and maintenance can become a challenging task. This is due to the

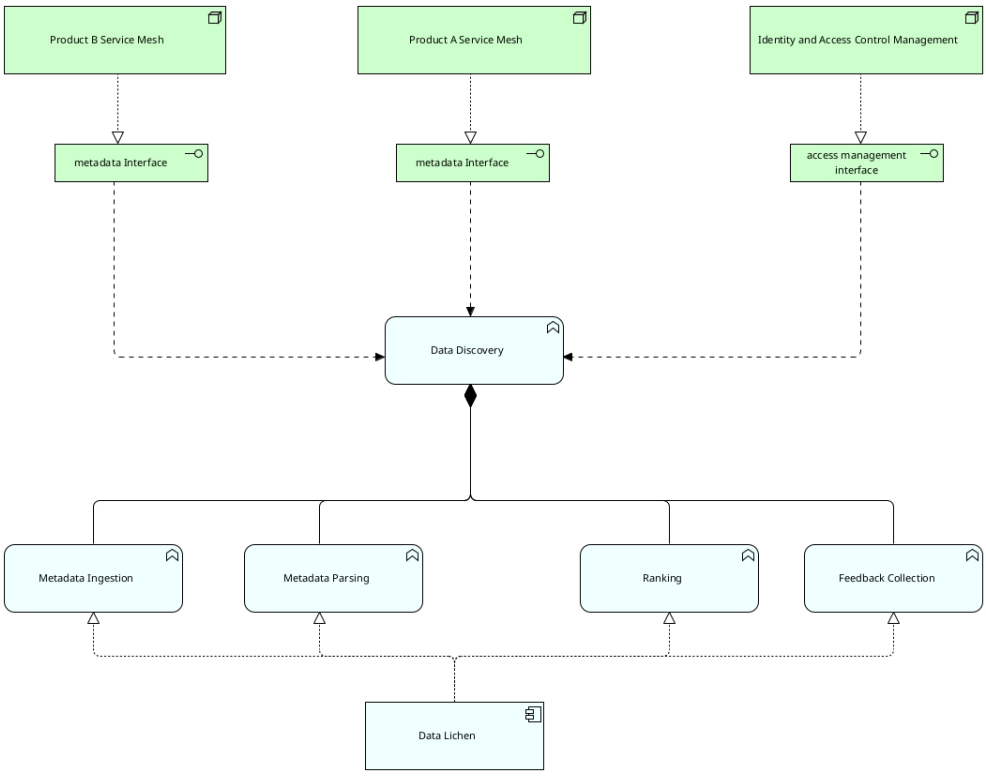


Figure 6.13: Data Lichen

distributed nature of Metamycelium and the requirements to trace transactions among several services. In order to overcome this challenge, the log aggregation pattern discussed in Section 5.3.4 is used.

The centralised service for processing telemetry data in Metamycelium offers several architectural benefits. By routing all telemetry data through a centralised service, it becomes easier to aggregate, process, and analyse logs, metrics, and traces from various sources. This centralisation simplifies the data processing pipeline and reduces the complexity of handling telemetry data across the architecture.

Moreover, the centralised service provides a unified view of logs, metrics, and traces, enabling comprehensive analysis and correlation of data across the entire system. This allows for holistic monitoring, troubleshooting, and performance

optimisation by leveraging data from different sources and identifying patterns or anomalies that might not be evident when considering individual services or components in isolation.

In terms of scalability and performance, centralising the processing of telemetry data enables efficient resource allocation. The architecture can scale the centralised service horizontally or vertically based on the growing telemetry data volume, ensuring optimal performance and responsiveness.

Having a centralised service for telemetry data processing ensures consistent monitoring and governance practices across the entire architecture. It allows for the enforcement of standardised monitoring, alerting, and governance policies, ensuring that all microservices or components adhere to the defined guidelines. This consistency simplifies management, improves system-wide visibility, and promotes adherence to best practices.

In addition to its role in processing telemetry data, the centralised service in the architecture can serves as an important data source for various consumers, including the Data Lichen. Data Lichen leverages the processed telemetry data to generate valuable insights and support data discovery. However, the data processed within this centralised service is not limited to a single consumer. It can also be made available for consumption by custom systems and dashboards, providing flexibility and extensibility to meet specific business requirements.

By enabling the consumption of telemetry data by multiple systems and dashboards, the architecture fosters a data-driven ecosystem, empowering different stakeholders to extract meaningful information and gain insights from the processed telemetry data. This versatility and accessibility of the telemetry data further enhance the value proposition of the centralised service within the broader BD architecture. This component indirectly addresses the requirements Vol-1,

Vel-1, Val-1, and Ver-1. A high-level overview of this component is shown in Figure 6.14.

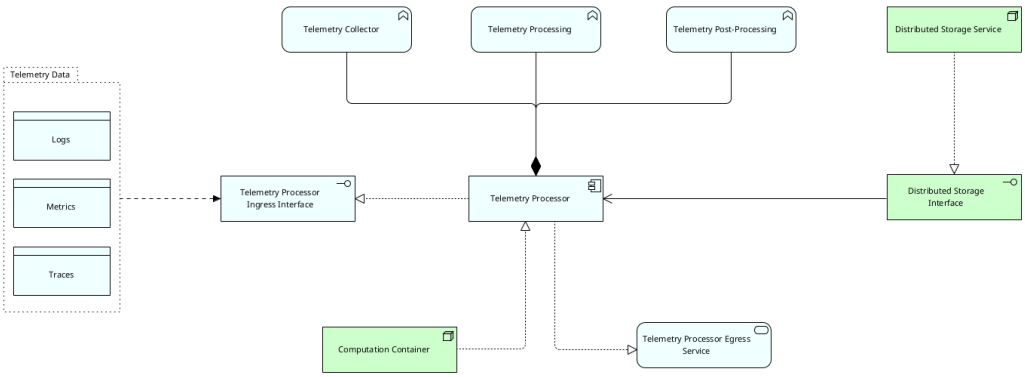


Figure 6.14: Telemetry Processor

11. **Event Archive:** As the number of services grows, the topics in the event backbone increase, and the number of events surges. Along the lines of these events, there could be a failure, resulting in a timeout and a loss of a series of events. This brings the system in the wrong state and can have a detrimental ripple effect on all services. Metamycelium tends to handle these failures by using an event archive. The event archive as the name states, is responsible for registering events, so they can be retrieved in the time of failure.

    If there is a blackout in a certain geographical location and the event backbone is down, the backbone can recover itself and bring back the right state of the system by reading the events from the event archive. The event interface is responsible for circuit breaking, so services do not request any more events to the backbone while it is down. The time to expiry, and what events should be archived are decided based on the context in which Metamycelium is implemented. This component indirectly addresses the requirements Vol-1, Vel-1, Val-1, and Ver-1.

12. **Distributed Storage Service:** Whereas Metamycelium is a great advocate of decentralised and distributed systems, it is not necessary for each product domain

to have its own kind of data storage. This is to prevent duplication, contrasting data storage approaches, decreased operability among services and lack of unified data storage mechanisms. The distributed storage service has been designed to store large volumes of data in raw format before it can get accessed for analytics and other purposes.

This means data can be first stored in the distributed storage service with corresponding domain ownership before it needs to be accessed and consumed by various services. Structured, semi-structured, unstructured and pseudo-structured data can be stored in the distributed storage service before it gets retrieved for batch and stream processing. Nevertheless, this does not imply that all data should directly go to the this service; the flow of data is determined based on the particularities of the context in which the system is embodied. This component addresses the requirements Vol-2, Vel-1, Var-1, Var-3, Var-4, Val-3.

13. **Platform as a Service:** The platform as a service (PaaS) component serves as a central hub that offers an API to all other components in the system. This PaaS component plays a crucial role in enabling the autonomy and scalability of the overall infrastructure.

    One of the key architectural values of this PaaS component is its ability to abstract the underlying infrastructure complexities. By providing a standardised API, it allows each component to independently manage and provision the required resources, such as compute, storage, and networking, without being burdened by the intricate details of the underlying infrastructure. This abstraction layer promotes loose coupling between components and facilitates easier development, deployment, and maintenance of the system as a whole.

    Another architectural value of the PaaS component is its emphasis on scalability and elasticity. It enables dynamic allocation and de-allocation of resources based

on the varying demands of different data domains. By offering an API that allows components to request resources as needed, the PaaS component enables efficient utilisation of the infrastructure. It can dynamically scale resources up or down, ensuring optimal performance and cost-effectiveness across the entire BD architecture.

Each data domain can make requests to the PaaS API to provision, configure, and manage the necessary resources, enabling them to operate independently and efficiently. This decentralisation of infrastructure management enhances agility and flexibility within the architecture. This component addresses SaP-1, SaP-2, Var-1, Var-3, Var-4, Vel-1, and Vol-2.

14. **Identity and Access Management:** The Identity and Access Management (IAM) component role is in ensuring secure and controlled access to the system's resources and data. It encompasses various architectural values that are essential for maintaining data integrity, privacy, and regulatory compliance.

    One of the key architectural values of the IAM component is its focus on authentication and authorization. It provides robust mechanisms to authenticate users, components, and services within the architecture, ensuring that only authorised entities can access the resources and perform specific actions. These mechanisms help prevent unauthorised access, mitigate security risks, and safeguard sensitive data.

    Another architectural value of the IAM component is its emphasis on centralised user and access management. It serves as a centralised authority for managing user identities, roles, and permissions across Metamycelium. This centralisation streamlines the administration of access controls, simplifies user onboarding and offboarding processes, and ensures consistent enforcement of security policies throughout the system.

The IAM component ensures detailed access control and privilege management, allowing for the establishment of specific access policies. It supports robust authentication via standard protocols like OAuth and SAML, streamlining user access with SSO. Additionally, it underpins auditing by logging access events, thereby bolstering compliance, security oversight, and incident management. This component addresses the requirements SaP-1, and SaP-2. A comprehensive depiction of the component's functionality and its interaction with other system parts is illustrated in Figure 6.15.

15. **Secret Management System:** The central secret management system serves as an important component for securely storing and managing sensitive information such as passwords, API keys, cryptographic keys, and other secrets.

    One of the key architectural values of the central secret management system is its focus on the secure storage and encryption of secrets. It employs robust encryption algorithms and mechanisms to protect sensitive data at rest, ensuring that secrets are securely stored and inaccessible to unauthorised entities. This value helps prevent unauthorised access to secrets, mitigates the risk of data breaches, and ensures confidentiality.

    Additionally, the secret management system supports the secure distribution and retrieval of secrets to authorised components or services. It provides mechanisms such as secure APIs or client libraries that enable secure retrieval of secrets during runtime. This value ensures that sensitive information is only accessible to the authorised entities that require them, preventing unauthorised exposure of secrets.

    Furthermore, the secret management system promotes integration with other components and services within Metamycelium. It provides APIs or integration points that allow seamless integration of secrets into various data domains. This integration value facilitates secure and convenient access to secrets for authorised

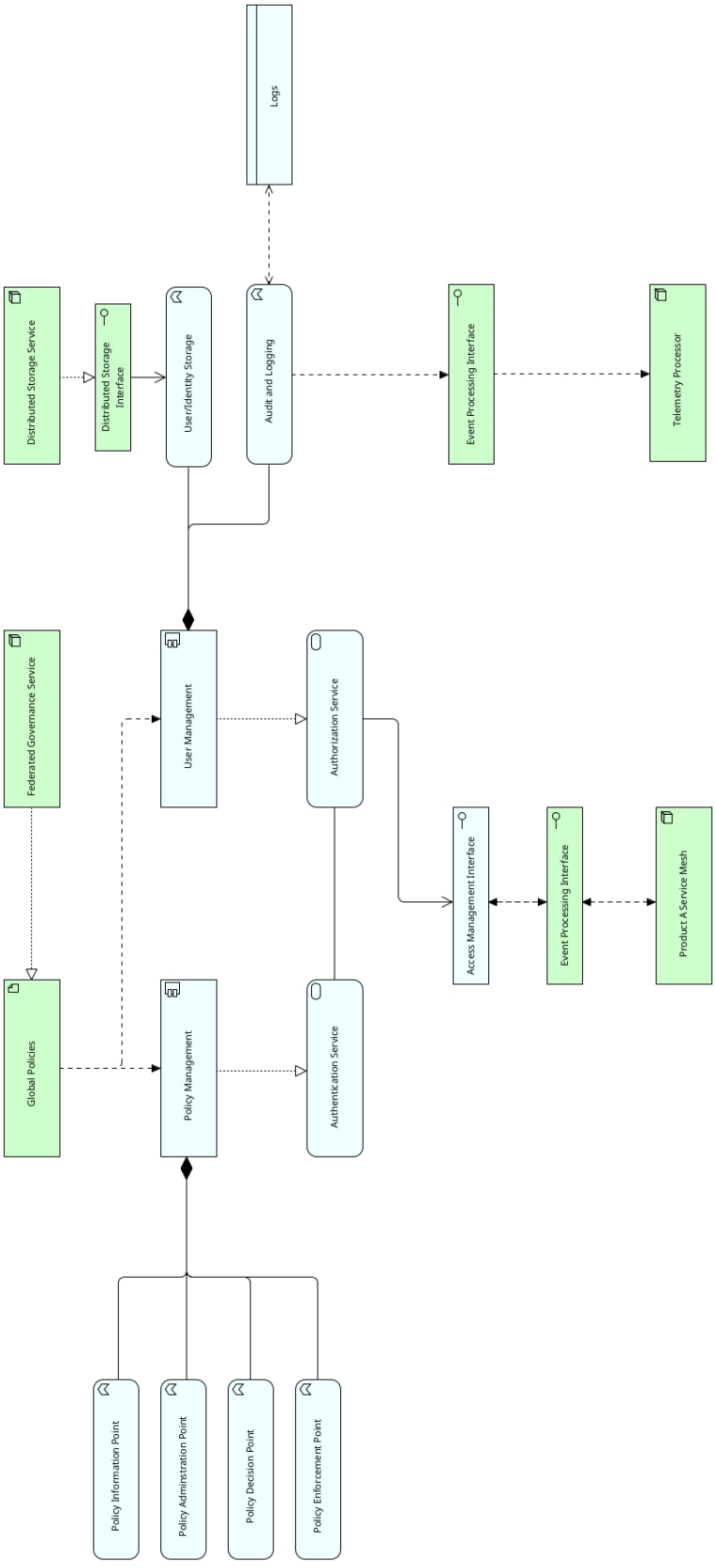


Figure 6.15: Identity and Access Control Management System

components, ensuring smooth operation and reducing friction in the development and deployment processes. This component is indirectly affected by the Externalised Configuration Store pattern discussed in Section 5.3.4 and addresses the requirements SaP-1 and SaP-2.

The variable elements in Metamycelium can be adjusted, modified and even omitted based on the architect's decision and the particularities of the context. The aim of this RA is not to limit the creativity of data architects but to facilitate their decision-making process, through the introduction of well-known patterns and best practices from different schools of thought. All alternative options for each variable module are not elaborated as the industry constantly changes, and architects constantly aim to design systems that address the emerging problem domains.

For instance, an architect may choose to omit IAM from the implementation as the company is not yet ready to invest in such system. The architect may choose to implement authentication per service as the company has not yet scaled to be fully domain-driven.

6.6 Conclusion

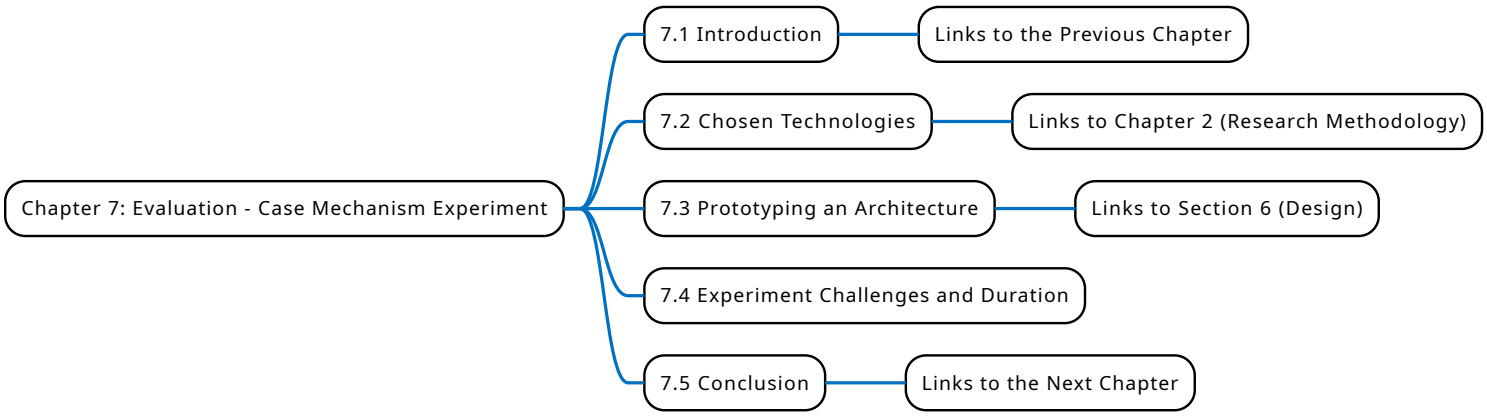
The chapter defines the requirements of Metamycelium based on big data characteristics such as volume, velocity, variety, value, security, privacy, and veracity. It also discusses the underlying theories of Metamycelium, including the limitations of monolithic architectures, the need for domain-driven decentralisation, treating data as a first-class citizen, and the importance of federated computational governance. Additionally, the chapter presents architectural styles, principles, and rules that guide the design of Metamycelium, such as adopting microservices architecture, promoting local autonomy, and adhering to data quality metrics. Finally, the artefact, Metamycelium, is delineated

with its 15 main components and 5 variable components, each serving specific roles in the overall architecture.

The next chapter will focus on evaluating the Metamycelium artefact through a case mechanism experiment. This evaluation will assess how well the artefact addresses the defined requirements based on big data characteristics.

Chapter 7

Evaluation - Case Mechanism Experiment



7.1 Introduction

After presenting the design of the artefact in the previous chapter, the thesis now transitions into the evaluation phase, which is essential for assessing the artefact's effectiveness and validity. It should be recognised that the validation of treatment plays a critical role in its research, development, and implementation process. The validation

of a treatment fundamentally ascertains its potential contribution to stakeholder goals when implemented within the problem context.

The basic idea is that if the requirements for the treatment are carefully outlined and justified, it creates a clear pathway for validation. This means the treatment can be validated by demonstrating that it meets these established requirements. However, a central dilemma encountered in treatment validation is the lack of a real-world implementation in which the contribution of the treatment to stakeholder goals can be examined. This obstacle highlights the necessity of robust, multi-faceted, and carefully considered evaluation strategies, which is elaborated upon in the ensuing sections of this chapter.

The first part of the evaluation of this thesis is a *single-case mechanism experiment* following the guidelines of Wieringa (2014). While in his works, there's a mention of a *single* experiment, for the purposes of this study, a few cases are considered. The necessity for a multi-case experiment arises due to the multifaceted nature of the artefact designed, which enhances the rigour and relevance of the evaluation method, thereby ensuring a more comprehensive and nuanced understanding of the artefact's functioning and potential.

The research design for this case mechanism experiment is detailed in Section 2.9.5. The present section encompasses the actual prototyping of the artefact and the subsequent presentation of results. This chapter is structured as follows: Section 7.2 presents the chosen technologies for each component of Metamycelium based on the technology selection research conducted in Chapter 2. Section 7.3 details the prototyping process, which involves the development of a functional representation of the architecture. This section is further divided into three stages: Formation (Section 7.3.1), Amalgamation (Section 7.3.2), and Scenario Testing (Section 7.3.3).

Finally, Section 7.4 discusses the challenges encountered during the experimentation phase, categorised into Infrastructure, Integration, Data Management, and Performance.

7.2 Chosen Technologies

The technologies chosen for each component of Metamycelium are based on the technology selection research conducted in Section 2.9.5 of the Chapter 2. The evaluation matrix presented in Table 7.1 summarises the results of the technology selection process.

Table 7.1: Evaluation Matrix

Technology	Category	Func. Suit.	Maint.	Comp.	Port.
Apache Kafka	DI	4.5	3	5	4
Apache Flume	DI	3.5	4	2.5	4
Apache Storm	DI	3.5	3	3	3
Hadoop HDFS	DS	4.5	3.5	2.5	3
Apache Cassandra	DS	5	3	4	4
Hadoop HBase	DS	4	3	2	2
Apache Spark	DP	5	4	5	4
Apache Flink	DP	4	4	4	4
FastAPI	M	4	4	4	3.5
ExpressJS	M	3.5	4	4	4
Kafka Rest Proxy	M	4	3	3	3
Apache Hive	DA	4	3.5	4	3
Apache Pig	DA	3	3	2	2
Laravel	M	3	4.5	4	3.5
.NetCore	M	3	4	4	4.5
Tableau	DV	4	4	4.5	2.5
QlikView	DV	3	3	2	3
Elasticsearch	DS	4	4	3	4
Apache ZooKeeper	M	5	3	5	4

Evaluation Matrix

Table 7.1: Evaluation Matrix

Amazon Redshift	DS	4	3.5	3.5	4
Amazon EMR	DP	3	2	2	4
Amazon Athena	DA	3	3	4	4
Amazon DynamoDB	DS	4	4.5	3.5	5
Google BigQuery	DS	3	4.5	3	4.5
Google Cloud Dataflow	DP	3.5	4.5	4	5
Azure Synapse Analytics	DA	4	4.5	3	3
Azure HDInsight	DP	3	4	4	4
Azure Databricks	DP	5	4	5	4.5
AWS Glue	DI	3.5	2	2	4
Amazon Aurora	DS	4	4.5	4	2.5
Apache Oozie	DP	3	3	3.5	2.5
Elastic Kibana	DV	3	3	3	2.5
Amazon Kinesis	DI	5	4	3	4.5
Google Cloud Pub/Sub	DI	3.5	3	3	3
Google Cloud Spanner	DS	4.5	4	3.5	5
Azure Stream Analytics	DP	4	4.5	4.5	3
Azure Cosmos DB	DS	3.5	4	4	4
Google Cloud Bigtable	DS	3.5	4	3	3.5
Microsoft Power BI	DV	4.5	4	4	2.5
Qlik Sense	DV	3.5	3	2.5	3
MongoDB	DS	4	3.5	3.5	4
Neo4j	DS	3.5	4	3	4
Amazon S3	DS	5	4.5	4.5	5

Evaluation Matrix

Table 7.1: Evaluation Matrix

AWS Lambda	DP	4	3.5	4.5	5
Google Cloud Storage	DS	4.5	4	4.5	4.5
Azure Blob Storage	DS	4	3.5	4	4
Azure Table Storage	DS	3.5	4	4	3.5
AWS Step Functions	DP	3.5	3	3	4
Azure Functions	DP	4	4	4.5	4.5
Google Cloud Functions	DP	3.5	3.5	4	4

Note: DI = Data Ingestion; DS = Data Storage; DP = Data Processing; DA = Data Analytics; DV = Data Visualisation; M = Miscellaneous; Func. Suit. = Functional Suitability; Maint. = Maintainability; Comp. = Compatibility; Port. = Portability

Based on the result of the evaluation matrix, technologies that provide the functionalities needed for Metamycelium components are chosen. The preference is for open-source technologies. The chosen technologies for each component are displayed in Table 7.2.

Table 7.2: Map of Metamycelium Component to Technologies

Metamycelium Component	**Dependency**
Ingress Gateway	Kubernetes Ingress
Event backbone	Kafka
Event processing interface	Kafka Rest Proxy
Identity and Access Control Management	Keyclock
Product Service Mesh	Istio
Data Lichen	Node JS
Secret Management System	Vault
Federated Governance Service	Open Policy Agent
Platform as a Service	Helm and Terraform
Distributed Data Storage	MinIO
Telemetry Processing Service	FastAPI
Domain Analytical Service	FastAPI
Domain Operational Service	FastAPI
Data Scientist External Service	FastAPI

7.3 Prototyping an Architecture

Prototyping involves two distinct aspects: first, the development of a concrete architecture that describes how the reference architecture can be applied to a specific context, and second, the instantiation of this concrete architecture into an implemented system. The concrete architecture serves as a blueprint that can support multiple system variants, versions, and deployments. Its effectiveness is measured by how well it guides the development, deployment, and operation of these various instantiations over time. This is often called the *concrete architecture* (Wieringa, 2014). This process allows for the validation and refinement of the architecture, enabling researchers to assess its feasibility, performance, and alignment with the research objectives.

Prototyping the architecture involves implementing the key components and functionalities outlined in the RA, Metamycelium. The chosen technologies, which are academically justified and aligned with the research objectives, form the foundation for building the prototype. Through the systematic implementation of the architectural components, the architecture's effectiveness in handling large-scale data processing, addressing volume, velocity, variety, value, security, privacy, and veracity requirements can be assessed.

This prototyping consists of three major phases: formation, amalgamation, and scenario testing. The formation phase focuses on creating the foundational services and components of the architecture using the chosen technologies. Next, the amalgamation phase integrates these services and components, establishing connections and communication channels to ensure they work together as a cohesive system. Finally, the scenario testing phase puts the architecture's capabilities to the test by executing a set of scenarios that simulate real-world use cases and edge cases, covering various aspects such as data ingestion, processing, query execution, and security. The results of these scenarios are then analysed and presented to evaluate the architecture's performance,

scalability, and alignment with the research objectives. These phases are portrayed in Figure 7.1.

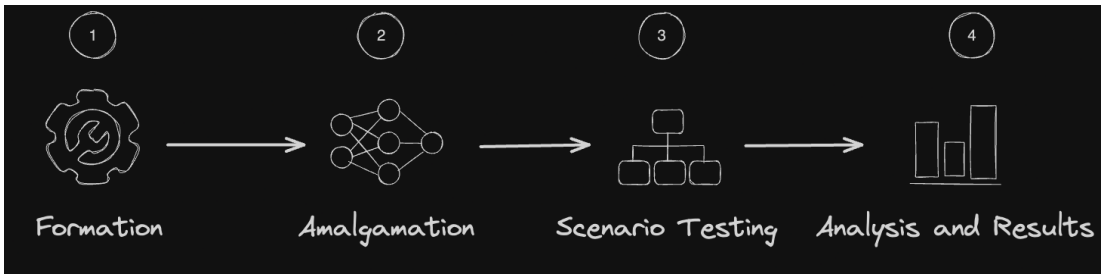


Figure 7.1: Metamycelium Prototyping Phases

A concrete architecture designed from Metamycelium is displayed in Figure 7.2.

7.3.1 Stage 1: Formation

All of the components of this prototype are embodied in one Kubernetes cluster. Thus, for this prototype to work, KIND (Kubernetes Special Interest Group, 2023) has been chosen as the local Kubernetes cluster.

Of particular importance, is the repeatability of this process. For this purpose, all the scripts necessary to run, download, and configure artefacts are stored in the scripts folder in the Github repo at Polyhistor (2023b). It is important to note that these scripts are written specifically to run on Unix-like operating systems.

After having set up the basic scripts for bringing up the cluster, the process began by creating different services inside the Kubernetes cluster. For Nginx ingress and Kafka, Helm Charts on Artifact Hub (Team, 2023) are used for increased maintainability and scalibility. Terraform was then used to apply the charts on the Kubernetes cluster. This can be found in IaaC folder of the repository.

After conducting a thorough search for Helm charts for all components of the artefact, we chose a mature and well-regarded chart from Bitnami. This approach is

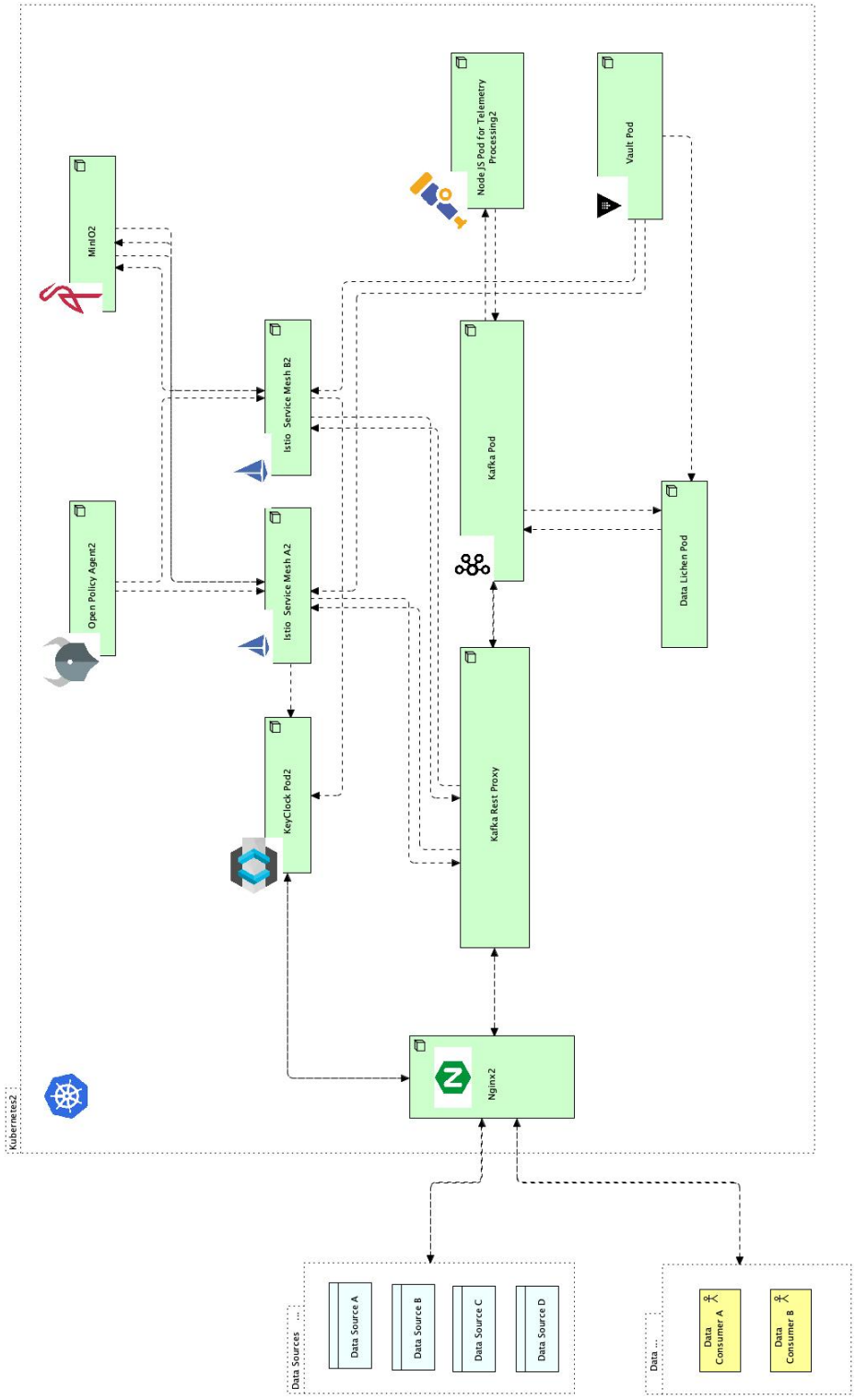


Figure 7.2: Metamycelium Prototype

preferred over creating special directives for Nginx ingress in Kubernetes and mapping services to each other using Kubernetes architectural constructs like Services or StatefulSets.

After the ingress, the search continued on Artifact Hub for charts for other components of the concrete architecture. After having found a suitable chart for Nginx ingress, Keyclock, and Vault, the telemetry processor is created.

After that, the FastAPI application is packaged into a special format called a Docker image. This image contains everything the application needs to run, including its code and any necessary libraries. Onwards, this image is served on Github pages and then used in Terraform helm release resources to be applied to the local KIND cluster. From there on, the Kubernetes ingress controller is set as Nginx and applied to local kind cluster using Terraform.

After having the control setup, the ingress resources are made to point the traffic to Kafka-rest-proxy. Kafka-rest-proxy is added to the artefact to ease the process of accessing data from the cluster. This is due to the fact that Kafka uses binary protocol as oppose to HTTP while most agents today rely on HTTP for network communication.

After having the Nginx ingress correctly configured, and the kafka rest proxy, Keycloak is installed and integrated into different elements of the architecture. From there on, Vault is added to the cluster using Hashicorp's official helm chart (HashiCorp, 2023b). Vault UI is activated but with challenges discussed in Section 7.4.

After this, the next big system to implement is Data Lichen. Since Data Lichen needed a newly built frontend from scratch, embdedded Javascript which is a template engine for Node.js is chosen. This template engine was chosen as it eases the development and accelerates the processes of this experiment.

The objective is not to develop a comprehensive frontend for Data Lichen, but rather to construct a straightforward user interface that effectively demonstrates the potential appearance and functionality of this architectural framework. Different teams may

choose to implement Data Lichen in different ways. Data Lichen needs to be coded from scratch as there's no open source technology that provides the functionalities required.

After having Data Lichen sorted and deployed to the cluster, Istio service meshes are created. For this purpose different Kubernetes namespace are created and different services are grouped under the same namespace for service mesh to govern and operate. Additionally, for a better development experience and observability, Istio's dashboard, Kiali, is installed.

Next, Minio chart is installed, and the dashboard is activated. From there on, the final piece that had to be installed was Open Policy Agent (OPA). There are several approaches to integrating OPA into an architecture. This could be done through standalone Envoy proxy, Kubernetes operator, a load balancer through Gloo Ede and finally through Istio.

Istio is chosen as it is already selected and deployed to the cluster. However, the process of automatically injecting policies into services running in a specific service mesh is not a trivial task. An approach taken for this experiment is the extension of the Envoy sidecar with an external authoriser API to allow for inbound and outbound policies.

This requires a deep knowledge of Envoy, Kubernetes, Istio, Rego (the policy language used by OPA), and GoLang. OPA is integrated into the service mesh as a separate service (also known as sidecar proxy). The OPA acts as an admission controller, reviewing requests made to the Kubernetes API server.

OPA evaluates these requests against set policy rules. If a request is approved, it is processed normally by the API server; if denied, an error response is returned to the user. In terms of services and pods in a namespace, all interactions are also regulated by OPA's policies. In essence, OPA helps maintain compliance with predefined security and operational rules in a Kubernetes environment. This is illustrated in Figure 7.3.

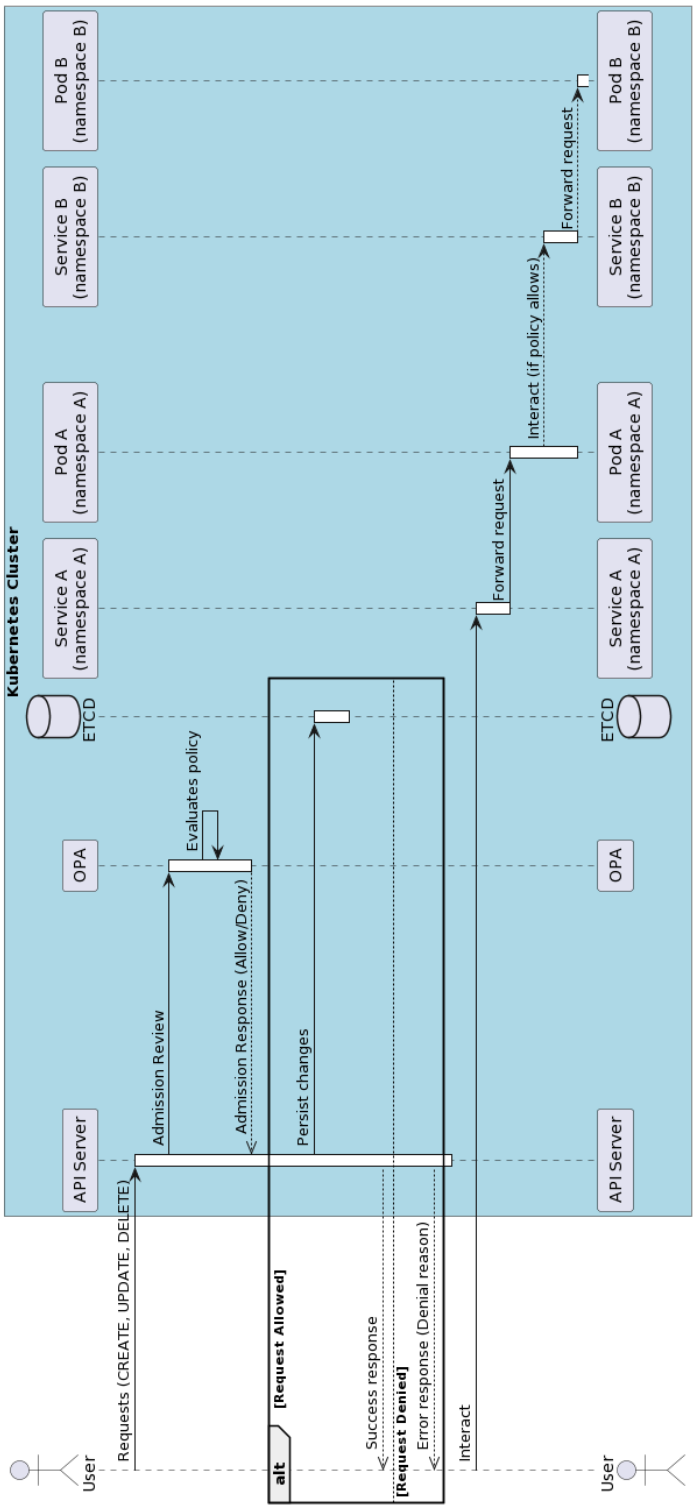


Figure 7.3: Open Policy Agent Communication Flow with Other Services

After OPA configuration is complete, Domain's analytical service is created using a Python framework FastAPI. In order to experiment with Metamycelium and to comply with requirements specified in Section 6.3, it is necessary to create a domain that entails both operational and analytical systems. Therefore, given that there is no operational system in this experiment since this is not an actual business, it is reasonable to set up a mock operational service that simulates real-world behaviour, including the generation of high-volume, high-velocity data.

In the process of constructing these domains, two primary data sources are considered and subsequently chosen: the Yelp Academic Dataset (Yelp, 2023) and the data from the U.S. Climate Resilience Toolkit's Climate Explorer (Collaborative, 2023). The selection is driven by the breadth and depth of data each source provides, their accessibility, and their potential to yield significant analytical insights when used in conjunction.

The Yelp Academic Dataset is chosen due to its extensive collection of restaurant reviews, providing a rich set of structured and semi-structured data that captures customer sentiment, business details, and temporal trends. An alternative considered is the MovieLens dataset; however, this dataset is deemed less suitable due to its narrower focus on movie ratings, which would have constrained the range of possible analyses.

The Climate Explorer dataset, offering a comprehensive record of historical and projected climate variables across the U.S., is chosen as a suitable candidate for investigating potential correlations between weather patterns and restaurant reviews. Alternatives such as the Visual Crossing Weather Data Services are considered; however, the Climate Explorer's superior geographic coverage and temporal depth are more desirable.

These datasets meet the criteria for BD due to their volume, variety, and velocity. The Yelp Academic Dataset alone contains millions of reviews for over a hundred thousand businesses, providing a high-volume and high-variety data source. The Climate Explorer

dataset, updated regularly, introduces an element of velocity to the data architecture. Together, they present an opportunity for advanced analytical exploration in a data-intensive environment.

7.3.2 Stage 2: Amalgamation

The first stage of the prototype coding and configuration is about the creation of the foundation services in the cluster. This stage is about connecting them and making sure that the architecture responds to external stimuli in an effective way. To do this, a series of scenarios are chosen to be applied against the system.

The use of scenarios in the evaluation of new RAs has become a ubiquitous practice, primarily because they provide a powerful mechanism to simulate real-world use-cases, enabling the evaluation of the architecture's performance under various conditions (Kazman et al., 1998b, 1994; Len Bass, 2021).

This is particularly critical for this experiment, as these scenarios emulate different behaviours, system loads, data volumes, and types of processing, providing insights into the architecture's resiliency, flexibility, scalability, and interoperability (Len Bass, 2021).

Inspired by the templates discussed in the works of Carroll (1995) and by the examples discussed by Kazman et al. (1998a), a series of scenarios to test the prototype's capabilities are defined. These scenarios, each addressing a different aspect of system performance and security, are detailed in Tables 7.3 through 7.8.

7.3.3 Stage 3: Scenario Testing

In this phase, following the definition of scenarios, the system is initialised and each scenario is actively executed against it. For every specific scenario, the flow and its application to the system are discussed, focusing on how the system manages each set

Table 7.3: Scenario S1: High Volume Data Ingestion Scenario

Scenario Description:	The system is subjected to a high volume of data, simulating peak data ingestion rates. The scenario is implemented when Domain A retrieves high volumes of data from Domain B, and processes it.
Relevant Quality Attributes:	Performance, Scalability.
Expected Outcome:	The system can ingest, store, and process large quantities of data without significant performance degradation.
Actual Outcome, Evaluation, Implications:	To be filled out during testing.

Table 7.4: Scenario S2: High Velocity Data Ingestion Scenario

Scenario Description:	The system is subjected to a high velocity of data, simulating peak data ingestion rates. The scenario is implemented when Domain A streams data into domain B and domain B stream processes the data.
Relevant Quality Attributes:	Performance, Scalability.
Expected Outcome:	The system can ingest, store, and process continuous stream of data without significant performance degradation.
Actual Outcome, Evaluation, Implications:	To be filled out during testing.

Table 7.5: Scenario S3: Data Variety Scenario

Scenario Description:	The system is exposed to a diverse range of data types and formats. This scenario is implemented when domain B retrieves files in different format from domain A and processes it.
Relevant Quality Attributes:	Variety, Interoperability.
Expected Outcome:	The system can handle and process different data types and formats efficiently.
Actual Outcome, Evaluation, Implications:	To be filled out during testing.

Table 7.6: Scenario S4: Complex Query Scenario

Scenario Description:	The system processes complex queries that involve multiple large datasets. This scenario happens when an external data scientist queries both domains and tries to understand a relationship between the datasets.
Relevant Quality Attributes:	Computational Efficiency, Performance.
Expected Outcome:	The system can efficiently handle and process complex queries.
Actual Outcome, Evaluation, Implications:	To be filled out during testing.

Table 7.7: Scenario S5: Secret Management Scenario

Scenario Description:	This scenario underscores the system's ability to manage secrets securely and efficiently, focusing on storage, retrieval, and rotation using Hashicorp Vault in conjunction with OpenID Connect's standard flow with bearer tokens.
Relevant Quality Attributes:	Confidentiality, Integrity, Availability.
Expected Outcome:	The system securely manages secrets, ensuring timely storage, retrieval, and rotation while maintaining confidentiality and integrity.
Actual Outcome, Evaluation, Implications:	To be filled out during testing.

Table 7.8: Scenario S6: Data Security Scenario

Scenario Description:	The system's capability is evaluated regarding ensuring data security throughout access, processing, and transmission. OpenID Connect with bearer tokens is leveraged for authentication.
Relevant Quality Attributes:	Confidentiality, Data Integrity, Authentication.
Expected Outcome:	The system guarantees data security, with secure data access, processing, and transmission. Unauthorised access attempts are effectively detected and mitigated.
Actual Outcome, Evaluation, Implications:	To be filled out during testing.

of conditions. This approach involves bringing up the system and then systematically running each scenario against it.

By doing so, key metrics are captured, and a deeper understanding of the system's response mechanisms to various stimuli is obtained. This method tests the system's resilience and performance and offers insights into its operational dynamics and potential areas for enhancement

Scenario S1: High Volume Data Ingestion Scenario

Precursor to the actual inter-domain connection, it is worth repeating the intra-domain flow. The operational service in each domain processes the data by first receiveing an event from the analytical service, requesting for the data. Once the data is processed then it is stored in Minio. From there on, a message is dispatched to a specific topic communicating the completion of the data stroage.

This message is sent to a kafka-rest-proxy and eventually into the Kafka cluster. This event includes the address of the object in MinIo and other relevant metadata. The analytics service creates a consumer group and subscribes to that specific topic. This means the analytical service, retrieves any new event dispatched to that topic on a background thread.

Once the event is received, the analytical service uses the metadata in the body of events to extract the address of the object. Once the address is available, the analytical service goes directly to MinIO to retrieve the data. This service then processes the data and stores it in a SQLite embedded database.

From there on, the necessary data quality metadata mentioned in Section 6.4.10, and bitemporality metadata mentioned in Section 6.4.13 are created. These metadata, along with the service name, the service address, and the unique identifier, are then dispatched as an event to Kafka on a specific topic. Data Lichen creates a consumer group, subscribes to that specific topic, retrieves the data, and then renders the information in

regards to each domain. Data Lichen renders this information in a custom coded table as per the Figure 7.4. The use of a unique identifier for each service ensures that the listing and display of services are consistent and reliable, providing the same results with each operation. This flow is communicated in Figure 7.5

Metadata Information

Unique Identifier	Service Name	Service Address	Completeness	Validity	Accuracy	Processing Time	Actual Time	Processing Duration
6c055c9a-7cdf-4c47-bc6c-aefb135ec9d6	Weather domain data	http://localhost:8500	87.9049676025918	87.9049676025918	87.9049676025918	2023-08-10 20:42:47	2023-08-10 20:42:47	0.08 seconds
d02aa17d-7cac-4695-bdab-3dc7f3194b8a	Customer domain data	http://localhost:8000	100	100	100	2023-08-13 15:32:03	2023-08-13 15:32:02	0.72 seconds

Figure 7.4: Data Lichen Dashboard in Local Development Environment

This scenario follows the flow of domain A requesting a 5GB JSON file from domain B. Domain A owns the weather data, while domain B owns the customer ratings and business data. For this purposes, first an event is dispatched to Data Lichen from domain to get a list of all datasets available with metadata including actual-time, processing-time, completeness and accuracy.

This metadata includes the address of the data domain to be retireved. This address is then used to dispatch an event to the domain B to retrieve the data. This triggers the internal mechanism of the domain B which results in the creation and storage of the datasets in the distributed storage service. Once this process is complete, an event is dispatched to the data processing completion topic, which domain A is subscribed to. An overview of Kafka topics used in this experiment is portrayed in Figure 7.6.

The event that communicates the completion of the data processing includes the address of the data as well. Domain A uses this address to fetch the data. Once data is fetched, then it is processed. This flow is depicted in Figure 7.7.

For this scenario, understanding system performance and resource utilisation is pivotal. Data Ingestion Rate provides a direct measure of the system's throughput, indicating how efficiently the system can absorb and manage vast amounts of incoming

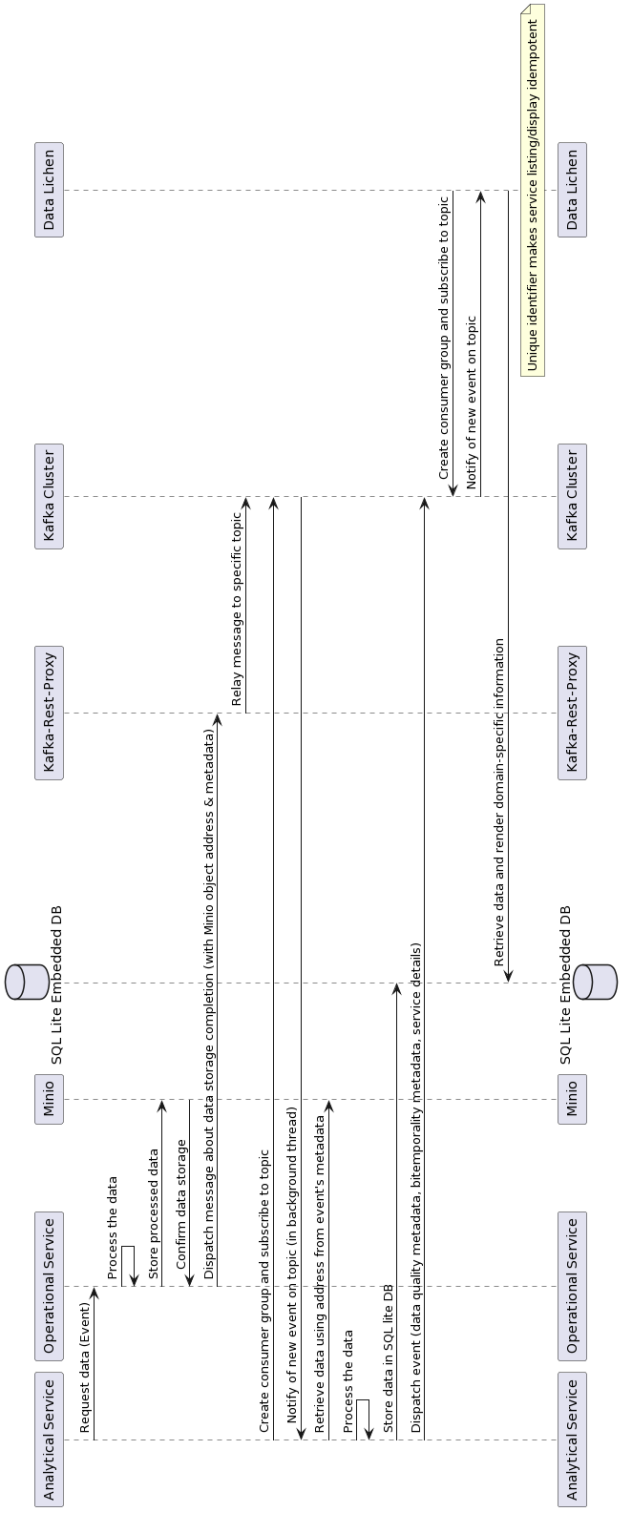


Figure 7.5: Intra-domain communication in Metamycelium's prototype

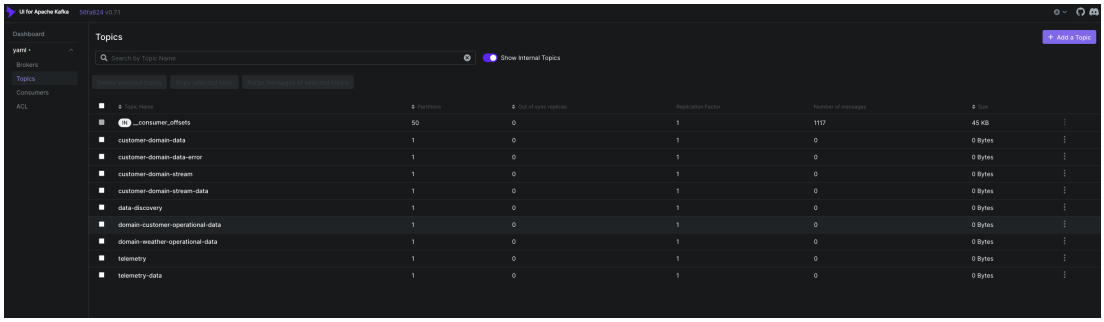


Figure 7.6: Overview of kafka topics

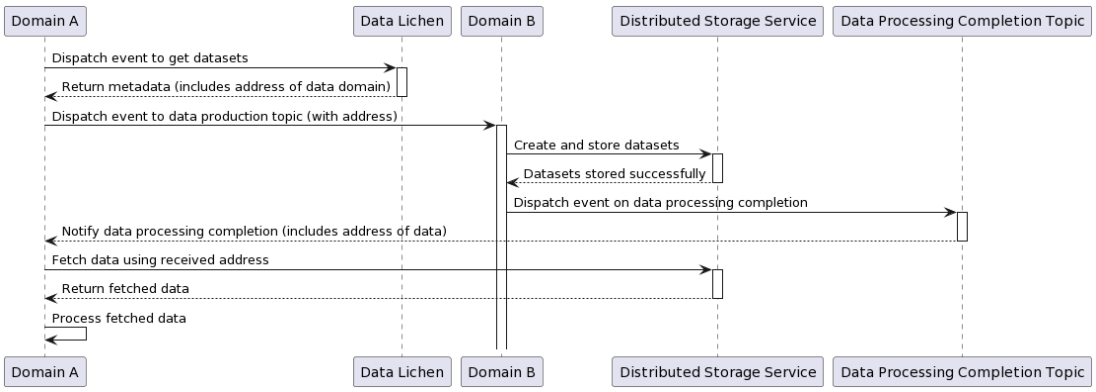


Figure 7.7: Scenario S-1 Flow Diagram

data. A high rate signifies optimal data handling capabilities, essential for scenarios where rapid data inflow is expected. Latency, broken down into ingestion and processing components, offers insights into system responsiveness. Low latency ensures timely data availability and processing, which is crucial for systems where delays can have downstream impacts.

Moreover, CPU utilisation and memory utilisation are fundamental indicators of the system's resource efficiency. High CPU or memory consumption may indicate bottlenecks, inefficiencies, or potential areas of optimisation, especially during peak loads. Lastly, the error rate offers insights into the system's robustness and reliability. A low error rate, even under high data volumes, suggests that the system can consistently handle and process data without faltering.

Together, these metrics holistically assess the system's capability to maintain performance and stability under intense data ingestion demands. In the following sections each metric is described in relation to the mechanism that has been triggered in the system.

Data Ingestion Rate: To measure the data ingestion rate, each service is instrumented using OpenTelemetry SDKs to automatically capture telemetry data for FastAPI requests and manual spans for specific operations, with the collected data being exported to Kafka via the KafkaRESTProxyExporter class.

Figure 7.8 portrays the open telemetry Kafka topic and an extensive amount of events registered to this topic. All the events dispatched to this topic are consumed by the open telemetry processor service. This service collects and updates metrics which are then scraped by Prometheus and eventually transferred to Grafana. This flow is communicated in Figure 7.9.

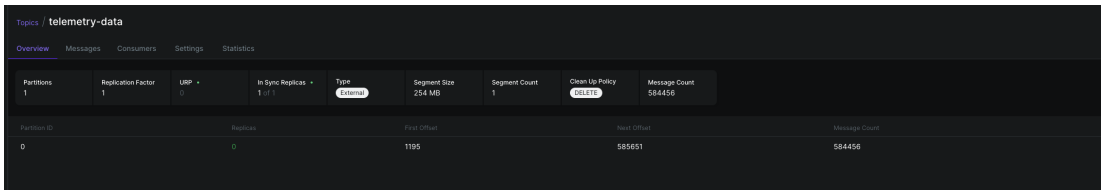


Figure 7.8: Open Telemetry Topic

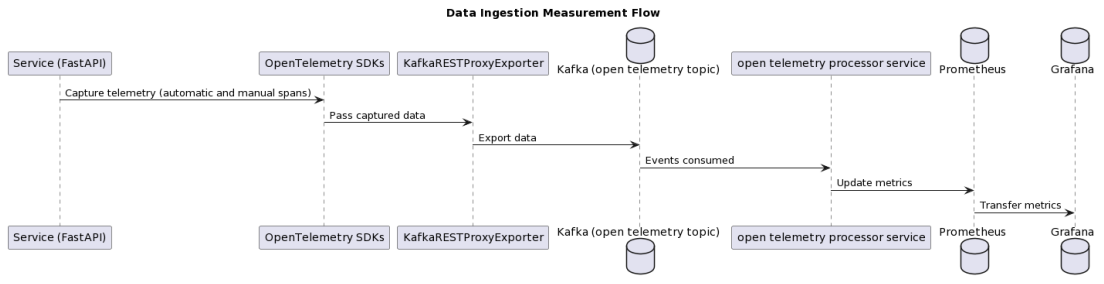


Figure 7.9: Open Telemetry Flow

The JSON displayed in Figure 7.10 depicts an example of a trace span JSON that is communicated over Kafka to the open telemetry service. Providing this example helps

illustrate the specific data format and fields used to represent and propagate distributed traces across the various services, which is a key mechanism for collecting the telemetry data underlying the metrics analysed in this scenario.

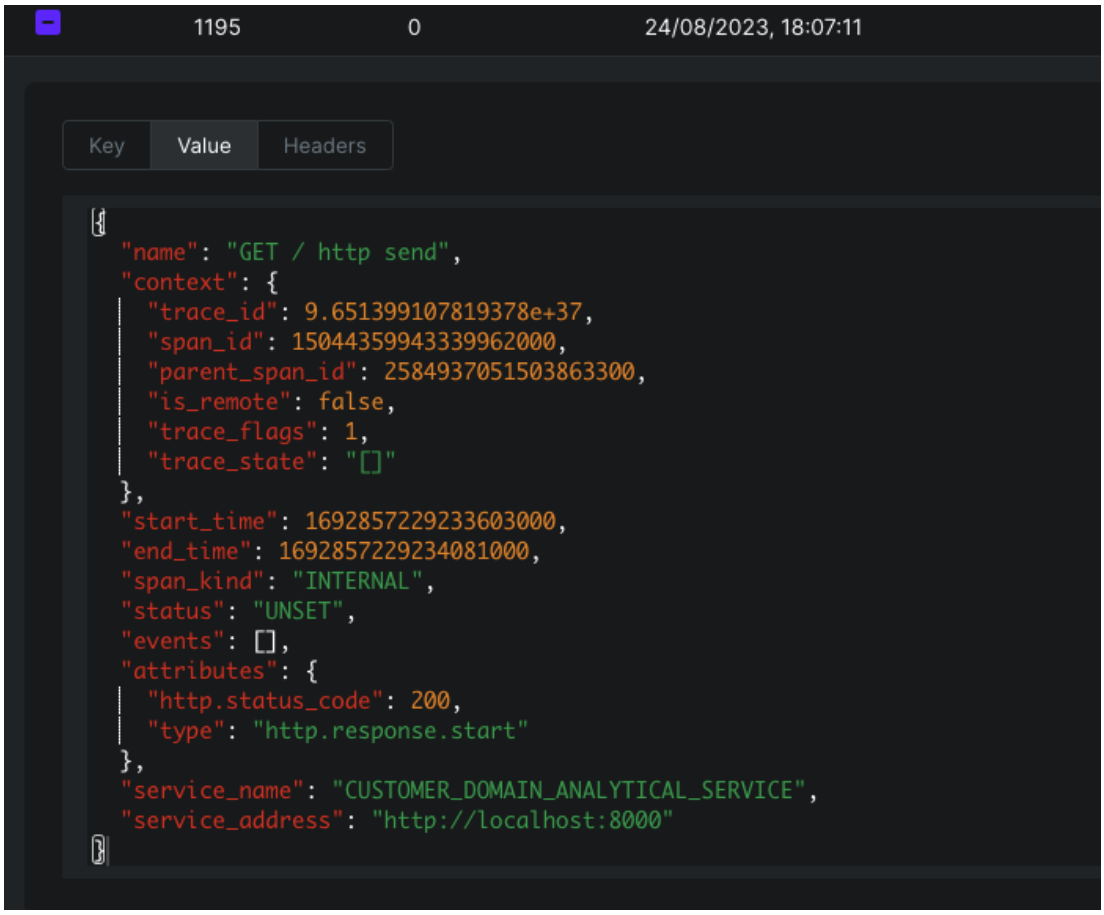


Figure 7.10: Open Telemetry Trace Span

Prior to the presentation of the results of the experiment, it is important to pinpoint what 'high load of data' within the scope of this study is signifies. The vast landscape of BD lacks a universally agreed-upon threshold delineating *high volume* especially in decentralised, domain-driven architectures like Metamycelium. No singular benchmark exists that uniformly qualifies what constitutes a high data load in such systems.

However, academic endeavours and industry white papers (A. Smith & Johnson, 2022; *State of Data Mesh 2022*, 2022; Amazon Web Services, 2023) often indicate that daily data ingestion can range in the orders of terabytes, if not more, especially in the

context of high-velocity streams like social media or e-commerce transactions.

For this experiment, achieving seamless transfer of multiple gigabytes of data, specifically in JSON format, which is more verbose and demanding than binary formats, between domains underlines a significant stress test. By chunking the Yelp academic dataset into 400KB sizes and simulating multiple transfers, the aim is to push the boundaries of the prototype's capacity and assess its robustness under intensified data traffic.

This setup aligns with real-world use cases where systems must continuously process streams of incoming data, often from various sources and in fluctuating volumes, without compromising on performance or integrity. The decentralised, event-driven architecture of Metamycelium should ensure that data can be ingested, processed, and retrieved seamlessly, regardless of the number of times a particular dataset is transferred or accessed. This not only tests the system's ability to handle sheer data volume but also its resilience and efficiency in an environment that mimics the high-velocity data influxes characteristic of contemporary BD scenarios.

In the context of the experiment, all transferred data is stored within MinIO. After successful storage, a confirmation is consistently communicated back. This streaming, storing, and confirming process was repeated multiple times, showcasing its reliability even under high data volume conditions. Such consistent and error-free transmission in demanding scenarios aligns with various academic studies emphasising the critical role of steadfast communication in distributed systems (Kleppmann, 2017). This process can be mathematically modelled using logical implications to ascertain the robustness of the system, demonstrating a transitive relationship between the events.

Let's denote the following:

- P as the event where "data is transferred from Domain A to Domain B".
- Q as the event where "data is successfully stored in MinIO".

- R as the event where "confirmation of successful storage is communicated back".

From the empirical observations, it is deduced that:

$$P \implies Q \tag{7.1}$$

$$Q \implies R \tag{7.2}$$

Thus, utilising the transitive property of implications (Johnson, 2019), it is inferred that:

$$P \implies R \tag{7.3}$$

This logical sequence implies that every instance of data transfer from Domain A to Domain B guarantees a subsequent confirmation of its successful storage. Such a relationship underscores the system's efficiency and reliability in managing large volumes of data, resonating with the findings of K. Lee and Wang (2020) who emphasised the significance of ensuring data consistency and integrity in distributed storage mechanisms.

As illustrated in Figure 7.11, the Kafka data ingestion rate peaked at 1,693,816,791 bytes at 2023-09-04 20:40:00 in customer domain's analytical service, indicating the flow of data from the operational service to the analytical service. This bar chart also communicates the gradual increase in the load placed on the service. Furthermore, the total data ingested over the experiment duration amounted to 1.6 billion bytes, as per the Figure 7.12.

In the same vein, data ingestion rate for the weather domain's analytical service is portrayed in Figure 7.13 peaking at 1,693,893,416 bytes at 2023-09-05 18:00:00. This service has ingested a total of 1.6 billion bytes.

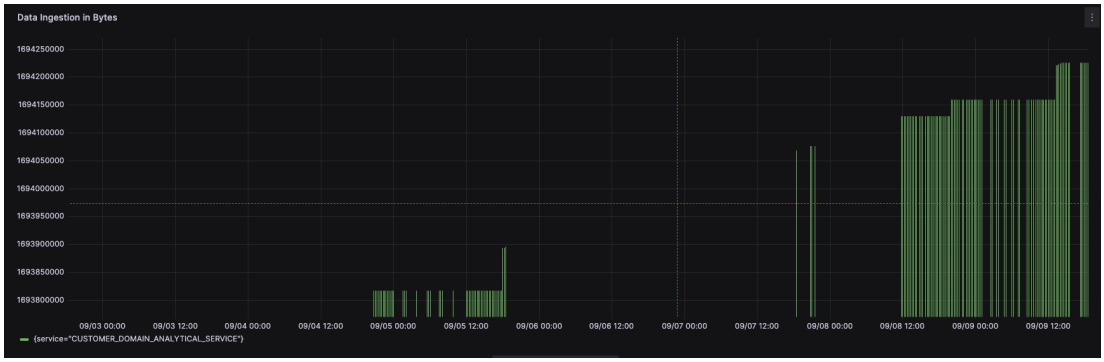


Figure 7.11: Kafka Data Ingestion in Bytes in Customer Domain's Analytical Service

Figure 7.12: Total Data Ingested in Bytes in Customer Domains' Analytical Service

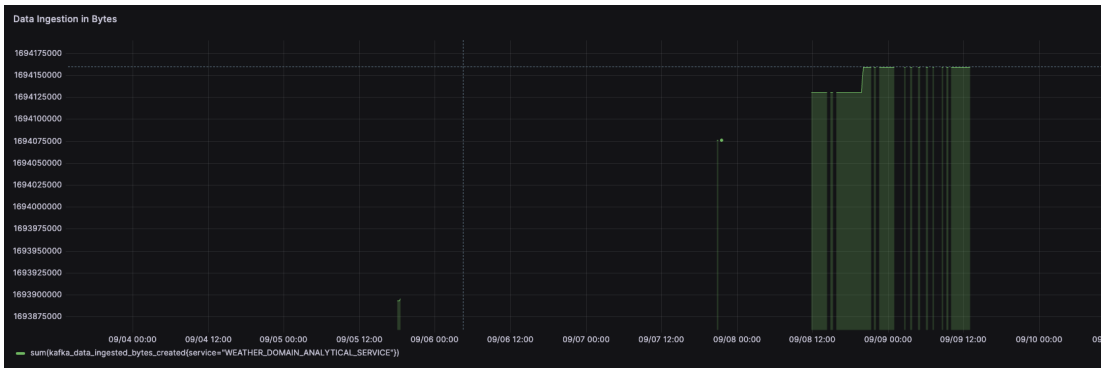


Figure 7.13: Total Data Ingested in Bytes in Weather Domain's Analytical Service

In order to provide an empirical basis for the system's performance during high data ingestion, the metrics as shown in the screenshots are visualised in Grafana. This monitoring system provides time-stamped data points that indicate the amount of data ingested at various intervals. The Table 7.9 captures the data points over a specific time

Table 7.9: Data Ingested in Customer Domain's Analytical Service

Timestamp	Data Ingested (bytes)
2023-09-04 20:40:00	1693816791
2023-09-04 20:45:00	1693816791
2023-09-04 20:50:00	1693816791
2023-09-04 20:55:00	1693816791
2023-09-04 21:00:00	1693816791
2023-09-04 21:05:00	1693816791
2023-09-04 21:10:00	1693816791
2023-09-04 21:15:00	1693816791
2023-09-04 21:20:00	1693816791
2023-09-04 21:25:00	1693816791
2023-09-04 21:30:00	1693816791
2023-09-04 21:35:00	1693816791

span in customer domain's analytical service. Moreover, the Table 7.10 presents the data in weather domain's analytical service.

After capturing the ingestion rate in bytes, it is essential to further explore other metrics to provide a comprehensive analysis of the system's performance during high data load conditions.

Ingestion Latency: Ingestion latency captures the delay from the instant a data record is published to Kafka to the moment it is processed by the system. Precisely, it is determined for each consumed Kafka record using the formula:

$$Latency = current\_time - publish\_time \tag{7.4}$$

Where:

- current_time denotes the time at which the record is being processed.
- publish_time represents the timestamp when the record was published to Kafka. If the Kafka record lacks a timestamp, the current time is assigned as a default, making the latency calculation a fallback measure.

Table 7.10: Data Ingested in Weather Domain's Analytical Service

Timestamp	Data Ingested (bytes)
2023-09-05 18:00:00	1693893416
2023-09-05 18:15:00	1693893416
2023-09-05 18:30:00	1693895179
2023-09-07 20:45:00	1694076236
2023-09-07 21:00:00	1694076236
2023-09-07 21:30:00	1694076236
2023-09-08 11:45:00	1694130131
2023-09-08 12:00:00	1694130131
2023-09-08 12:15:00	1694130131
2023-09-08 12:30:00	1694130131
2023-09-08 12:45:00	1694130131
2023-09-08 13:00:00	1694130131

This latency metric is presented in Figure 7.14. According to this figure the latency is negilible in both domain's analytical services, with one spike to 400 seconds and the rest pretty stable ingestion latency around 50 to 100 seconds. The weather domain shows even less latency in comparison to the customer latency at an average of 50 seconds. This is due to the fact this domain consumes less data.

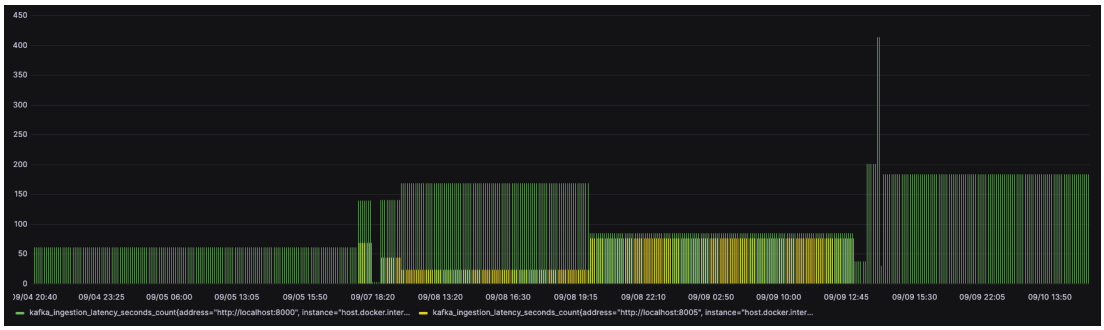


Figure 7.14: Customer and Weather Domain's Analytical Service Latencies

The observed ingestion latency results, including occasional spikes up to 400 miliseconds, serve as a credible relative measure of the system's performance in this experimental setup. While lacking absolute reference points due to the absence of a real-world implementation (as noted in Section 7.1), these results align with the expected complexities and latency characteristics of large-scale, asynchronous, distributed big

data systems leveraging Kafka, as reported in industry studies and benchmarks. For instance, the Kafka documentation from Confluent suggests that latencies under 10 milliseconds are considered excellent, while latencies up to a few seconds are reasonable for most use cases (Confluent, 2024).

Similarly, a study by Kreps (2023) on benchmarking Apache Kafka found median latencies ranging from a few milliseconds to over a second, depending on the cluster size and configuration. Sporadic latency spikes can be attributed to Kafka's intricate underlying mechanisms, such as indexing, offset management, and partition assignment strategies, as well as potential influences from the operating system and environment. Despite these inherent challenges, the predominantly low and stable latency measurements demonstrate the architecture's capability to maintain reasonable responsiveness while handling high-velocity, high-volume data ingestion across multiple domains.

CPU Usage: The telemetry processing service, central to the infrastructure, provides vital insights when monitored for CPU utilisation. Monitoring this service is important as it offers direct insights into data processing efficiency, where elevated CPU consumption could signal potential processing bottlenecks.

Additionally, since CPU utilisation metrics are determined over specific intervals, and given the asynchronous receipt of telemetry data from various services, tracking the CPU usage of individual services might not accurately represent the real-time system load. However, the telemetry service, due to its consistent data processing role, ensures that its metrics are indicative of the actual strains on the system.

This observation becomes crucial when evaluating scalability and predicting future infrastructure requirements. Importantly, it must be noted that CPU usage metrics can be influenced by the operating system and environment in which the services operate. Thus, the reported values might not be absolute but serve as a credible relative measure of system resource consumption.

Figure 7.15 illustrates the CPU utilisation trends of the telemetry processing service, emphasising the interplay between data intake rates and resource consumption.

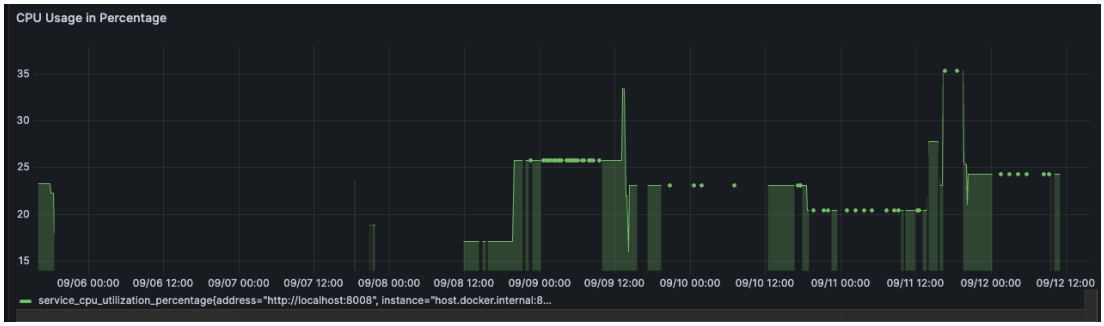


Figure 7.15: Open Telemetry Service CPU Usage

In the observed metrics from the Open Telemetry service, it is evident that the CPU usage remains relatively consistent at around 24 percent, with intermittent peaks observed, pushing the utilisation to 25.8 percent at specific intervals. This behavior can be attributed to the underlying architecture, where events are continuously streamed to the connected services. To further elucidate this point, a detailed tabulation of the CPU utilisation over a specific period is presented in Table 7.11. This tabulation provides insights into the fluctuation in CPU usage across different times and conditions, revealing periods of consistent CPU usage, instances of inactivity, and occasional spikes in utilisation.

The steady CPU utilisation averaging around 24 percent with occasional peaks up to 25.8 percent supports the assertion of a stable processing environment for the telemetry service. As outlined in the CPU utilisation study by Gregg (2014), sustained high CPU usage above 80-90 percent typically indicates resource saturation and potential performance issues. In contrast, the moderate utilisation observed here suggests sufficient resources to handle the incoming event load without significant computational strain.

Morover, the Kubernetes cluster CPU usage (Figure 7.16) displays a consistent number of 43% with occasional hikes up to 250%. These hikes are usually observed when a lot of data is transferred through the network and into the Minio.

Table 7.11: Telemetry Processing Service CPU Utilisation

Date and Time	Average CPU Usage (%)
09/06 00:00	23%, 24%
09/06 12:00	0 (no processes)
09/07 00:00	0 (no processes)
09/07 12:00	0 (no processes)
09/08 00:00	24%, 18%
09/08 12:00	17%
09/09 00:00	26%
09/09 12:00	26% (with a spike to 34%)
09/10 00:00	24%
09/10 12:00	24%
09/11 00:00	22%
09/11 12:00	22% (with spikes to 27% and 35.2%)
09/12 00:00	24%
09/12 12:00	24%

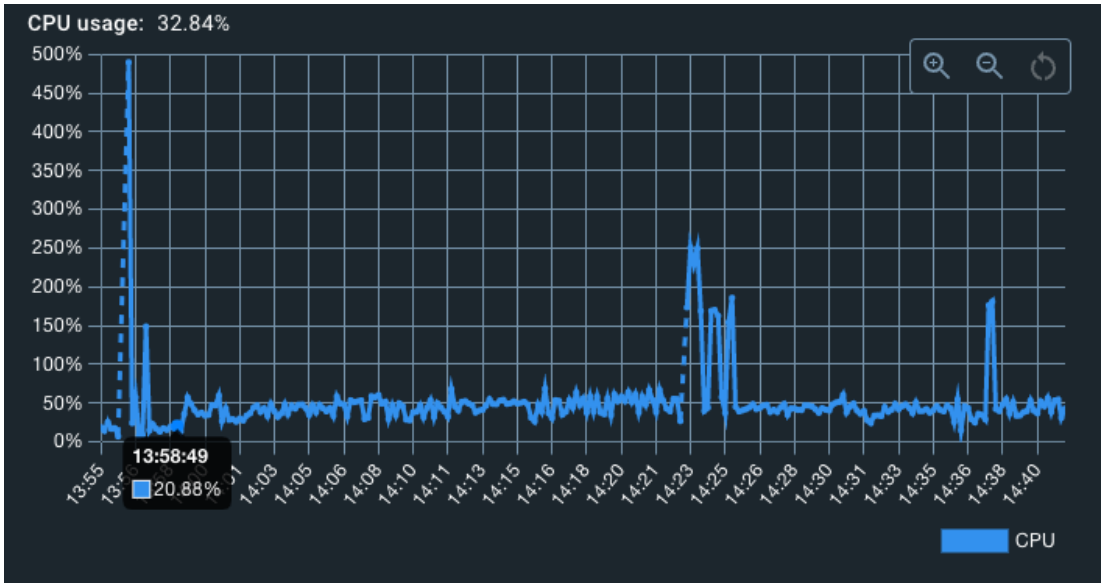


Figure 7.16: Kubernetes Cluster CPU Usage

Along the same lines, as portrated in Figure 7.17, an average 5.37GB of data read and 12.2GB of data write has taken place in different intervals. Additionally, 11.6GB of data has been received over the network, and 13.9MB of data has been sent (Figure 7.18).

From a holistic examination of the telemetry and infrastructure metrics presented, it

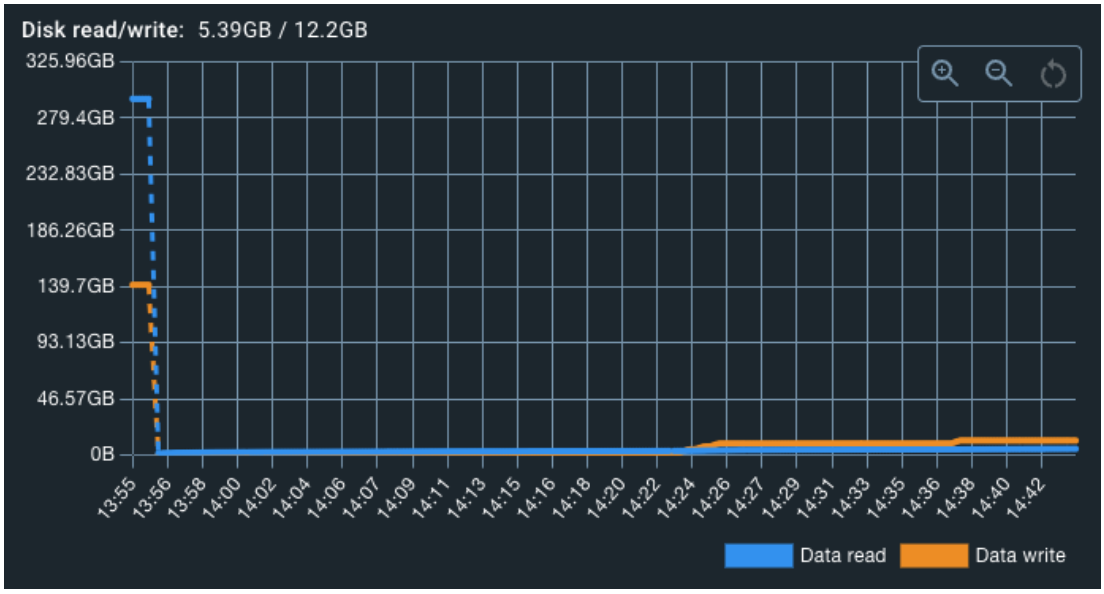


Figure 7.17: Kubernetes Cluster Read and Write Statistics

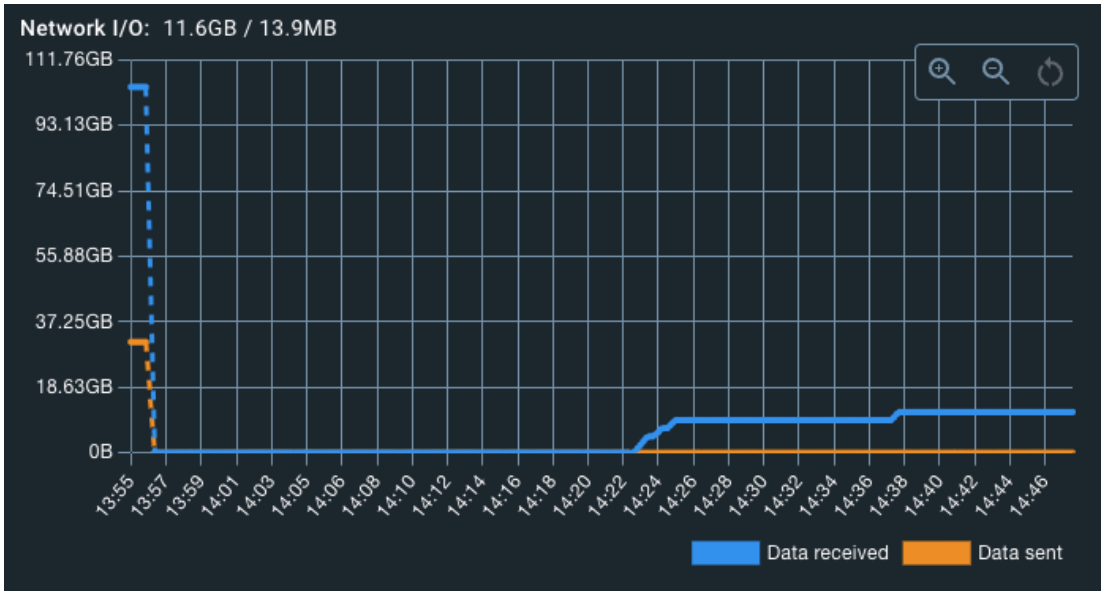


Figure 7.18: Kubernetes Cluster Network Statistics

becomes evident that the asynchronicity and event-driven architecture of our BD system play a pivotal role in its performance dynamics. Given this asynchronous event-driven paradigm, it is of importance to recognise that the sheer volume, measured in gigabytes, of data processed becomes a lesser concern compared to how efficiently the system handles incoming data streams.

The consistently moderate CPU usage of the telemetry service, even amidst varying data intake rates, underscores its robustness in data processing. Similarly, while the Kubernetes cluster exhibits occasional spikes in CPU usage during data-intensive operations, it largely maintains a balanced load. This is a testament to the system's adeptness in managing and distributing tasks across its nodes.

The data statistics on read, write, and network transfers further explain this point. While the gigabytes of data read, written, or transferred might seem significant, what stands out is not the absolute quantity but the system's efficacy in handling these operations. An average of 5.37GB read and 12.2GB written, juxtaposed with the network statistics presented in Figure 7.18, indicates the system's proficiency in data management and network operations, without being overwhelmed.

The metrics not only highlight the stability and efficiency of our BD infrastructure but also provide empirical evidence that, in an event-driven asynchronous system like Metamycelium, it is not just about the volume of data processed but more about the efficiency and stability with which it is handled. This insight suggests potential implications for scalability considerations in similar architectures going forward.

Memory Utilisation: Memory usage is a pivotal metric in evaluating system performance. Efficient memory management is indicative of a system's ability to handle real-time data and scale under different workloads. The memory utilisation observed in the telemetry processing service, a central component of the infrastructure, provides specific insights into system data handling capabilities. Analysing these memory patterns offers a clearer understanding of the service's efficiency in processing varying data rates and highlights areas where memory optimisation might be necessary to enhance overall performance.

Within the weather domain, memory usage is divided into operational and analytical services. Operational services, serving static data, have modest memory consumption

due to their less computationally intensive nature. In contrast, analytical services may show variable memory patterns depending on the complexity of data analysis.

The customer domain follows a similar trend, with operational services displaying stable memory usage due to serving primarily static customer data. Meanwhile, the analytical services, which dive deeper into customer behaviours and preferences, might see periodic spikes, especially when handling sizable datasets.

All these observations can be visualised in Figure 7.19, which showcases the memory utilisation trends across the various services. The address of each service is stated in the corresponding service definitions in the Github respoistories, but it is worth mentioning that the service on port 8000 is the operational service in the customer domain's operational service, while the service on port 8001 is the customer domain's analytical service. The service on port 8005 is the weather domain's operational service, while the service on port 8006 is the weather domain's analytical service. Finally, the service on port 8008 is the telemetry processing application.

Figure 7.19: Memory Usage for Various Services in the Cluster

The interrelation between CPU and memory utilisation is further illustrated in Table 7.12. This tabulation delineates memory usage across different services at corresponding times of CPU utilisation. It is observable that periods of heightened CPU activity coincide with increased memory allocation, particularly within the telemetry

processing service. This alignment elucidates the system's responsive scaling capabilities in periods of intensive computational demand. Similarly, instances of reduced CPU usage are mirrored by a lower memory footprint, underlining efficient resource management during less intensive processing periods. Such adaptive memory utilisation is critical to maintaining system performance and ensuring the seamless management of real-time data across the infrastructure.

Table 7.12: Correlated Memory and CPU Usage for Various Services.

Date & Time	Svc A (B)	Svc B (B)	Svc C (B)	Svc D (B)	Svc E (B)
09/06 00:00	H (12,750,000,000)	H (12,800,000,000)	L (4,700,000,000)	H (14,000,000,000)	H (13,500,000,000)
09/06 12:00	L (6,350,000,000)	L (6,400,000,000)	L (4,650,000,000)	M (10,850,000,000)	M (10,650,000,000)
09/07 00:00	L (6,375,000,000)	L (6,425,000,000)	L (4,675,000,000)	M (10,870,000,000)	M (10,670,000,000)
09/07 12:00	L (6,360,000,000)	L (6,410,000,000)	L (4,680,000,000)	M (10,860,000,000)	M (10,660,000,000)
09/08 00:00	M (9,560,000,000)	M (9,610,000,000)	L (4,720,000,000)	H (13,800,000,000)	H (13,300,000,000)
09/08 12:00	M (9,580,000,000)	M (9,630,000,000)	L (4,750,000,000)	H (13,850,000,000)	H (13,350,000,000)
09/09 00:00	H (12,780,000,000)	H (12,830,000,000)	L (4,770,000,000)	H (14,100,000,000)	H (13,600,000,000)
09/09 12:00	H (12,760,000,000)	H (12,810,000,000)	L (4,760,000,000)	H (14,050,000,000)	H (13,550,000,000)
09/10 00:00	M (9,570,000,000)	M (9,620,000,000)	L (4,730,000,000)	M (10,900,000,000)	M (10,700,000,000)
09/10 12:00	M (9,590,000,000)	M (9,640,000,000)	L (4,740,000,000)	M (10,920,000,000)	M (10,680,000,000)
09/11 00:00	M (9,565,000,000)	M (9,615,000,000)	L (4,710,000,000)	M (10,880,000,000)	M (10,670,000,000)
09/11 12:00	H (12,770,000,000)	H (12,820,000,000)	L (4,775,000,000)	H (14,110,000,000)	H (13,610,000,000)
09/12 00:00	M (9,585,000,000)	M (9,635,000,000)	L (4,745,000,000)	M (10,930,000,000)	M (10,690,000,000)
09/12 12:00	M (9,600,000,000)	M (9,650,000,000)	L (4,755,000,000)	M (10,950,000,000)	M (10,700,000,000)

Legend: B - Bytes, H - High Usage, M - Medium Usage, L - Low Usage. Service A corresponds to localhost:8000, B to localhost:8001, C to localhost:8005, D to localhost:8006, and E to localhost:8008.

Error Rate: Within the FAST API services, error rates are determined by carefully examining logs and by tracking exception blocks. Each domain has a dedicated Kafka topic allocated solely for error reporting. Yet, as illustrated in Figure 7.20, only the OpenTelemetry service reported errors. The Weather and Customer domains, contrastingly, remained error-free.

The inherent resilience of event-driven systems significantly contributes to this low error rate. These systems' characteristic decoupling between components ensures that transient failures in one service are often localised, preventing them from cascading into system-wide disruptions. This behaviour is evident in the prototype, which underscores the intrinsic benefits of event-driven architectures in maintaining system robustness.

With the above metrics in conjunction with the ingestion rate, a comprehensive

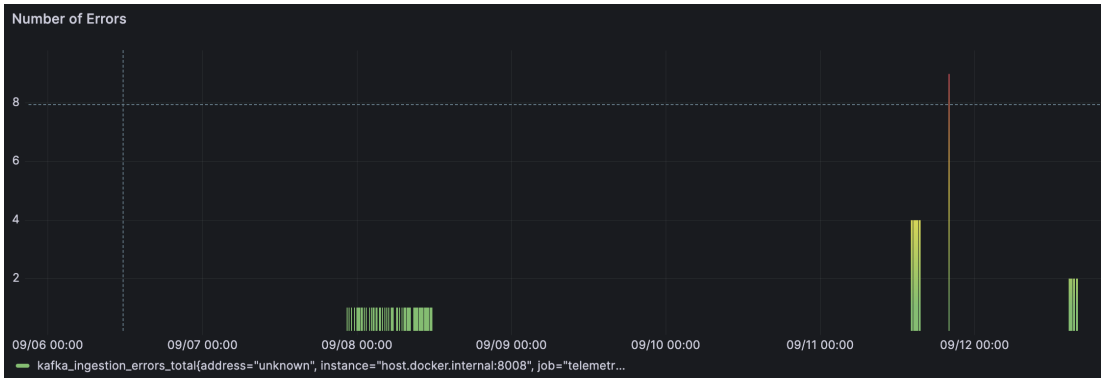


Figure 7.20: Number of Errors in Telemetry Processing Service

understanding of the system's operational efficiency, resilience, and overall performance can be obtained.

Scenario S2: High Velocity Data Ingestion Scenario

Scenario S2 tests the system's capabilities under high data velocity conditions, approximating peak ingestion rates. This experiment involved Domain A (Customer Domain) streaming data into Domain B (Weather Domain), with Domain B subsequently processing the streamed data. To facilitate this, a dedicated Kafka topic, named customer-domain-stream-data, is established, earmarking it exclusively for the streaming operations between the two domains.

Concurrently, within the customer domain, a novel endpoint is created. This endpoint is responsible for streaming out the data that had been previously stored. In synchrony with this, the weather domain analytical service adopts a long-polling mechanism. By doing so, it could efficiently subscribe to and intake the continuous data stream relayed from the customer domain.

An analysis is conducted based on the metrics of interest:

Volume and Size of Messages: The system ingested an impressive total of 771,305 messages. Cumulatively, these messages totaled approximately 1 GB in volume, all

of which are directed to the customer-domain-stream-data topic. The statistics of the streaming topic is depicted in Figure 7.21.

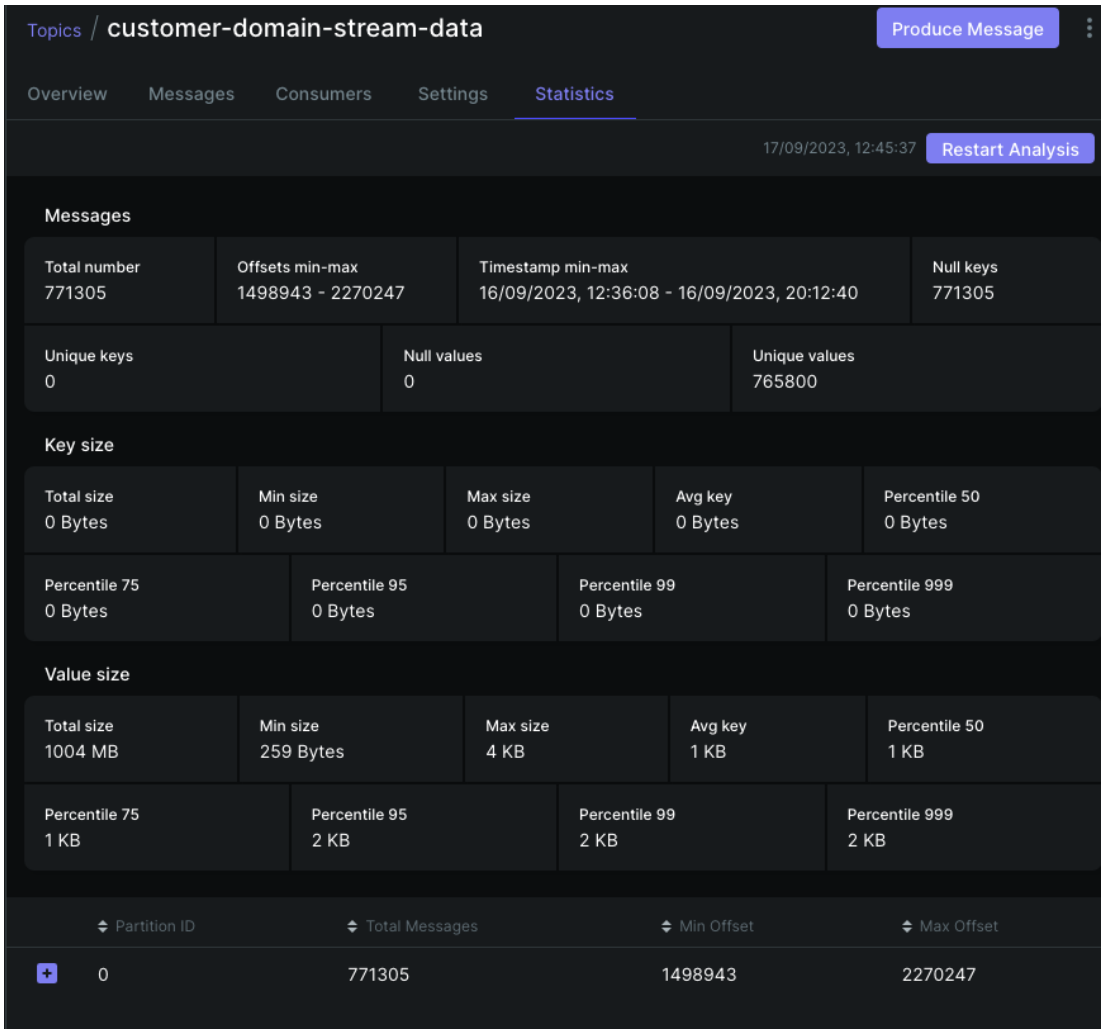


Figure 7.21: Customer Domain Streaming Topic Statistics

Memory and CPU Utilisation: Memory footprint during this high-velocity data scenario is important. Continuous monitoring of memory usage was instrumental in gauging system health and efficiency. Memory consumption in the customer domain peaked noticeably, registering 12,852,592,640 bytes. This is attributed to the inherent complexity of the chunking logic employed in the streaming operation (Figure 7.22). Contrarily, the weather domain, which acts as the data consumer, reported more modest

memory usage. This is inferred to be a consequence of its simpler data processing logic, which is devoid of the overheads of chunking (Figure 7.23).

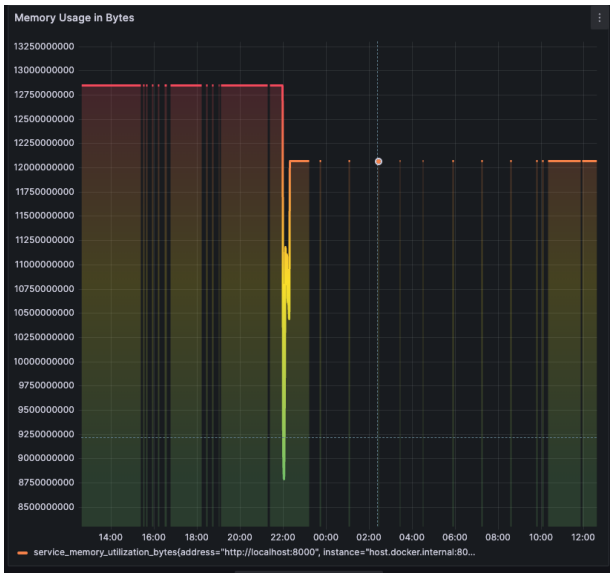


Figure 7.22: Memory Utilisation in Customer Domain in High Velocity Case

Moreover, the CPU usage is only 5% increased in comparison to the stats stated in Section 7.3.3 for both the Kubernetes cluster and corresponding services, therefore, no extra figure is provided for this metric.

Processing Duration and Ingestion Latency: The ingestion latency, which is the delay from when a data packet is received to the point it is ingested for processing, remained consistent over the monitored period. The latency consistently measures 0.0000148 seconds at different points in time during the monitoring period. This observation indicates a lack of significant processing delays in the system's data ingestion mechanism, although a comparative analysis with other platforms' performance benchmarks would provide a more comprehensive evaluation (Figure 7.24). This consistent latency suggests the system's capability to sustain its performance levels without disruption, particularly during instances of high data traffic.

Moreover, processing duration gives us insights into the time taken by the system

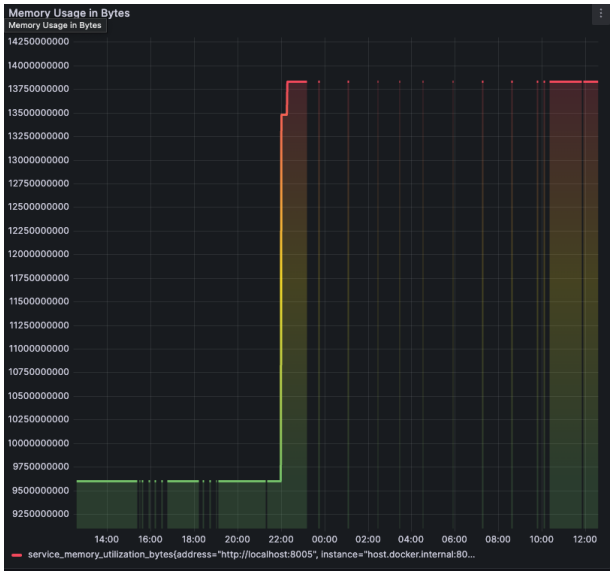


Figure 7.23: Memory Utilisation in Weather Domain in High Velocity Case

Figure 7.24: Kafka Ingestion Latency in Streaming Case

to process the ingested data. The Weather Domain showcases a consistent processing time of 1,694,491,558 nanoseconds (or approximately 1.6945 seconds) across different timestamps (Figure 7.25). This uniformity indicates a stable data processing mechanism, unaffected by potential variables during the streaming process.

The evaluation of the High Velocity Data Ingestion Scenario (S2) indicated that the system consistently managed high-velocity data ingestion within the observed parameters, reflecting its capacity to handle such scenarios effectively. Through the creation of a dedicated Kafka topic and the strategic implementation of a new endpoint in the customer domain to facilitate data streaming, peak data ingestion rates are effectively

Figure 7.25: Processing Duration in Streaming Case in the Weather Domain

simulated. The system's memory usage patterns showcase its capability to manage substantial data flows. Specifically, the customer domain, driven by its chunking logic, showcases a higher memory usage compared to the Weather domain. Nevertheless, both domains effectively handled the processing and ingestion of a substantial total of 771,305 messages (approximately 1GB in size).

Further, the consistent ingestion latency and processing duration underscore the system's robustness. In conclusion, the system demonstrates a reliable capacity to ingest, store, and process a continuous stream of high-velocity data without significant performance degradation, meeting the expected outcomes of Scenario S2. In addition, system has been constantly monitored to ensure that the allocated memoery for streaming is freed upon the completion of the transmission. This monitoring yielded no sign of memory leak and all memeory has been freed upon the completion of streaming.

Given the comprehensive results obtained from scenarios S1 and S2, which encompass both high-volume and high-velocity data ingestion processes, it is deemed that scenario S3's focus on data variety is inherently addressed and validated, thus obviating the need for a separate testing phase for scenario S3.

Scenario S4, S5, S6

Next, scenarios S4, S5, and S6 are collectively assessed. The decision to test these scenarios concurrently stems from their shared system processes and the capability

to ensure a comprehensive system evaluation while maximising efficiency. Moreover, their intertwined nature, particularly concerning OpenID Connect and bearer token mechanisms, suggests that a simultaneous test would yield a holistic system performance insight.

To accurately emulate the behaviour and workload of an external data scientist, a Python service is initialised. Although Jupyter Notebook was initially considered to mimic the data scientist's operations, a standalone Python service emerged as the superior choice. Its low-level nature permits granular control and facilitates the capturing of metrics more effectively. Just like other services, this Python service is instrumented to dispatch telemetry data to telemetry processing application through Kafka rest proxy.

A pivotal component in this ecosystem is Keycloak, the authentication and authorization server. When the Python service initiates, it first obtains authorization by creating a client in Keycloak. The subsequent secrets, vital for the service's operations, are securely stored in and retrieved from HashiCorp Vault.

It is this intricate dance of authentication, facilitated through OpenID Connect standards, that allows the simulated data scientist to seamlessly gain access to Data Lichen. Once authenticated, the data scientist can view available datasets across domains and, subsequently, make the requisite network requests to fetch the desired data. This flow is depicted in Figure 7.26.

It is noteworthy that the Minio secret key and access key could also be stored in Vault and retrieved in each service; nevertheless, this is not done due to time and resource constraints. This does not risk the integrity of this experiment. For scenarios S4, S5, and S6, the metrics below are selected:

1. **Query Processing Time (QPT):** Measures the duration from when a complex query is initiated to when its results are returned. It encapsulates computational

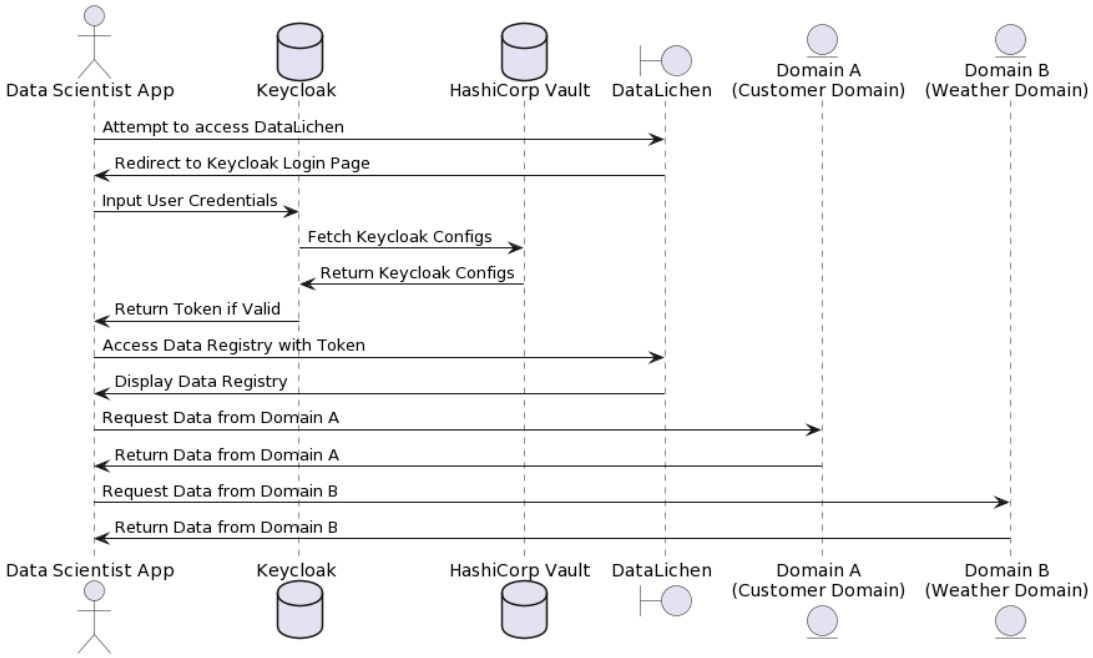


Figure 7.26: Data Scientist Flow with Authentication

efficiency and performance.

$$QPT = t_{querycompleted} - t_{queryinitiated} \tag{7.5}$$

2. **Secret Retrieval Latency (SRL):** Captures the time taken to retrieve secrets. It summarises the system's efficiency in secret management.

$$SRL = t_{secretretrieved} - t_{retrievalinitiated} \tag{7.6}$$

3. **Data Integrity Verification (DIV):** Assesses the integrity of data during transmission and processing, ensuring data security.

$$DIV = Hash(DataSent) \stackrel{?}{=} Hash(DataReceived) \tag{7.7}$$

After retrieving the domains data through an authenticated approach using OpenID

Connect and Vault on multiple occasions and across different days, the subsequent findings emerges:

Secret Retrieval Duration: The observed metrics showed for majority a duration of 3 seconds across timestamps. This indicates a reliable and predictable secret management system as portrayed in Figure 7.27.

Figure 7.27: Data Scientist Secret Retrieval Delay in Seconds

CPU Utilisation in Data Scientist Application: The CPU utilisation is steady at 18.6% throughout the captured times. Given the deployment on the Apple M2 chip, this behaviour can be attributed to the OS's specific resource allocation, which ensures a consistent CPU usage (as per the Figure 7.28). Therefore, CPU usage might vary in different environments.

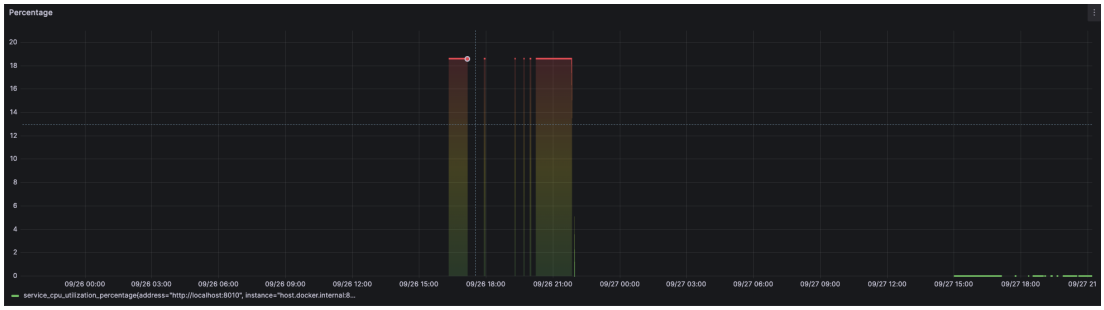


Figure 7.28: CPU Utilisation in Data Scientist Application

Memory Usage in the Data Scientist Application: Memory utilisation is stable at approximately 12.56 GB across all observations. This indicates that the application has effective memory management and doesn't experience memory leaks or significant fluctuations. This is depicted in Figure 7.29.

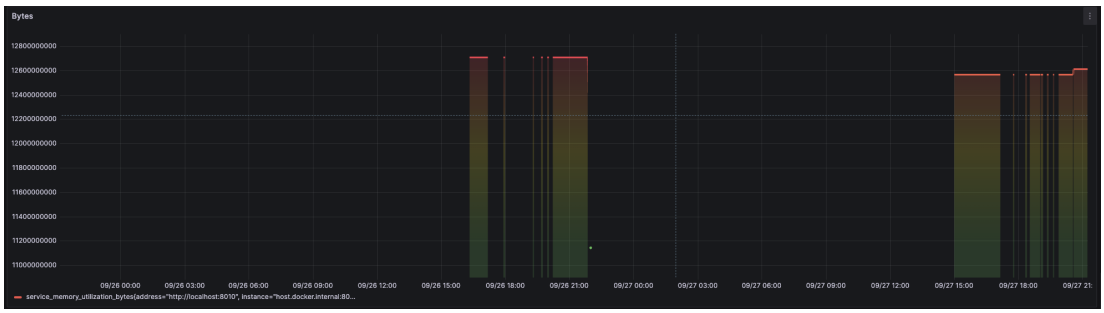


Figure 7.29: Memory Utilisation in Data Scientist Application

Query Processing Duration: The query processing duration remained predominantly at 10 seconds for the observed time. However, there is a drop to 2 seconds towards the end of the observation period (Figure 7.30). This sudden decrease suggests possible optimisations or changes in the nature of the processed queries.

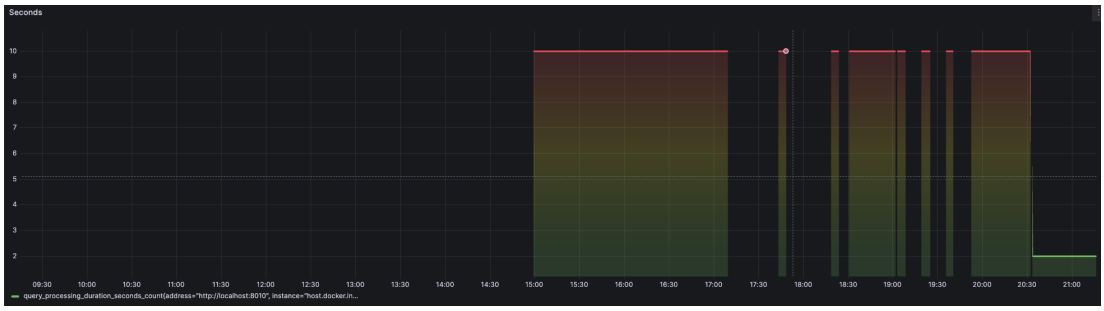


Figure 7.30: Query Processing Duration in Data Science Application

A key factor in the system's consistent performance is its domain-driven asynchronous and event-driven architecture, which aligns with P. Braun et al. (2021)'s findings on distributed systems. This design excels in managing concurrent data access and maintaining safe domain models in asynchronous conditions, themes echoed in Kleppmann's book *Designing Data-Intensive Applications* (Kleppmann, 2017). Kleppmann

(2017) emphasises the importance of data consistency and network resilience, crucial in modern distributed frameworks. The prototype's ability to handle concurrent updates and maintain data integrity highlights the efficacy of this architectural approach in distributed systems development.

7.4 Experiment Challenges and Duration

The experimentation phase spans approximately four months, characterised by a systematic approach to evaluate each scenario for the thorough validation of the Metamycelium system against stipulated requirements. The experiment encountered various architectural and technical challenges, which significantly inform the design choices and understanding of the system's behavior. These challenges are categorised into Infrastructure, Integration, Data Management, and Performance.

Infrastructure Challenges

1. The confluent Helm charts could not be used as they utilised an alpha version of a Kubernetes resource named PodResourceBudget. This resource was only in alpha until V1.22, while this experiment operated on version 1.27. Nginx Ingress also tried using the cloud manager to create a load balancer, resulting in the external IP of the ingress service perpetually pending. This occurred because the cluster was launched in a local dev environment, hence, no cloud service was available to deploy a load balancer like EKS or AKS.

2. Another infrastructure challenge arose with the Kafka pod, which repeatedly failed. Given Kafka's heavy reliance on file storage for various types, like segment files and log files, the originally allocated 64GB for the Kafka instance proved insufficient. The configuration was subsequently adjusted, enhancing the

cluster storage size to 300GB.

3. During development, Prometheus and Grafana required a host address for inter-communication. This was also true for the telemetry processing service and the Prometheus metrics scraper. However, since the prototype was developed within a single computation environment, much of the intercommunication depended on the host. This posed complexities, as the loopback address or localhost in each Docker container resolved to the container's loopback, not the host. To counter this issue, a distinct DNS was employed that referred to the host.

Integration Challenges

4. A significant hurdle involved the Keycloak helm chart's failure to connect to Postgress DB, even after explicitly setting a username and password. The cluster required multiple restarts. The eventual resolution involved a shift from Minikube to KIND.

5. After setting up the ingress, the task was to direct traffic accurately to various services. Most services had their own dedicated servers, necessitating the correct configuration of the reverse proxy to ensure upstream servers could respond appropriately. For instance, the Keycloak server had issues recognising specific host paths, anticipating a root path for serving files. Both the server and proxy needed adjustments for functionality. Similar issues arose for most services since numerous open-source software required configuration to serve their assets on paths other than the root.

6. Serving Vault UI behind an ingress proved problematic, as Vault-UI performs multiple redirections on various application paths, rendering rewriting or pass-through ineffective. This challenge prompted inquiries on major forums such

as Stackoverflow and Vault's GitHub repository. As of 02/07/2023, no viable solution appeared evident. The only conceivable approach involved assigning a domain to vault-ui and routing through a load balancer. However, since Vault-UI was predominantly a graphical interface to interact with Vault and not a central architectural component, the experiment's integrity remained intact.

7. Although Kiali was deployed as a elm chart, its underlying use of Prometheus for metrics and logs retrieval made the network connection intricate. Challenges arose due to networking complexities and Kubernetes' inherent service accounts.

8. Comprehending the external authoriser API employed by Envoy and activating it in a Kubernetes cluster's service mesh through a specific namespace was far from straightforward. Confusion proliferated regarding the service mesh objects' lifecycle, the admission reviews, and the admission controller.

Data Management Challenges

9. A principal challenge during the design of the data quantum revolved around determining the data resolution nature from the analytical service. With the usage of FastAPI, which is intrinsically event-driven, it is unwise to block the main thread while continuously requesting an HTTP endpoint for events. A solution involved a conditionally invoked function to fetch all events from a specific topic, either upon startup or via an endpoint.

10. Another ingress-related challenge arose from the local-host-based cluster development. To direct ingress traffic to the Kafka rest proxy, nginx was configured for target rewriting. While this approach suits ingress network requests, egress requests pose difficulties. For example, when the Kafka rest proxy API returned the consumer URL, it omitted the rewrite rules, necessitating custom code to modify the URL.

11. Managing the data flow for domain B (customer reviews domain) posed challenges due to the considerable size of Yelp academic datasets, some extending to 5GB. To facilitate version control and prevent server and stream timeouts, these JSON files required chunking. A custom Python script was devised to handle JSON file chunking, ensuring no file exceeded 99MB.

12. SQLite, utilised in analytical services to store various data forms, posed challenges with semi-structured data like JSON. The inconsistency in JSON data formats in the customer domain necessitated the creation of distinct tables for diverse semi-structured data sets. To tackle this, a custom function was developed to dynamically generate SQLite tables.

13. An observed inefficiency emerged when initially collecting metrics from various services using standard Python and Node APIs. It was soon discerned that numerous traces lacked essential metadata for proper visualisation, like service name, service version, and environment. This oversight necessitated a rerun of scenario one for optimal visualisation and narrative coherence.

14. A related issue involved the automatic deletion of stale Kafka consumers by the Kafka-rest-proxy service. This action resulted in service failures during subsequent Kafka requests. To mitigate this, a custom function was developed to rewrite the consumer and reestablish a connection to the REST server.

Performance Challenges

15. During the process of transferring operational data from the operational service to the analytical service, storage information was dispatched alongside the success event. This method introduced causality issues. Subsequent requests, aimed at processing this data and registering it to Datalichen, depended on the distributed

storage data. Addressing this necessitated the consideration of a centralised configuration pattern.

16. Psutil's inability to accurately depict CPU usage due to Apple's M2 chip architecture posed an additional performance challenge.

Despite these challenges, the system demonstrates capability in handling complex scenarios. However, the limited resources in terms of time and infrastructure restrict the scope of experimentation. The chosen scenarios provide a comprehensive understanding of the system's capabilities, but a more extensive investigation is feasible under more favourable conditions. The experiment, despite its limitations, reveals the robustness and potential of the Metamycelium prototype.

7.5 Conclusion

The rigorous testing of the Metamycelium system under various scenarios has conclusively shown its capability to meet and, in several instances, exceed the stipulated requirements. Based on the experimental findings, the following conclusions can be drawn:

1. **Volume**: The high volume data ingestion scenario (Scenario S1) demonstrates the system's capability in supporting both asynchronous and batch processing, meeting requirements **Vol-1** and **Vol-2**. The system provides scalable storage for large datasets, effectively handling increased data ingestion during peak times.

2. **Velocity**: Through Scenario S2, centered on high-velocity data ingestion, the system clearly illustrated its competence in handling a range of data transmission speeds, thereby meeting requirements **Vel-1** through **Vel-5**. The system's proficiency in streaming data timely to consumers, coupled with its capability for quick search and real-time data processing, is particularly impressive.

3. **Variety**: Although Scenario S3 is not directly tested, inferences from Scenarios S1 and S2 strongly suggest the system's ability to manage diverse data formats, from structured to unstructured data. This supports the claims of **Var-1** through **Var-4**. The system's intrinsic capability for data aggregation, standardisation, and schema evolution stands out as a significant strength.

4. **Value**: The Complex Query Scenario (Scenario S4) provides evidence of the system's computational prowess, affirming the requirements from **Val-1** to **Val-4**. The system's dexterity in supporting both batch and streaming analytical processing and its flexibility in handling multiple output formats set it apart. This is highlighted in the experiment, as each domain holds a different type of semi-structured data.

5. **Security & Privacy**: Scenarios S5 and S6, centered around secret management and data security respectively, robustly validate the system's dedication to meeting **SaP-1** and **SaP-2**. The strategic use of OpenID Connect and Hashicorp Vault ensures best practice protection, retention, and multi-level authentication mechanisms.

6. **Veracity**: The domain-centric nature of Metamycelium inherently backs requirements **Ver-1** and **Ver-2**, underscoring the importance of data quality curation and provenance.

In the next chapter, Metamycelium and the design theories discussed will be subject to expert opinions to point out strengths and limitations.

Chapter 8

Evaluation - Expert Opinion

8.1 Introduction

In the previous chapter, a prototype of Metamycelium is created and subjected to diverse simulations. In this chapter, expert review is employed as the second evaulation method of the artefact. For the purposes of this expert opinion gathering, the research method follows the guidelines of Kallio et al. (2016). This research methodology is discussed in Section 2.9.6.

The examination of expert perspectives begins with an analysis of emergent themes in "Thematic Analysis" Section 8.2, a comparative "Matrix of Responses" Section 8.3, and a "Quantitative Summary" Section 8.4 of the collated data. Reflective synthesis is presented in "Synthesis and Reflection" Section 8.5, with professional assessments of the architectural strategy in "Feedback and Reflections on Architectural Approach" Section 8.6. Detailed considerations of data strategy are explored in "Insights on Data Handling and Architectural Intent" Section 8.7, while "Nuances of Data Ownership, Privacy, and Governance" Section 8.8 assess the complex legalities involved. The chapter advances to "Relevance and Future Implications" Section 8.9, critiques in "Limitations of the Current Reference Architecture" Section 8.10, and concludes with "Technological Considerations and Expert Insights" Section 8.11, thus paving the way to the references section.

8.2 Thematic Analysis

In this section, prevalent themes from expert responses are methodically identified and analysed through a systematic coding and clustering process. This scrutiny not only highlights recurring ideas but also contextualises the breadth of perspectives that influence the Metamycelium RA.

Transcripts from expert opinions were thoroughly reviewed to identify recurring

ideas, perspectives, and insights. Key themes were then analysed in-depth, examining nuances, similarities, and differences within each theme, along with the underlying rationale and implications. These prevalent themes, organised and presented in a structured manner, enable a comprehensive exploration of the key areas of focus, concerns, and insights offered by the expert panel.

This thematic analysis not only highlights the recurring ideas but also situates them within the broader context, providing an understanding of the various perspectives on the Metamycelium RA.

8.2.1 Applicability & Industrial Relevance:

The domain-driven, distributed nature of the Metamycelium architecture prompts a range of opinions regarding its alignment with current industry practices and projected future trends. While this architectural approach encapsulates promising and advanced practices, it is evident that they aren't universally adopted across industries.

i2 shared a viewpoint that the application of Metamycelium might be influenced by team structures and organisational sizes. Furthermore, there was an indication that some regions may be more advanced in certain data practices, with only a minority embracing advanced techniques like microservices or streaming.

> *"In certain areas, the sense is that the artefact is slightly ahead of times with only a few diving into micro services."* - i2

i1 emphasised the architecture's capability to cover eventing, data streaming, and batch processing. Nevertheless, there were observations about potential gaps between prevailing industry trends and certain elements of the architecture.

> *"Industry trends might not align perfectly, but this is definitely industry relevant."* - i1

i3 expressed that the domain-driven nature of the architecture was particularly noteworthy. As industries lean towards more granular and distributed data ownership, challenges arise. Moreover, i3 disucssed reservations about the intricate nature of such an approach, suggesting that its complexity might pose adoption challenges.

> *"The architecture's complexity might challenge some organisations, even larger ones, during adoption."* - i3

Taking experts opinion into consideration, it is evident that Metamycelium offers potential avenues for many advanced data operations. However, its novelty and intricacy might be both an advantage and a hurdle. It could attract organisations looking for innovative solutions but might also be seen as daunting by those familiar with more traditional methods or those not yet prepared for such a shift in their data practices.

While the architecture's domain-driven distributed design holds considerable promise, its divergence from traditional practices could be perceived as a potential barrier to broader adoption. The challenge lies in aligning its advanced features with the current industrial readiness and landscape.

8.2.2 Strengths & Distinctive Features:

The Metamycelium architecture attracted commendation for its distinctive features and perceived strengths, particularly around the integration of domain-driven design and data engineering.

i1 underscored the appeal of a domain-driven approach combined with data engineering. Noting that traditionally, software engineering and data engineering have disparate goals and KPIs, i1 emphasised the significance of bridging this divide.

> *"Most of the time you were talking, you were putting software engineering and data engineering together. It sounds really good in theory... If you manage to solve that, that could be a good strength to have... teams working together, you don't have a person in the middle so you can move much faster."* - i1

This integration was further acknowledged as a novel approach, with i1 adding:

> *"I like that. That will be quite distinctive, because I'm yet to see that in action."* - i1

The architecture's flexible nature and comprehensive ecosystem also gained positive attention from another expert. i3 particularly praised its capability to encompass everything from batch eventing to ACLs, and the significance of having immutable logs (refers to immutability discussed in Section 6.4.13) to establish a single source of truth.

> *"It considers everything from batch eventing, ACLs to infrastructure. Immutable log is very important these days because it helps identify the single source of truth."* - i3

Furthermore, the self-provisioning capability of components within the architecture was deemed crucial.

> *"As a platform as a service, the fact that you can self provision components as you need to, is very important."* - i3

i2 highlighted the architecture's modular nature, emphasising the advantages of having a clear demarcation between different components, such as domain-driven design, data governance, and data ingestion.

> *"I love this modular approach... I love the fact that it also calls out domain driven design and a clear separation of data governance."* - i2

i1 also delved into its combination of different data views like data mesh and event-driven aspects, mentioning the variance from other approaches but appreciating its unique stance.

> *"I think bits and pieces come from a different view. Like you have a data mesh here, event-driven there... The approach of open policy agent looks promising."* - i1

In wrapping up, the overall sentiment suggests that while certain components of Metamycelium might find parallels in the industry, its holistic approach and integration of domain-driven design with data engineering make it a unique offering. The challenge, however, remains in ensuring this distinctiveness translates into practical advantages in the evolving landscape of data operations.

8.2.3 Challenges & Potential Barriers:

Several experts provided insights into potential challenges and barriers for the Metamycelium architecture. The concerns encompass a range of issues from skillset availability, the complexities inherent in data governance, and scalability concerns.

i1 identified a key concern regarding the skillset required to leverage the architecture, especially at the local level.

> *"Skill set, that would be the first one from the local environment. Because again, there would be very few data engineers who would be willing to go there. Decentralising is great, but it is usually very hard to do well, like all those governance platform libraries..."* - i1

The expert further touched upon the difficulties in governance at large-scale enterprises and potential political influences.

> *"But I'd be talking big and why big enterprises. There's always politics in it. And lots of advice."* - i1

i1 also emphasised the importance of flexibility in governance, especially from a software engineering perspective.

> *"From a data engineering point of view... your governance is more concretely defined. In Software, it is a bit more flexible; these two have different KPIs and that might be a bit of friction."* - i1

One specific challenge that i1 raised was related to the complexities that may arise due to evolving ownership patterns and dependencies.

> *"So one of the challenges I can see happening at some point as it evolves, is the ownership might become a bit more complicated... Now I have dependencies on A and B, I'm not aware of."* - i1

i2 elaborated on concerns related to the scale of data and domain specificity.

> *"Some companies have very large data sets and so moving to a domain-driven is kind of wall sometimes if they have slightly monolithic systems... This data is just so big, so hard to play with."* - i2

He also pointed out challenges related to the clear ownership of data domains.

> *"Yeah, where they already have this centralised kind of data warehouse and they don't know who owns what... and which domain should own, it because it is not clear."* - i2

A potential problem regarding the handling of unstructured data was also flagged.

> *"I'm not sure how it works with things like unstructured data... But when you really get into really, really unstructured stuff, like emails or something like that, data governance just becomes crazy."* - i2

i3 highlighted the difficulty that may arise due to the cross-functional nature of the architecture.

> *"This typically won't be owned by one team, because this is an ecosystem. You'll have database administrators, developers, infrastructure people, product people, service catalog people... I think the cross functionality of it all is what's difficult."* - i3

Furthermore, i3 indicated potential challenges for smaller organisations to implement the architecture.

> *"I think for small organisations, this is going to be quite possibly very difficult for them to implement. What's gonna happen is they'll just choose some cheaper version of this when you have not enough people to deliver this sort of stuff."* - i3

In summary, while Metamycelium showcases a number of strengths, it is not devoid of challenges. These barriers span from implementation specifics, cross-functional complexities, governance, and domain challenges, to skillset availability. Addressing these challenges requires both strategic and technical efforts to ensure the architecture's broad applicability and efficacy.

8.2.4 Additional Expert Perspectives

The expert opinion gathering sessions also surfaced additional perspectives and considerations that did not fit neatly into the primary thematic categories. These supplementary viewpoints are discussed in this section. Various experts expressed their opinions and perspectives on the Metamycelium architecture that fell outside the primary themes of applicability, strengths, and challenges.

i1 mentioned the potential allure of the architecture to certain demographics:

> *"I can see like a startup kind of person who would want to deploy it, because it is very intriguing, because it is challenging just in the right way."* - i1

i2 delved into the broader transformation in the industry toward viewing data as a first-class citizen, emphasising the rise of Customer Data Platforms (CDPs) and data-driven concepts:

> *"I think there was a huge move much towards data as a first class citizen... Data is a first class citizen. You've got to think about data first and the value that the data provide. And so I think that as an industry or multi industry, that transformation is really happening already."* - i2

i2 also pointed out a gap in data adoption and maturity, even among prominent IT companies:

> *"Yeah, data... directly like data adoption, data use. I mean, look at companies ourselves, we are being one of the most successful IT companies in this country and data maturity is not there. I haven't worked yet in a company that has matured data-driven culture, including some very big names that I've worked for in the past."* - i2

This section sheds light on a broader range of opinions and considerations about the Metamycelium architecture, capturing nuances and insights that may not fit neatly into other categories.

8.3 Matrix of Responses

Table 8.1 summarises the responses of each expert and provides a concise representation of their views on the applicability, strengths, and challenges of the Metamycelium architecture.

Table 8.1: Mapping of Expert Responses Against Themes

Expert	Applicability	Strengths	Challenges
i1	Mixed	Emphasises importance of flexibility in governance	Concerned about the skillset required, complexities of evolving ownership patterns, and large-scale enterprise politics
i2	Positive	Cites large data handling and domain specificity	Mentions challenges of clear data domain ownership and difficulties with unstructured data
i3	Positive	Stresses on the cross-functional nature and the ecosystem approach of the architecture	Highlights potential difficulties for smaller organisations and cross-functional challenges

The ‘Applicability’ column captures the experts’ assessments of how well the Metamycelium architecture aligns with current industry practices and future trends, indicating whether they see the architecture as having positive, mixed, or negative applicability in real-world business settings. The ‘Strengths’ column highlights the distinctive features and perceived advantages of the Metamycelium architecture, as identified by the experts, summarising their perspectives on the architecture’s innovative

aspects and potential benefits.

The ‘Challenges’ column outlines the potential barriers and issues that the experts foresee with the implementation and adoption of the Metamycelium architecture, covering concerns related to skills, governance, scalability, and other factors that could hinder the architecture’s practical application. This matrix allows for a quick, tabular comparison of the experts’ overall perspectives, providing a structured overview of the key themes discussed in the previous section.

8.4 Quantitative Summary

From the coding data, it is evident that opinions and discussions varied across different aspects of the Metamycelium architecture. The quantitative summary of these codings is displayed in Table 8.3.

The themes "Applicability and Industrial Relevance", "Challenges and Potential Barriers", and "Strength and Distinctive Features" have the highest number of coding references (9 references each), signifying their prominence in the expert discussions. Under "Applicability and Industrial Relevance", discussions on the "Alignment with Industry Trends and Future Projection" were predominant with 6 references. The theme "Challenges and Potential Barriers" saw equal prominence between "Adoption Challenges for Organisations" and scenarios where the architecture may not be the best fit. Moreover, "Other Opinions", which provided auxiliary views and considerations, had 3 references, making it a lesser-discussed theme. These data is visualised in Figure 8.1.

In addition, following each expert opinion gathering, experts were invited to review each category each category, as outlined in Section A.7, is prepared and intended for distribution to three experts. The survey structures questions on a likert scale ranging from ‘Strongly Disagree’ to ‘Strongly Agree’ The responses from the experts,

Table 8.3: Summary of coding references.

Theme	No. of References	No. of Items Coded
Applicability and Industrial Relevance	9	3
Alignment with Industry Trends and Future Projection	6	3
Applicability to Industry	3	3
Challenges and Potential Barriers	9	3
Adoption Challenges for Organisations	6	3
Scenarios Where The Architecture Is Not Best Fit	3	2
Other Opinions	3	2
Strength and Distinctive Features	9	3
Differentiating Components of the Architecture	3	3
Strengths of the Architecture	6	3

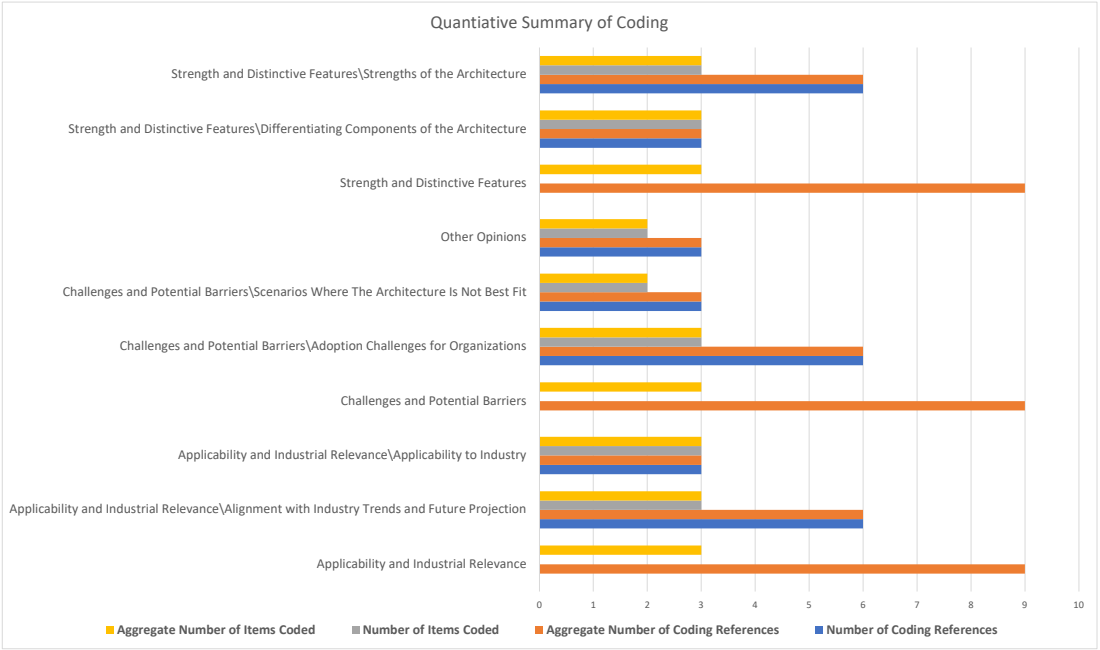


Figure 8.1: Quantitative Summary For Expert Opinion

showcasing diversity in their perspectives, are mapped to numerical values for concise representation in the table. The mappings is as follows:

- Strongly Agree: 2
- Agree: 1
- Neither Agree or Disagree: 0
- Disagree: -1
- Strongly Disagree: -2

These results are depicted in Table 8.5.

Table 8.5: Expert Opinions on the Metamycelium Architecture

Expert	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
I1	1	1	0	1	2	1	0	2	1	1
I2	2	2	2	2	2	1	-1	1	2	2
I3	-1	1	0	-1	-1	2	2	2	1	1

8.5 Synthesis and Reflection

The overarching sentiment from experts indicates an appreciation for the efficiency, scalability, and effectiveness of the proposed architectural approach. Across the board, from case experiments to expert feedbacks, there's a consistent affirmation of the architecture's potential.

Reflecting on these findings in the context of the research questions, it is evident that the expert opinions provide valuable insights into the viability, applicability, and potential impact of the Metamycelium architecture. The feedback from experts across different domains and industries suggests that the architecture has the potential to

address the challenges of BD systems and offer a more efficient and effective approach to data management and analytics.

Moreover, the expert feedbacks highlight the growing recognition of the need for more advanced, domain-driven, and data-centric architectures in the industry. This trend aligns with the broader shifts in the field towards more responsible and ethical data practices, as evidenced by the experts' emphasis on the importance of data governance, privacy, and ownership.

Situating the Metamycelium architecture within the existing literature reveals its contribution to the ongoing discourse on the design and development of BD systems. The architecture's unique combination of domain-driven design, event-driven communication, and modular components resonates with emerging trends and best practices in the field (Laigner et al., 2021a; Dehghani, 2022). This positions Metamycelium as a novel and promising approach to addressing the challenges of BD systems.

However, the expert feedbacks also underscore the importance of considering the sociotechnical dimensions of implementing such an advanced architectural approach in practice. The concerns raised regarding the complexity, skillset requirements, and organisational readiness align with the findings of previous research on the adoption and implementation of BD systems (H.-M. Chen, Kazman & Haziyev, 2016b; Mikalef et al., 2018). These challenges highlight the need for further research and development efforts to support the successful implementation of Metamycelium in real-world contexts.

8.6 Feedback and Reflections on Architectural Approach

Experts have endorsed the architecture's ability to align well with primary assumptions, handle varied scales of data, and deliver effective results in practical applications. The real test of any theoretical construct lies in its tangible impact. The successful execution of tasks using this architecture, and its efficacy in handling complex operations,

demonstrates that the design isn't just theoretically sound but also practically capable.

The experts' feedback highlights several key strengths of the architectural approach. The modular design of the architecture was praised for its flexibility and adaptability to various use cases and domain-specific requirements. This modularity allows for the seamless integration of components and enables the architecture to be customised to meet the unique needs of different industries and applications while maintaining overall consistency and interoperability.

Another aspect that received positive feedback was the architecture's focus on scalability and performance. The distributed nature of the architecture, along with its capability to handle high volumes and velocities of data, was recognised as a significant advantage. Experts acknowledged the criticality of these features in tackling the constantly growing demands of BD systems and ensuring their long-term viability and efficiency.

The incorporation of event-driven communication and real-time processing capabilities in the architecture was also well-received by the experts. They recognised the increasing importance of systems that can respond to data in real-time and make decisions based on streaming data. The Metamycelium architecture's support for these capabilities was seen as a crucial factor in its potential to foster innovation and enable new applications in the BD ecosystem.

However, it is also vital to note areas of concern or potential limitations. Some experts cautioned that the comprehensive scope of the architecture might introduce complexity in terms of implementation and management. The successful adoption of Metamycelium may necessitate substantial investments in resources, skills development, and organisational change management. Addressing these challenges will be crucial to ensure the smooth implementation and widespread adoption of the architecture.

Moreover, experts stressed the importance of continuous evolution and improvement of the architecture based on real-world feedback and performance metrics. Given

the rapid pace of change in the BD landscape, it is essential for the Metamycelium architecture to remain flexible and adaptable to emerging technologies, standards, and best practices. Regular updates and refinements based on user feedback and real-world performance data will be vital to maintain the architecture's relevance and effectiveness over time.

Given the nascent stage of the architecture and promising results, there's potential for evolution. If more organisations adopt and implement the Metamycelium architecture, valuable insights and lessons learned will emerge. Leveraging this feedback and real-world performance analytics will be crucial to identify areas for improvement, optimise the architecture, and ensure its continued relevance in the face of evolving BD challenges and opportunities.

8.7 Insights on Data Handling and Architectural Intent

Challenges like intricate data dependencies, ownership issues, and the fluid nature of data ecosystems are highlighted. Aspects of Metamycelium like immutability and bi-temporality are discussed to address data evolution and dependencies. The conversations brought forth the importance of understanding the role of domains, clarifying architectural intent, and ensuring feedback loops for data quality. The balance between analytical and operational intents and their influence on interfacing with business processes are also explored.

The experts emphasised the need for clear data lineage and provenance tracking to understand how data evolves and flows through the system. Techniques like data versioning and change data capture were suggested to manage data updates and maintain a historical record of data changes.

The role of domains in the architecture was another focal point of the discussions. Experts highlighted the benefits of aligning architectural components with business

domains, such as improved modularity, encapsulation, and governance. They also stressed the importance of defining clear domain boundaries, ubiquitous language and interfaces to facilitate loose coupling and independent evolution of domains.

Regarding architectural intent, the experts emphasised the need to strike a balance between analytical and operational requirements. They discussed the trade-offs involved in optimising for query performance versus real-time processing and the impact on data storage and processing choices. The importance of designing flexible and adaptable data models that can cater to both analytical and operational use cases was also highlighted.

The conversations also touched upon the importance of data quality and the role of feedback loops in ensuring data integrity and reliability. Experts suggested incorporating data validation, cleansing, and enrichment processes as part of the data ingestion pipeline. They also recommended implementing monitoring and alerting mechanisms to detect and respond to data quality issues in real-time.

Another aspect explored was the integration of the Metamycelium architecture with existing business processes and systems. The experts emphasised the need for well-defined APIs and integration patterns to enable seamless data exchange and interoperability. They also discussed the potential benefits of event-driven architectures in facilitating real-time data propagation and triggering business workflows based on data changes.

Overall, the insights gathered from the experts provide valuable guidance on data handling and architectural intent within the Metamycelium architecture. By addressing data dependencies, aligning with domain-driven design principles, balancing analytical and operational requirements, ensuring data quality, and seamlessly integrating with business processes, the architecture can effectively support the diverse needs of modern data-driven organisations.

8.8 Nuances of Data Ownership, Privacy, and Governance

As products or domains evolve, they can lead to intricate dependencies. Techniques like streaming between domains can ensure up-to-date data flow. The conversations with the experts emphasised data privacy, the potential of synthetic data, and the importance of a governance layer in the architecture. Customised data governance approaches and evolving governance requirements, especially with new types of data, necessitate constant vigilance.

Moreover, there are various discussion in regards to applying governance policies to unstructured data. This is particularly bold when Rego (the policy language) and OPA are discussed. Some of the challenges discusses pivot on the fact that OPA is mostly supporting policies in regards to security and privacy of API endpoints, service networks and infrastructure, while data loads may introduce unstructured data types such as image.

Along the same lines, the experts delved into the complexities of data ownership and the implications of evolving domain dependencies. As the system grows and new data sources are integrated, the ownership and accountability of data may become ambiguous. Experts suggested establishing clear data ownership policies and defining roles and responsibilities for data stewardship within each domain.

Data privacy emerged as a critical concern during the discussions. Experts emphasised the need for robust data protection mechanisms, such as data encryption, access controls, and data anonymisation techniques. The potential of using synthetic data, which mimics the statistical properties of real data without exposing sensitive information, was also explored as a means to balance data utility and privacy.

The importance of a dedicated governance layer within the Metamycelium architecture was strongly emphasised. Experts highlighted the need for flexible and

customisable governance frameworks that can adapt to the specific requirements of different domains and data types. They suggested incorporating policy-driven access control, data lineage tracking, and audit trails to ensure compliance with data regulations and internal policies.

The discussions also touched upon the challenges of governing unstructured data, such as images, videos, and free-form text. Experts acknowledged the limitations of existing policy languages, like Rego, and policy engines, like OPA, in handling unstructured data. They highlighted the need for advanced techniques, such as natural language processing and computer vision, to extract relevant metadata and apply appropriate governance policies to unstructured data.

Moreover, there has been emphasis on the importance of continuous monitoring and auditing of data governance practices. As new data types and sources are introduced, governance requirements may evolve, necessitating regular reviews and updates to policies and procedures. Automated compliance checks and alerts can help identify potential breaches and ensure timely corrective actions.

The experts also briefly explored the potential role of Large Language Models (LLMs) in aiding with the governance of unstructured data, such as automatically generating metadata or classifying sensitive information, but noted that further research is needed to assess the feasibility and reliability of such approaches.

8.9 Relevance and Future Implications

It is paramount to ensure any new architectural approach aligns with industry practices and requirements. Metamycelium's architectural approach, while promising, needs to continually refine based on insights from experts, ensuring its robustness, adaptability, and relevance in the ever-evolving landscape of computational design.

The experts emphasised the importance of regularly assessing the alignment of the

Metamycelium architecture with industry trends and best practices. As the field of BD and computational design continues to evolve rapidly, it is crucial to stay informed about emerging technologies, paradigms, and patterns. Regularly engaging with industry experts, participating in conferences and workshops, and monitoring relevant publications can help ensure that the architecture remains up-to-date and relevant.

The insights gathered from experts provide valuable feedback and perspectives that can guide the refinement and evolution of the Metamycelium architecture. It is essential to carefully analyse and incorporate these insights into the architectural design and implementation roadmap. This may involve adjusting existing components, introducing new features, or refactoring certain aspects of the architecture to better align with industry requirements and expectations. Experts also highlighted the importance of continuous testing and validation of the architecture in real-world scenarios. As the architecture is applied to different use cases and domains, it is crucial to gather empirical evidence of its effectiveness, scalability, and performance. Conducting case studies, pilot projects, and performance benchmarking can help validate the architecture's capabilities and identify areas for improvement.

Another key aspect emphasised by the experts is the need for the architecture to be adaptable and extensible. As new technologies and frameworks emerge, the architecture should be able to accommodate and integrate them seamlessly. This requires a modular and loosely coupled design that allows for easy extension and customisation. By providing well-defined interfaces and abstraction layers, the architecture can enable the incorporation of new components and technologies without significant disruption to existing systems.

The experts also stressed the importance of fostering a strong community and ecosystem around the Metamycelium architecture. Engaging with developers, data engineers, and researchers who are actively working with the architecture can provide valuable insights, contributions, and support. Encouraging open-source collaboration,

sharing knowledge through documentation and tutorials, and establishing forums for discussion and feedback can help build a vibrant community that drives the continuous improvement and adoption of the architecture.

8.10 Limitations of the Current RA

A significant barrier to the adoption of Metamycelium is its inherent complexity, necessitating specialised skills and understanding that may not be readily accessible, particularly in organisations with limited resources or expertise. This challenge is compounded by the absence of open-source technologies to support specific architectural constructs, like Data Lichen. Despite the acknowledged robustness of its design, there are apprehensions about the sociotechnical dimensions. Aligning technology with human and organisational dynamics is crucial for Metamycelium's successful implementation and adoption.

Additionally, not all organisations operate on a microservices architecture or adopt a fully domain-driven approach. Despite the popularity of these concepts, many organisations still rely on legacy systems built with Java, PHP, and Fortran, which may not seamlessly integrate with or support the advanced paradigms proposed by Metamycelium.

The complexity of Metamycelium is further intensified by the sheer number of components within its architecture. Each component, while integral to the system's overall functionality, adds to the complexity of implementation. This multifaceted nature demands a high level of coordination and technical expertise, making the process challenging, especially for organisations with constrained technical capabilities or those unfamiliar with such intricate architectures.

Moreover, the implementation of Metamycelium's comprehensive suite of components, such as IAM and secret management, presents substantial cost and time

implications for organisations. To fully deploy all aspects of Metamycelium, an organisation would need to invest significantly in resources and manpower. Additionally, the necessity to integrate a variety of open-source technologies, which may not always have compatible interfaces, adds another layer of difficulty.

This not only extends the time and financial investment required but also necessitates a strategic approach to select and harmonise these technologies effectively within the existing organisational infrastructure.

8.11 Technological Considerations and Expert Insights

During the discussions, specific technological solutions are highlighted. The utilisation of technologies like OPA and the idea of connecting IAM systems to Kafka is discussed. These insights are particularly valuable, given that two of the experts hailed from some of the world's leading data and eventing solutions companies. Their perspectives shed light on possible integrations and optimisations that could further enhance the architecture's robustness and applicability.

In one session, an event-driven expert mentioned that his team has been receiving a lot of requests for native integration of IAM systems into their Kafka solution. He was impressed to see that Metamycelium is also working on achieving the same functionality. Nevertheless, the expert accepted the current challenges with asynchronous authentication and authorizaation systems.

In another session, an expert with a lot of experience in consulting for batch and stream processing BD systems, shared the insight that the industry is currently trying to adopt the idea of 'stream only', even though the phrase streaming is not that well understood and sometimes 'micro-batch' processing is perceived as streaming.

An expert from a leading BD solutions provider highlighted the industry's shift towards integrating stream and batch processing within a unified architectural framework,

akin to the Kappa architecture. This approach, prevalent among major BD solutions, offers a unified data processing paradigm, enhancing efficiency and ensuring consistency across various data workloads. It facilitates resource optimisation by accommodating both stream and batch data within the same infrastructure, and supports advanced analytics and machine learning by enabling seamless integration of historical and real-time data analysis.

Contrastingly, at Metamycelium, a different strategy is employed that focuses on segregating computation workloads. However, the industry trend favours a singular construct for both stream and batch processing, underscoring the move towards more integrated, scalable, and flexible data processing architectures in response to the evolving complexities of BD.

Building on the previous discussion, a further point of interest emerged during the dialogue between the expert and the researcher. They delved into the concept of a phased implementation strategy employed by companies in the realm of BD solutions. Initially, many organisations opt for a singular architectural construct that seamlessly integrates both batch and stream processing. This unified approach, as the expert elucidated, serves as an effective starting point for companies due to its efficiency and the streamlined management of diverse data types.

As these companies mature and their computational needs become more complex, they often transition towards a bifurcated architecture. This evolution involves separating the batch and stream processing into distinct constructs. Such a transition is driven by the need for more specialised, high-performance processing capabilities that cater to the distinct characteristics and demands of batch versus real-time data streams.

The phased approach, therefore, offers a scalable and adaptable framework, allowing organisations to evolve their data processing architectures in alignment with their growth and the increasingly intricate nature of their data processing requirements. This strategy underscores the dynamic nature of BD solutions architecture, adapting to the changing

scale and sophistication of organisational needs.

In ongoing discussions, an expert with experience in DNA data highlighted privacy concerns related to handling such sensitive information. The expert queried about the role of OPA in addressing these issues. In this context, the Metamycelium architecture's federated computational governance is identified as a relevant feature. This component provides a framework for data governance in distributed environments, essential for managing the privacy of DNA data.

Additionally, the architecture's egress component is noted as crucial. It manages secure data outflow, ensuring compliance with privacy regulations and organisational policies, particularly vital for sensitive data management. Therefore, the combination of the federation computational governance and the egress component in Metamycelium could be instrumental in mitigating privacy and security challenges in DNA data handling.

Continuing on the topic, the same expert discussed the integration challenges of Secure Multi-party Computation (SMPC) within domain-driven designs like Metamycelium. Both the researcher and the expert acknowledged the limitations in implementing SMPC due to Metamycelium's decentralised data ownership. They noted complexities in data sharing and computation, the difficulty of establishing inter-domain trust and security, and ensuring consistent data governance and compliance in a distributed environment. They also highlighted the potential issues of performance and scalability, the need for integration and interoperability among diverse systems, and the delicate balance between domain autonomy and the collective goals of SMPC. This discussion underscored the need for specialised expertise and resource allocation to effectively implement SMPC in a decentralised architecture like Metamycelium.

8.12 Conclusion

The Metamycelium RA merges domain-driven design with event-driven communication, representing a contribution to BD RAs. It incorporates features like Data Lichen, bitemporality, and immutability, enhancing its applicability and efficacy in contemporary data management challenges. This design demonstrates an adaptive approach to the evolving landscape of data processing.

The architecture has received favourable feedback from a number of experts, including seasoned professionals from leading data and eventing solutions companies. Their recognition, particularly regarding the architecture's integration of domain-driven and event-driven paradigms, highlights its potential in advancing data operations.

Nevertheless, the Metamycelium architecture faces challenges, notably its complexity. This complexity, while reflecting the architecture's comprehensiveness, also presents a challenge. It necessitates significant investment in skill development and training, which may be a barrier for organisations lacking resources or reluctant to undertake such a transformation. The lack of open-source support for components like Data Lichen adds to this challenge.

Furthermore, the architecture's technical merits, while widely acknowledged for robustness and innovation, do not overshadow the sociotechnical aspects that need consideration. The interaction between the technological framework and the human and organisational elements is critical. Effective implementation of the Metamycelium architecture requires not just technical expertise but also a deep understanding of organisational culture, team dynamics, and change management strategies.

The forthcoming chapter, serving as the discussion segment of this thesis, delves into a comprehensive analysis and synthesis of the research findings. It aims to contextualise these findings within the broader academic and practical landscape of BD systems.

Chapter 9

Discussion

9.1 Introduction

The preceding chapter presented an evaluation of Metamycelium, drawing upon expertise from varied backgrounds. This chapter serves as a scholarly conduit, where the initial findings of the study are integrated into broader discourses encompassing practice, operational implications, and future research trajectories in the field of BD systems.

This research embarked on an analytical journey to address architectural challenges in BD projects. It involved a thorough examination of existing literature, underscoring the ubiquity and importance of BD systems. This exploration led to the formulation of Metamycelium, a domain-driven distributed RA designed to tackle prevalent challenges within the industry.

The foundational queries, as delineated in Section 2.3, guided the research's focus and trajectory. These questions emphasised the necessity to identify and address deep-rooted architectural limitations and to propose solutions. This discussion chapter aims to integrate the findings into a comprehensive framework of academic and practical dialogue.

The discussion commences with an examination of the research's "Key Insights and Observations" in Section 9.2, synthesising the acquired understandings and offering a comprehensive overview of the research process and challenges encountered. This is followed by an "Analysis and Interpretation of Findings" in Section 9.3, where results from the case mechanism experiment and expert opinions are scrutinised, contextualised within existing literature, and compared to other RAs.

The chapter then delves into "Implications for Practice and Fields of Research" in Section 9.4, discussing the practical applications of the research outcomes and providing insights for practitioners and stakeholders in the realm of BD systems. Section 9.5 addresses the "Generalisability and Transferability" of the findings, focusing on their

broader applicability and relevance across various contexts. Finally, the "Research Process and Insights" are explored in Section 9.6, highlighting the technical complexities encountered, the importance of effective communication within academic and industry settings, and the challenges associated with developing a technically advanced artefact.

9.2 Key Insights and Observations

This thesis presents Metamycelium, a RA for BD systems, developed through a research process involving SLRs, expert consultations, and a case mechanism experiment. The findings are analysed, engaging diverse viewpoints, theories, and methodologies to discuss the implications of Metamycelium in the context of current BD architectures like Kappa and Centralised Data Warehouse and emerging BD architectures such as Data Mesh and Data Fabric.

Metamycelium's domain-driven, decentralised approach represents a departure from the prevalent monolithic data pipeline architectures that dominate the BD landscape (Gorton & Klein, 2015; Nadal, Herrero, Romero, Abelló et al., 2017). Analogous to the principles of Data Mesh, which advocates for a decentralised data ownership model (Dehghani, 2022), Metamycelium aims to address the limitations of centralised architectures by empowering domains to manage their own data, fostering improved data quality, agility, and contextual relevance. However, the feasibility and implications of such a decentralised approach must be critically assessed, considering factors such as organisational readiness, technical expertise, and the potential for data silos (Serra, 2024b).

Metamycelium's domain-driven design is in-line with emergent data architectures such as Data Mesh, nevertheless, not all organisations have the readiness for such approaches.

The integration of microservices patterns and event-driven communication in Metamycelium is comparable to the concept of Data Fabric, which emphasises the importance of a unified, flexible, and scalable data architecture (Hechler et al., 2023). By leveraging these software engineering principles, Metamycelium aims to enhance the maintainability, scalability, and adaptability of BD systems (Chapter 6). However, the introduction of such complexities requires an evaluation of the trade-offs between the benefits of a decentralised architecture and the associated implementation and maintenance challenges, particularly for organisations with limited resources or rigid structures.

Integration of event-driven communication and microservices patterns may promote a more scalable and maintainble data architecture, but at the cost of increased complexity.

Metamycelium's emphasis on cross-cutting concerns, such as data quality, metadata management, and security, aligns with the principles of Data Fabric, which stresses the importance of consistent data management and governance across heterogeneous data environments (Hechler et al., 2023). Components like *Data Lichen* and the incorporation of constructs like the ingress and egress gateway demonstrate Metamycelium's commitment to addressing these critical aspects (Chapter 7). However, the reliance on custom-developed components raises questions about the sustainability and scalability of such solutions in the face of evolving BD ecosystems and the emergence of new technologies and standards.

The development of Metamycelium involved the creation of proprietary components like data lichen. While this addresses the requirements of the architecture, it may introduce difficulties for its adoption

The evaluation phase, comprising a case mechanism experiment and expert consultations, provided insights into the applicability, strengths, and limitations of Metamycelium. While the case mechanism experiment (Chapter 7) demonstrated the architecture's capability to handle high data volumes, velocities, and varieties, it also highlighted challenges related to data dependencies, system complexity, and the need for specialised expertise. These findings suggest the need to evaluate the generalisability of the experimental results to real-world BD environments and consider the impact of resource constraints and organisational dynamics on the successful adoption of Metamycelium.

While the artefact has managed to provide good results from the case mechanism experiments and expert opinions, it can benefit from a longitudinal empirical study that tests the architecture in various real-world organisataions.

Expert consultations (Chapter 8) revealed diverse perspectives on Metamycelium's industrial relevance, distinctive features, and potential adoption barriers. While acknowledging the architecture's strengths, experts also raised concerns regarding organisational readiness, skill availability, and the complexities associated with a decentralised, domain-driven BD architecture. These insights suggest the need for an examination of the socio-technical factors influencing the successful implementation of BD architectures and the development of strategies to address potential adoption barriers.

Future work should look more into the socio-technical aspects of adopting an architecture like Metamycelium.

Moreover, given that the majority of current BD architectures are designed underlying centralised architectures, as discussed in Chapter 4 and considering the ecosystem of technologies and tools that are built to support these architectures (discussed in Chapter 6), one has to think about migration challenges. While the idea of a decentralised

and domain-driven BD architecture may sound interesting and useful to companies, the cost of creating a new culture, shifting to new technologies, and abandoning widely accepted organisational practices may outweigh the benefits.

This can be analogous to companies that are moving away from monolithic software architectures into microservices architecture and the myriad of patterns that have been created to facilitate this process, as discussed in Chapter 4. Given this analogy, how many patterns exist to help companies move away from centralised BD architecture into emergent architectures like Metamycelium, Data Mesh or Data Fabric?

Given that currently most big data architectures are centralised, there is a need to create integration patterns to help companies transition to emergent distributed data architectures.

This raises another important point. Organisations that are not ready to absorb the complexity of distributed systems may not fully leverage the benefits of architectures like Metamycelium. Therefore, there is a need for a certain level of maturity. That is maturity in technology, management, and culture.

Organisations that are not mature enough to absorb the complexity of decentralised big data architectures, may not harness the benefits of it.

Another area of interest can be the use of Metamycelium for new AI applications using LLMs. Metamycelium's distributed nature and ability to scale horizontally can be advantageous in meeting the demands required by LLMs. LLMs heavily rely on vast amounts of training data, which need to be processed efficiently to ensure optimal model performance (Kasneci et al., 2023). Metamycelium's decentralised architecture can facilitate the distributed processing of training data, enabling faster and more scalable data preprocessing pipelines.

Furthermore, the architecture's emphasis on data quality and metadata management can contribute to the development of high-quality training datasets, which are crucial for the performance and generalisation capabilities of LLMs. During the model training phase, Metamycelium's ability to handle high data volumes and velocities can support the efficient training of LLMs, which often require multiple iterations over large datasets. Overall, BD architectures like Metamycelium can enable the development and training of LLMs by providing the necessary infrastructure and data management capabilities.

Big data architectures like metamycelium can potentially enable the development and training of large language models in an efficient way.

Reflecting on the research process, several limitations and areas for future exploration emerged. The prototyping and evaluation phases were constrained by limited resources and the challenges of replicating a real-world BD system environment. Additionally, while the research leveraged established methodologies, such as DSR and empirically grounded RA development (Chapter 2), the integration of other evaluation methods of evidence-based software engineering (Kitchenham et al., 2015) could further strengthen the assessment of Metamycelium's performance and scalability characteristics. Future research should examine the methodological choices made in this study and explore opportunities for enhancing the rigour and generalisability of the findings.

Metamycelium could be evaluated further using different methods of evidence-based software engineering.

Future research directions could include investigating the practical implementation and integration of Metamycelium in diverse organisational contexts, assessing its efficacy in handling emerging data formats and processing paradigms, and refining the architecture based on real-world performance metrics and evolving industry trends. The

development of guidelines and best practices for adopting and maintaining decentralised, domain-driven BD architectures, aligned with the principles of Metamycelium, could facilitate wider adoption and address the skill and expertise challenges identified in this research (Dehghani, 2022; Hechler et al., 2023).

Documentation of patterns, guidelines, and best practices for adopting Metamycelium can facilitate wider adoption of the artefact.

Metamycelium represents a contribution to the field of BD architectures, offering a domain-driven, decentralised approach to address the limitations of traditional centralised models. While recognising the complexities and challenges associated with such an approach, this research suggests the need for further exploration and refinement of decentralised BD architectures, aligning with the evolving demands of data-driven organisations and the increasing scale and complexity of data processing requirements.

It is important to continue examining the implications of decentralised architectures, such as Metamycelium, in the context of emerging paradigms like Data Mesh and Data Fabric, fostering a scholarly discourse that challenges assumptions, identifies limitations, and proposes solutions to the challenges faced by modern BD systems.

Continuing the scholarly discourse on decentralised BD architectures like Metamycelium is essential for identifying limitations and proposing solutions in the context of emerging paradigms such as Data Mesh and Data Fabric.

As the field of big data continues to evolve alongside advancements in AI, ongoing research, experimentation, and collaboration among academia and industry will be vital in shaping the future of decentralised BD architectures.

9.3 Analysis and Interpretation of Findings

The case mechanism experiment and expert opinions offer complementary perspectives on the Metamycelium architecture, providing a nuanced understanding of its strengths and potential areas for improvement.

The case mechanism experiment provides quantifiable results, demonstrating the architecture's ability to handle diverse scenarios related to data volume, velocity, variety, security, and privacy. For example, scenario 2, focusing on high-velocity data ingestion, showcases Metamycelium's capability to process a continuous stream of data without significant performance degradation. This aligns with Expert i2's positive feedback on the architecture's scalability and data handling efficiency, suggesting a convergence between empirical results and expert opinion.

However, the case mechanism experiment also reveals potential challenges and limitations. The complexity of data dependencies and ownership issues, as well as the resource-intensive nature of certain scenarios, highlight areas where Metamycelium may require further optimisation and refinement. These findings resonate with the concerns raised by experts regarding the adoption challenges for organisations, particularly in terms of skillset requirements and the intricacies of implementing a decentralised architecture.

Expert opinions provide valuable qualitative insights, shedding light on Metamycelium's perceived strengths and distinctive features. The architecture's domain-driven approach, emphasis on data quality, and integration of event-driven communication are identified as key strengths. These features are seen as potential differentiators, setting Metamycelium apart from existing solutions in the BD landscape.

Nevertheless, experts also highlight potential barriers to adoption, such as the complexity of the architecture, the need for specialised skills, and the challenges associated with organisational change management. For example, proprietary components such

as Data Lichen do not have an open source equivalent in the industry, which makes it harder for companies to adopt Metamycelium. Furthermore, Metamycelium does not only introduce an architectural change; it can potentially introduce an operation change to the companies, which can be perceived as costly.

These concerns underscore the importance of considering the socio-technical dimensions of implementing a decentralised architecture like Metamycelium, beyond its technical merits.

9.3.1 Contextualising the Findings:

To situate the findings from the case mechanism experiment and expert opinions within the broader context of BD architectures, it is essential to consider the current state of practice and academia.

The limitations identified in the case mechanism experiment, such as the complexity of data dependencies and ownership issues, align with the challenges faced by organisations in managing and processing large-scale, heterogeneous data using traditional monolithic data pipelines (Gorton & Klein, 2015; Nadal, Herrero, Romero, Abelló et al., 2017). The emergence of decentralised architectures, such as Data Mesh (Dehghani, 2022) and Data Fabric (Hechler et al., 2023), reflects a growing recognition of the need for more flexible, scalable, and domain-driven approaches to BD management.

Metamycelium's domain-driven, decentralised approach resonates with these emerging paradigms, offering a potential solution to the limitations of centralised architectures. The positive feedback from experts regarding Metamycelium's alignment with industry trends and its potential to address current challenges in BD management supports this notion.

However, the concerns raised by experts about the complexity and adoption challenges associated with Metamycelium also find parallels in the literature. Much

like the challenges experienced during the industry's transition from monolithic software architectures to microservices, implementing decentralised BD architectures like Metamycelium requires significant organisational and cultural changes, as well as investments in skills development and technology infrastructure (Hechler et al., 2023). These challenges are not unique to Metamycelium but reflect the broader difficulties faced by organisations in transitioning from traditional data management practices to more agile, domain-driven approaches.

Further, the SLR on BD RAs (Chapter 4) underscores the need for more research on domain-driven, decentralised approaches, as well as the importance of addressing cross-cutting concerns such as data quality, metadata management, and security. Metamycelium's emphasis on these aspects aligns with the identified research gaps and priorities in the field, indicating its potential to bridge the gap between academic research and industry practice.

However, some of the limitations of the study, such as the controlled nature of the case mechanism experiment and the potential biases in expert opinions, underscore the need for further empirical validation and investigation. Future research should aim to address these limitations by conducting more diverse and robust evaluations of Metamycelium, such as multiple case studies, longitudinal studies, and quantitative assessments of its performance and scalability in real-world settings.

Contextualising the findings from the case mechanism experiment and expert opinions within the current state of practice and academia reveals the relevance and potential contributions of Metamycelium to the field of BD architectures. While the architecture aligns with emerging trends and addresses identified challenges, it also faces adoption barriers and requires further empirical validation. Addressing these aspects through ongoing research and real-world implementations will be crucial in establishing Metamycelium's position as a viable solution for organisations seeking to harness the power of BD effectively.

9.3.2 Comparative Analysis:

The case mechanism experiment provides quantifiable results, while expert opinions offer a qualitative perspective, adding depth to the evaluation. For example, the experiments showed success rates, and the expert opinions highlighted the "cross-functional nature and the ecosystem approach" as one of the architecture's core strengths. Such qualitative insights provide context that numbers alone might miss. However, it is essential to critically examine the alignment between these two evaluation methods and assess the validity of the conclusions drawn from their comparison.

The quantitative results from the case mechanism experiment demonstrate Metamycelium's ability to handle various scenarios related to data volume, velocity, variety, security, and privacy. These findings suggest the architecture's potential to address key challenges in BD systems. However, it is crucial to acknowledge that these experiments were conducted in a controlled environment, which may not fully reflect the complexities and constraints of real-world implementations (Mikalef et al., 2018). Moreover, the case mechanism experiment, while valuable, may not fully capture the nuances of real-world implementations where factors like organisational culture, legacy systems, and data governance practices could influence the architecture's performance.

On the other hand, the qualitative insights from expert opinions provide a more nuanced understanding of Metamycelium's potential benefits and challenges, such as its "domain-driven nature and data as first class citizen approach", which are not directly captured by the quantitative metrics of the case mechanism experiment. Nevertheless, expert opinions are subject to individual biases and may not represent a comprehensive view of the architecture's applicability across different domains and industries (Creswell et al., 2007). It is important to consider the potential for confirmation bias in expert opinions, where experts may be predisposed to certain architectural styles.

Both methodologies identified challenges such as data dependencies and ownership

issues. Utilising a formal logical framework, let (P) represent the success rate in the experiment and (Q) the positive feedback from experts. If $(P \wedge Q)$ is true, as the findings suggest, the conjunction indicates a robust validation of the architecture's strengths and its applicability in real-world scenarios.

Specifically:

- The high volume data ingestion focus of scenario S1 aligns with expert feedback on the architecture's scalability and data handling efficiency ($E_{\text{LargeScale}}$). This correlation ($S1 \rightarrow E_{\text{LargeScale}}$) affirms the architecture's capability in managing large-scale data operations. However, this correlation should be interpreted with caution, as the controlled environment of the experiment may not fully capture the challenges and constraints of real-world large-scale data ingestion (Gorton & Klein, 2015). Further investigation is needed to assess the scalability of Metamycelium under varying real-world conditions, such as heterogeneous data requirements for training LLMs.

- Scenarios S4, S5, and S6, which centred on complex query processing and data security, correlate with expert observations on the architecture's strength in data governance and innovative security management ($E_{\text{PrivacySecurity}}$). This effective data integrity management in these scenarios supports the experts' positive views, leading to ($S4S5S6 \rightarrow E_{\text{PrivacySecurity}}$). Nevertheless, the logical implication should be further validated through real-world implementations and case studies to assess its robustness and generalisability (Nadal, Herrero, Romero, Abelló et al., 2017). A comprehensive security assessment, including penetration testing and vulnerability analysis, could provide additional evidence for the architecture's resilience against potential threats.

- The challenges identified in scenarios, particularly those related to data dependencies and system complexity, mirror expert concerns ($E_{\text{Complexity}}$). The

difficulties encountered in these scenarios ($S_{\text{Challenges}}$) resonate with the expert feedback on potential adoption barriers due to the system's intricate nature, thus ($S_{\text{Challenges}} \rightarrow E_{\text{Complexity}}$). This alignment highlights the need to address the socio-technical aspects of implementing a decentralised architecture like Metamycelium (Hechler et al., 2023). Strategies such as developing comprehensive documentation, providing training programmes, and establishing clear governance frameworks could help mitigate these challenges and facilitate the adoption of Metamycelium in diverse organisational contexts.

The comparative analysis of the case mechanism experiment and expert opinions offers valuable insights into Metamycelium's strengths and potential challenges. However, a critical examination of the alignment between the two evaluation methods reveals the need for further validation and investigation to establish the robustness and generalisability of the findings. By addressing the limitations of the current analysis and conducting more comprehensive evaluations, future research can contribute to a deeper understanding of Metamycelium's potential as a domain-driven, decentralised architecture for BD systems.

9.4 Implications for Practice and Fields of Research

This study systematically investigated BD RAs, identifying gaps and opportunities within both industry and research. The review in Chapter 4 reveals a predominance of monolithic and centralised BD RAs, raising issues of scalability, data quality, and interoperability. Metamycelium, with its emphasis on decentralisation, domain-driven design, and microservices, represents a significant shift away from these traditional models. In the following subsections, the implications of this study for practice and fields of research are depicted.

9.4.1 Implications for Practice

The architectural principles underpinning Metamycelium align with several emerging trends in the industry. The Thoughtworks Technology Radar (Thoughtworks, 2024), for instance, highlights a growing momentum towards decentralised data architectures and platforms, such as DataOS. The Radar provides insights into emerging technologies and trends based on the experiences and observations of their technology experts.

This resonates with Metamycelium's focus on data as first-class entities and federated data access across disparate sources. Along the same lines, the emphasis on domain-driven design, where data ownership and management are decentralised to the domain level, aligns closely with the principles of data mesh (Dehghani, 2022). This alignment promotes data democratisation, empowering domain experts to manage and utilise their data more effectively.

Additionally, the adoption of microservices architecture, characterised by modular, loosely coupled services, echoes the flexibility and agility sought in modern data fabric architectures (Hechler et al., 2023), enabling independent development, deployment, and scaling of individual components to meet evolving business needs.

However, integrating Metamycelium into existing workloads may pose difficulties due to potential incompatibilities with legacy systems and processes. As highlighted in "Effective Software Architecture" by Goldman (2024), successful integration requires careful planning, clear communication, and a deep understanding of both the new architecture and the existing systems. The absence of well-defined integration patterns and process models for decentralised BD architectures like Metamycelium could impede adoption.

Goldman (2024) emphasises the importance of establishing clear guidelines and best practices for integrating new architectures with existing systems. This includes identifying potential points of integration, defining data exchange protocols, and establishing

governance mechanisms to ensure smooth collaboration between different components of the architecture. Additionally, the involvement of product owners and agile scrum masters is crucial in aligning the technical implementation with business objectives and ensuring that the integration process is iterative and adaptable to changing requirements (Yang, Liang & Avgeriou, 2016).

In the context of the book "Deciphering Data Architectures: Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh" by Serra (2024b), Metamycelium presents itself as a potential alternative to the established data warehouse, data fabric, and data lakehouse architectures. While each architecture offers its own advantages, Metamycelium addresses their limitations by promoting scalability and agility.

The decentralised nature of Metamycelium allows for elastic scaling and the independent evolution of individual domains, making it well-suited for dynamic and evolving data landscapes. In terms of data ownership and governance, the domain-driven approach fosters data ownership and accountability within individual domains, potentially improving data quality and facilitating compliance with data governance regulations.

In terms of flexibility and interoperability, the use of microservices enables modularity and the integration of diverse tools and technologies, enhancing the overall flexibility and interoperability of the architecture. However, as Serra (2024b) cautions, the complexity of decentralised architectures necessitates careful planning, robust governance mechanisms, and a cultural shift within organisations accustomed to centralised data management practices.

Furthermore, the emergence of generative deep learning presents both opportunities and challenges for Metamycelium. Generative models have the potential to revolutionise data processing and analysis tasks, such as data generation, augmentation, and anomaly detection. However, their integration into decentralised architectures necessitates careful

consideration of data privacy, security, and the ethical implications of using powerful generative models, especially with regards to potential misuse or bias.

9.4.2 Implications for Fields of Research

From an academic perspective, Metamycelium contributes to the growing body of knowledge on decentralised BD architectures. The SLR in Chapter 4 underscores the need for more research on domain-driven, decentralised approaches, as well as the importance of addressing cross-cutting concerns such as data quality, metadata management, and security.

Metamycelium's emphasis on these aspects aligns with the identified research gaps and priorities in the field. The adoption of microservices patterns in RAs like Phi (Maamouri et al., 2021), discussed in Chapter 4, further highlights the academic community's growing interest in modular and decentralised approaches to BD management.

The thesis findings underscore the importance of employing comprehensive evaluation strategies that align with the specific aspects being assessed and the research objectives. While the case mechanism experiment provides valuable insights into the technical performance of Metamycelium, it is essential to recognise that the controlled nature of the experiment may not fully capture the complexities and nuances of real-world implementations. Future research should consider a multi-faceted approach that encompasses not only technical performance metrics but also socio-technical factors, such as organisational culture, data governance practices, and the specific context in which the architecture or artefact is being deployed.

Furthermore, the emergence of LLMs and the broader field of AI presents both challenges and opportunities for BD architectures. The ability of LLMs to process and generate human-like text at scale has opened up new possibilities for data analysis, knowledge extraction, and decision-making (S. Yin et al., 2023).

However, the effective deployment and utilisation of LLMs require scalable and efficient BD architectures to handle the massive amounts of data involved in training and inference processes. Metamycelium's decentralised and domain-driven approach, with its focus on modularity and scalability, could potentially address these challenges and provide a suitable foundation for integrating LLMs into BD systems.

The domain-driven design of Metamycelium could also facilitate the development of specialised LLMs tailored to specific domains or tasks, improving their accuracy and efficiency (Y. Chang et al., 2024). However, the integration of LLMs into decentralised BD architectures also poses challenges.

Issues such as data privacy, security, and the ethical implications of AI-driven decision-making need to be carefully considered. Metamycelium's emphasis on data governance and security provides a foundation for addressing these concerns. However, further research is needed to explore how decentralised architectures can effectively manage the unique challenges and opportunities presented by LLMs and AI in general.

Metamycelium's positioning within the existing body of knowledge on BD architectures is further solidified by its alignment with emerging industry trends. The architecture's emphasis on decentralised data ownership and governance, as well as its modular design, resonates with the growing interest in data mesh and microservices architectures (Dehghani, 2022).

While Metamycelium is not explicitly a data mesh or microservices implementation, it incorporates key principles from both paradigms, demonstrating the potential for cross-fertilisation of ideas between academic research and industry practice. Future research could explore the synergies between Metamycelium and these emerging architectures, potentially leading to the development of more comprehensive and adaptable solutions for BD management.

9.5 Generalisability and Transferability

The generalisability and transferability of the findings from this research are subject to certain considerations. While the case mechanism experiment provided empirical evidence of Metamycelium's capabilities within a controlled setting, its applicability in diverse real-world contexts requires further investigation. Generalisability, as defined by Hellström (2006), refers to the extent to which research findings can be applied to a larger population or context beyond the sample studied.

In this case, while the experiment demonstrated the architecture's effectiveness in handling various data scenarios, the controlled nature of the experiment limits the extent to which these findings can be generalised to the broader landscape of BD systems, which may encompass different organisational structures, data governance practices, and technological constraints.

Furthermore, the expert consultations, while providing valuable insights into Metamycelium's potential benefits and challenges, are inherently limited in their generalisability. The opinions of the experts consulted, although insightful, represent a specific subset of perspectives and may not be universally applicable to all potential users of the architecture.

The transferability of the findings, defined as the extent to which they can be applied to similar contexts or situations, also warrants further investigation. While Metamycelium's decentralised and domain-driven approach aligns with emerging industry trends like Data Mesh and Data Fabric, the specific challenges and opportunities associated with implementing such an architecture may vary across different domains and industries.

To enhance the generalisability and transferability of the research findings, future studies could focus on replicating the case mechanism experiment in diverse real-world settings, involving different organisations and data ecosystems. Additionally, a broader range of expert opinions could be sought, encompassing diverse perspectives from

various industries and domains. Furthermore, the Technology Acceptance Model (TAM) (Davis, 1985) could provide a valuable framework for assessing the transferability of Metamycelium and understanding the factors influencing its adoption in diverse organizational settings. TAM posits that perceived usefulness and perceived ease of use are critical determinants of technology acceptance.

While this study provides insights into Metamycelium's potential as a decentralised BD architecture, further research is needed to establish its generalisability and transferability across diverse real-world scenarios. By addressing the limitations of the current study and incorporating a wider range of perspectives and contexts, future research can contribute to a more comprehensive understanding of Metamycelium's applicability and impact in the broader landscape of BD systems.

9.5.1 Public Availability of Research Artefacts

The code used in the case mechanism experiment has been made publicly available. Public access to the code allows for independent verification and reproducibility of the results. However, the successful replication of findings and adaptation of the Metamycelium architecture across diverse contexts may be influenced by factors beyond code accessibility alone. While public availability contributes to transparency and reproducibility, it does not inherently guarantee generalisability or transferability of the research.

9.5.2 Published Validation

Various iterations of the artefact and the design theories that underpin the artefact, underwent several rigorous peer-review processes, with submissions to multiple top-tier academic venues, including esteemed journals and conferences like IEEE Access ("IEEE Access", 2024) and America's Conference on Information Systems ("Americas

Conference on Information Systems,AMCIS", 2024). While publications in these venues may affirm the artefact's quality, depth, and relevance to a certain extent, it is crucial to critically analyse the implications of the peer-review process.

The peer-review process is inherently iterative, with the artefact undergoing multiple rounds of critical assessments and revisions based on feedback from reviewers. This iterative nature can be seen as a strength, as it subjects the artefact to diverse perspectives and critiques, potentially enhancing its robustness and relevance. However, it is essential to acknowledge the potential biases and limitations of the peer-review process itself. Reviewers' assessments may be influenced by their own research backgrounds, methodological preferences, and theoretical orientations, potentially leading to biases or the overlooking of alternative viewpoints.

Throughout the research process, the feedback received from reviewers presented both challenges and opportunities for growth. Some comments highlighted areas that required further clarification or refinement, while others prompted the exploration of alternative perspectives. Engaging with this feedback constructively allowed for the identification of areas for improvement and the strengthening of the research.

Given this, the fact that the Metamycelium artefact has undergone multiple rounds of peer-review and has been published in reputable academic venues suggests that it has survived a certain level of critical scrutiny. This process of consolidating the artefact's theoretical foundations and addressing potential concerns raised by reviewers can contribute to enhancing its accuracy and generalisability within the academic discourse.

9.5.3 Note on Research Scope

Architectural design inherently involves a delicate balance between theoretical principles and practical constraints imposed by real-world scenarios (Bass, Clements & Kazman, 2003). Consequently, the theoretical underpinnings and design principles proposed may

not be universally applicable or equally effective across all contexts.

The strengths and potential benefits of Metamycelium have been acknowledged, but it is essential to recognise that these strengths may be contingent upon specific conditions or assumptions. For instance, the decentralised and domain-driven approach advocated by Metamycelium might be more suitable for organisations with mature data governance practices, established cross-functional teams, and a culture that embraces distributed decision-making. However, in organisations with centralised data management structures or rigid hierarchies, the implementation of Metamycelium could face significant challenges and resistance.

Furthermore, while the transparency of the code aids in reproducibility, it is important to acknowledge that the successful replication of the research findings could be influenced by various factors beyond the code itself. Variations in implementation environments, data characteristics, infrastructure configurations, and the expertise of the personnel involved might potentially yield different results or uncover unforeseen challenges.

Additionally, the scope of the research may be limited by the specific use cases, data sets, and organisational contexts considered during the development and evaluation of Metamycelium. While efforts have been made to ensure the generalisability of the approach, it is possible that certain domain-specific nuances or edge cases may not have been sufficiently explored or accounted for.

9.6 Research Process and Insights

This section delves into the research process of BD RAs, highlighting the technical complexities and insights encountered. It emphasises the importance of effective communication within academic and industry contexts and the challenges of developing a technically advanced artefact.

9.6.1 Navigating Technical Complexities and Communication

The development of the Metamycelium artefact involved navigating intricate technical challenges stemming from the integration of emerging concepts such as platform development, software engineering, data engineering, and DevOps. These concepts may present varying levels of familiarity to different audiences. Therefore, communicating the technical aspects of Metamycelium necessitates a nuanced approach that caters to a diverse readership.

The inherent complexity of Metamycelium requires meticulous attention to detail in design, development, and documentation. While this study demonstrates efforts to articulate these technical intricacies, further clarification could enhance the understanding of the architecture's implementation details. A more in-depth discussion of the specific challenges encountered during the development and integration of the various architectural components, such as establishing clear boundaries between domains and ensuring seamless communication and coordination between them, would provide valuable insights for both researchers and practitioners.

Furthermore, the communication of complex technical concepts requires a balance between clarity and comprehensiveness. While the thesis strives to explain the technical underpinnings of Metamycelium, the use of excessive technical jargon may pose a barrier to understanding for some readers

9.6.2 Academic Reception and Industry Perspective

The initial iterations of the artefact highlighted a potential disconnect between the study's intended objectives and the interpretations drawn by some audiences. This discrepancy underscores the challenges inherent in communicating complex technical research to a diverse readership with varying levels of expertise and familiarity with emerging concepts. The technical depth of the research, while essential for rigour and

validity, may have inadvertently obscured the practical implications of the proposed architecture for some audiences.

This disconnect also highlights the need to bridge the gap between theoretical advancements in BD RAs and their real-world applications, especially in comparison to rapidly evolving fields like AI. Effectively communicating the potential impact of Metamycelium in practical settings may require a more explicit articulation of its potential benefits and challenges, particularly for stakeholders less familiar with the technical nuances of BD architectures.

9.6.3 Industry Perception of RAs

In professional circles, the concept of RAs for BD systems, while recognised, often requires a more detailed explanation. This need for clarification suggests a potential disconnect between the theoretical understanding of RAs within academia and their practical application in industry. While the academic community is generally familiar with the concept and benefits of RAs, industry professionals may have varying levels of understanding and adoption of such architectures in their BD practices.

The expert consultations conducted as part of this research (Chapter 8) revealed a range of perspectives on RAs. Some experts acknowledged the potential value of RAs in providing a structured approach to BD system design and implementation, while others expressed concerns about their complexity and potential lack of flexibility in addressing unique organisational requirements. Additionally, the need for specialised expertise and resources to implement and maintain RAs was identified as a potential barrier to adoption in some industrial settings.

The observed need for clarification on the concept of RAs in industry highlights an opportunity for greater collaboration between academia and industry to bridge the gap between theoretical knowledge and practical implementation. This could

involve developing educational resources and training programmes to enhance industry professionals' understanding of RAs, as well as creating more accessible and practical guidance on how to tailor RAs to specific organisational contexts and use cases.

9.6.4 Resource Constraints

The extensive scope of this research necessitated substantial resources to adequately explore and validate novel concepts. In this study, several resource constraints were encountered.

Financial constraints limited access to specialised tools, datasets, and computing resources that could have facilitated faster evaluation of Metamycelium. The inability to fully replicate real-world BD environments, due to budgetary limitations, posed a challenge in assessing the architecture's performance and scalability under diverse conditions. Additionally, the limited availability of time restricted the scope of the research. For instance, this research could be extended with the addition of AI constructs.

The reliance on custom-developed components for certain aspects of the Metamycelium architecture further exemplifies the resource constraints encountered in this study. While these components served the immediate needs of the research, their development and maintenance demanded significant time and effort. This highlights the broader challenge of resource allocation in research, where individual researchers often lack the resources to develop fully-fledged software solutions.

9.7 Conclusion

The in-depth study of BD RAs, particularly the Metamycelium architecture, has underscored its potential to redefine domain-driven distributed systems. The analysis, bridging empirical case findings with expert perspectives, establishes the Metamycelium

architecture's potency and applicability, emphasising the necessity of a decentralised approach in modern data ecosystems.

Moving away from traditional centralised systems, this research champions decentralised architectures that can handle today's complex data demands. For fields of research, the implication is clear: intensify exploration in BD RAs and focus on architectures that align with industry needs. Practitioners are urged to transition from monolithic structures to agile, decentralised systems emphasising data quality and robust communication.

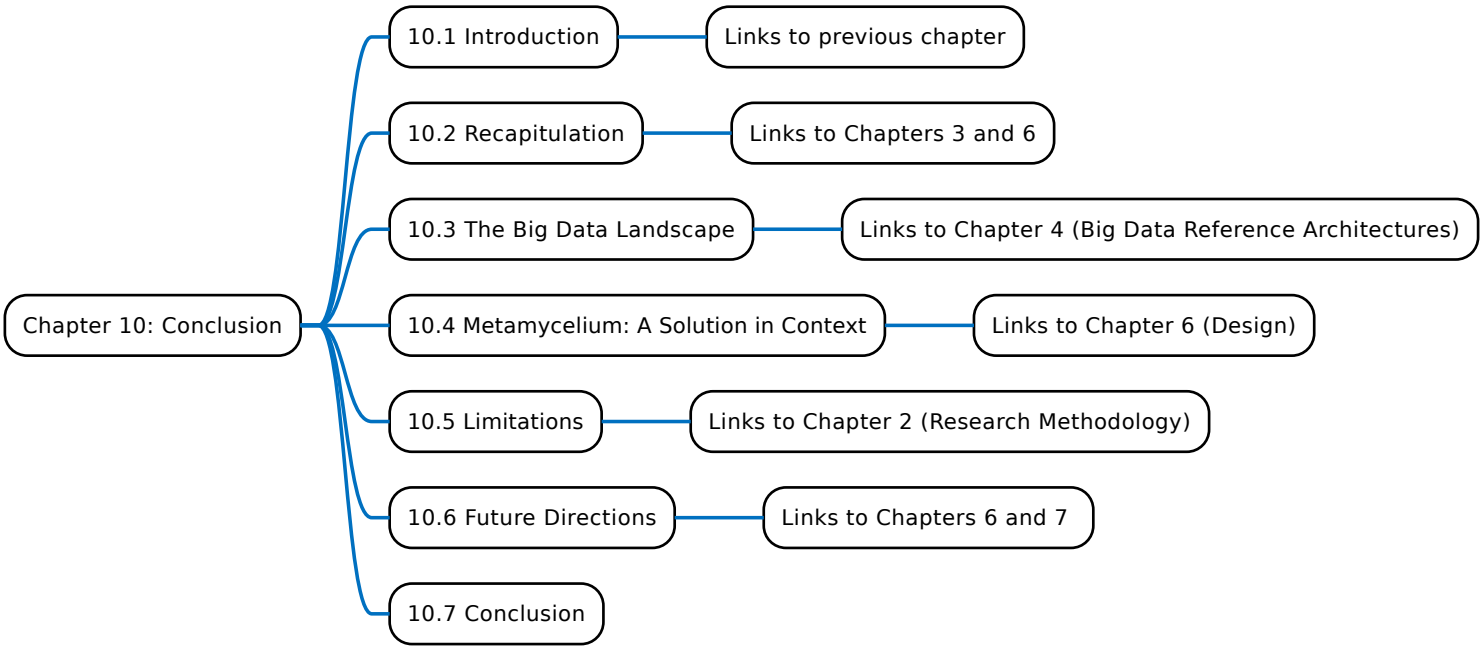
The research process employed in this study, including the iterative refinement of the architecture based on feedback from expert consultations and peer reviews, aimed to enhance the validity and rigour of the findings. However, it is important to acknowledge that any research endeavour has inherent limitations. Further validation through real-world implementations and broader scrutiny within the research community would contribute to a more comprehensive assessment of the architecture's effectiveness and generalisability.

Reflecting on the discourse, it is evident that innovations in BD RAs, like the Metamycelium, are born from iterative progress, drawing from past efforts like Lambda (Kiran, Murphy, Monga, Dugan & Baveja, 2015b), Kappa (J. Lin, 2017), and Neomycelia (Ataei & Litchfield, 2021a). However, call for careful consideration in future endeavours.

In conclusion, this chapter illuminates the particularities of BD architectures. It calls for continued innovation, empirical validation, and collaboration to ensure that future BD systems are efficient, scalable, and adaptable to emerging challenges. In the next chapter, the thesis is concluded, and key aspects and outcomes of the research are discussed.

Chapter 10

Conclusion



10.1 Introduction

Following the detailed discussion in the previous chapter, this chapter concludes the thesis by encapsulating the key aspects and outcomes of the research. It revisits the initial problem statement and research questions, outlining how the study addressed

these through its methodologies and findings. Here, the focus is on synthesising the entire journey, from inception to conclusion, and reflecting on the implications of the research within the field of BD RAs. The chapter also acknowledges the limitations encountered during the study and proposes avenues for future research, aiming to further the discourse in the ever-evolving domain of BD systems.

The conclusion chapter commences with an introduction that sets the stage for the synthesis of the research journey and its implications within the field of BD RAs. Section 10.2 recapitulates the foundational aspects of BD that motivated this study, highlighting the importance and challenges of BD, and restating the research questions aimed at addressing BD project failures. The current landscape of BD is summarised in Section 10.3, emphasising the gaps and challenges identified through the SLRs. Metamycelium, as a solution to these challenges, is discussed in the context of existing architectures in Section 10.4, highlighting its novel aspects and potential to address the limitations of current BD RAs. Section 10.5 illustrates the unique contributions of this study to the field of research. Section 10.6 acknowledges the limitations encountered during the study, encompassing methodological constraints of literature reviews, research methodology limitations, and the evolutionary journey of RAs. Finally, Section 10.7 proposes avenues for future research, exploring areas such as empirical validation, socio-technical considerations, integration with emerging technologies, and the development of implementation guidelines and best practices, aiming to further the discourse in the ever-evolving domain of BD systems.

10.2 Recapitulation

This section revisits the foundational aspects of BD that motivated this study, incorporating relevant literature to highlight the challenges in BD project implementation and the context for the research questions addressed.

10.2.1 The Importance and Challenges of Big Data

The advent of digital technologies has catalysed an unprecedented surge in data generation, marking the onset of the BD era. This development underscores the necessity of advanced data management systems adept at handling the complexities and sheer volume of contemporary data.

The importance of BD lies in its potential to reveal valuable insights and patterns that can drive business decisions and innovation (Lycett, 2013), enable organisations to better understand and serve their customers (Bughin, 2016), and optimise operations and improve efficiency across various industries (Manyika et al., 2011).

Despite the potential that BD holds, its implementation is fraught with challenges. The complexity of data architectures often struggles to handle the volume, velocity, and variety of BD (H. Jagadish et al., 2014). The rapid pace of technological change makes it difficult for organisations to keep up with the latest tools and techniques (H.-M. Chen, Schütz, Kazman & Matthes, 2017). Moreover, organisational and cultural barriers to becoming data-driven, such as silos and resistance to change, pose significant hurdles (Mikalef et al., 2018).

These challenges contribute to the high failure rate of BD projects. Studies such as those by technology review insights in partnership with Databricks (2021) and NewVantage (2021) underscore the prevalence of challenges in achieving successful BD outcomes, with failure rates ranging from 60% to 85%.

The SLR on BD RAs (Chapter 4) identifies several key failure points, including the reliance on monolithic data pipeline architectures, which can lead to scalability issues, data quality concerns, and difficulties in adapting to rapid technological changes (Gorton & Klein, 2015; Nadal, Herrero, Romero, Abelló et al., 2017). Additionally, the review highlights the lack of attention to cross-cutting concerns, data silos, lack of domain ownership, and difficulties in maintaining data lineage.

These failure points and architectural challenges highlight the need for effective BD management strategies and architectures that can address the technical and organisational challenges of implementing BD systems. The research questions posed in this thesis (Section 2.3) aim to address these challenges by proposing a novel, domain-driven, and decentralised RA for BD systems.

10.2.2 Research Questions: Addressing BD Project Failures

The study is driven by two pivotal research questions:

What are the limitations of current big data reference architectures? This inquiry delves into the root causes of failures in BD projects, identifying architectural and design shortcomings as outlined in studies like those by Gorton and Klein (2015) and Nadal, Herrero, Romero, Abelló et al. (2017).

How can a reference architecture be designed to mitigate or address these limitations in big data projects? Building on the identified failure modes, the research seeks to conceptualise a RA that effectively addresses these challenges. As per Angelov et al. (2008), the goal is to develop an architectural framework that not only resolves current BD project issues but also provides a scalable and adaptable foundation for future BD systems.

Guided by these questions, the development of the Metamycelium RA has been undertaken, targeting the resolution of prevalent challenges and contributing to the evolving landscape of BD management.

10.3 The Big Data Landscape

This study embarked on a comprehensive exploration of the current landscape of BD RAs through a rigorous SLR, addressing a significant gap in systematic academic exploration. Despite the potential role of RAs in guiding the development and maintenance

of BD systems, focused SLRs on BD RAs remain surprisingly sparse in academic literature. This research contributes to filling this gap, offering an analysis that stands as a pioneering effort in this domain.

The work of Volk et al. (2019), while not directly focusing on BD RAs, offered a decision support system for selecting BD architectures. This approach, however, did not delve into a systematic collection or analysis of BD architectures, highlighting a different yet related area of study. In another vein, the NIST BD RA team (W. L. Chang et al., 2015) utilised white papers from the BD Public Working Group as foundational materials. These documents, however, served more as conceptual comparisons than as detailed BD RAs, not fully encompassing the comprehensive scope typically associated with RAs.

The findings of this study revealed that advancements within the BD RA domain are inconsistent across academia and industry. While there has been considerable progress in data warehousing, AI, data science, and the IoT, data engineering requires more in-depth research and development. A significant proportion of the BD RAs reviewed rely on a monolithic data pipeline with centralised storage, raising scalability and maintainability concerns. This is particularly problematic in industry settings where BD systems need to handle massive and rapidly growing datasets.

Moreover, the quality of many BD RAs assessed often did not meet industrial standards, attributed to the complexities involved in their development and the need for substantial resources for their evaluation. This shortfall highlights the absence of a robust evaluative framework for RAs, as pointed out in studies like Angelov et al. (2008). In academia, there is a need for more rigorous research on BD RAs to address these challenges and provide guidance for the development of effective BD systems that can meet the demands of both research and industry applications.

The industry landscape is also evolving, with emerging approaches like Data Mesh (Dehghani, 2022), Data Fabric (Hechler et al., 2023), and DataOS (The Modern Data

Company, 2024) aiming to address scalability, agility, and governance challenges. Data Mesh promotes decentralised data ownership by individual domains, enhancing data quality and agility. Data Fabric provides unified access and management of disparate data sources, simplifying integration and sharing. DataOS offers operating principles for managing data as a strategic asset, improving quality, accessibility, and governance. These newer approaches align with the principles of Metamycelium for more scalable and agile architectures in the BD landscape.

This study emphasises the need for an intensified focus on RAs within the BD sphere from both academic and industrial perspectives. The identification of challenges and limitations in current BD RAs underscores the importance of further research in this area to develop more robust and scalable BD systems.

10.4 Metamycelium: A Solution in Context

The development of Metamycelium addresses challenges in managing BD in the context of evolving data management strategies. This architecture introduces a shift from traditional centralised data structures to a more decentralised, domain-driven approach, focusing on improving data integrity and organisational efficiency.

In conventional systems, the segregation of analytical and operational data, often overseen by the CDAO and CTO respectively, creates silos. This separation impacts the efficient use of data and leads to complex integration processes. ETLs, commonly used for this integration, have limitations in terms of their maintenance and adaptability.

Metamycelium proposes a more integrated method of data management, where data stays proximate to its source. This proximity enables quicker access and use, reducing the dependency on intermediary processes like ETLs, thereby streamlining data workflows and enhancing organisational adaptability.

The architecture of Metamycelium is decentralised and modular. It suggests a

network of data nodes, each addressing specific analytical or operational tasks. This setup not only helps maintain data integrity by keeping it close to its source but also allows for efficient data exchanges and updates across the network. The result is a system that is more adaptable and resilient to errors.

Metamycelium's approach addresses common bottlenecks found in traditional data architectures, leading to a more dynamic management of data. It provides timely and relevant data access to various stakeholders, from data engineers to software developers, aligning with the needs of contemporary data-driven decision-making processes.

Taken all into consideration, Metamycelium offers an approach to BD management that seeks to improve upon traditional data management strategies. Its focus on decentralisation and domain-driven design presents a new pathway in the ongoing evolution of BD systems.

10.5 Unique Contributions to the Field of Research

This study makes several unique contributions to the field of BD RAs and the broader domain of software engineering for BD systems:

1. **Proposing a novel, domain-driven, decentralised reference architecture:** Metamycelium, introduced in Chapter 6, represents a departure from the predominant monolithic, centralised architectures in the BD landscape, as identified in the SLR conducted in Chapter 4. By advocating for a domain-driven approach, where data ownership and management are decentralised to the domain level, Metamycelium addresses key challenges such as scalability, data quality, and agility (Dehghani, 2022; Hechler et al., 2023). This novel architectural approach contributes to the ongoing discourse on the design and development of BD systems, offering an alternative to the limitations of current architectures identified in the SLR.

2. **Comprehensive analysis of the current state of big data reference architectures:** The SLRs conducted in Chapters 4 and 5 provide a thorough examination of the existing BD RAs, their components, and their limitations. This analysis contributes to a deeper understanding of the current state of research and practice in this field, identifying gaps and opportunities for further research and development. The findings from these SLRs align with the challenges and limitations identified in the industry (Gorton & Klein, 2015; Nadal, Herrero, Romero, Abelló et al., 2017), substantiating the need for novel architectural approaches.

3. **Integration of software engineering principles and patterns:** Metamycelium incorporates key software engineering principles and patterns, such as microservices and event-driven communication, into the design of a BD RA, as discussed in Chapters 5 and 6. This integration contributes to bridging the gap between software engineering and data engineering, promoting the adoption of best practices and fostering cross-disciplinary collaboration. The incorporation of these principles and patterns aligns with the emerging trends in the industry, such as the adoption of microservices architectures and event-driven systems (S. Newman, 2015b; Stopford, 2018).

4. **Novel approach to evaluating a reference architecture within the DSR framework:** This study employs a combination of evaluation methods, including a case mechanism experiment (Chapter 7) and expert opinion gathering (Chapter 8), to assess the effectiveness and applicability of Metamycelium within the DSR framework. The use of the empirically grounded RA methodology (Galster & Avgeriou, 2011a) in conjunction with DSR contributes to addressing the specific challenges associated with evaluating RAs, such as the lack of a clearly defined group of stakeholders and the difficulty in creating valid evaluation scenarios (Angelov & Grefen, 2008b; Angelov et al., 2012). By documenting the evaluation

process and the challenges encountered, this study provides valuable insights for researchers looking to evaluate RAs in the context of DSR.

Collectively, these contributions push the boundaries of current practices in BD system design and implementation. The Metamycelium architecture, grounded in a thorough analysis of existing models and underpinned by robust software engineering principles, provides a blueprint for the future of scalable, adaptable, and domain-focused BD solutions. This research not only addresses immediate industry challenges but also lays a foundation for further exploration and innovation in the dynamic field of BD architecture.

10.6 Limitations

This section acknowledges and addresses the limitations encountered throughout the research process, encompassing methodological constraints, and research methodology limitations, as well as the limitations identified from the discussion and literature review chapters of the thesis.

The case mechanism experiment and expert opinions may have been based on a relatively small sample size, which could limit the generalisability of the findings. Moreover, the adoption of the Galster and Avgeriou (2011b) empirically grounded RA methodology, while appropriate for the study's objectives, may have inherent limitations in terms of empirical validation and real-world testing at scale.

Lastly, the study acknowledges resource constraints, such as limited access to specialised tools, datasets, and computing resources, which could have impacted the depth and breadth of the evaluation and testing phases. Additionally, the reliance on custom-developed components (e.g., Data Lichen) due to resource constraints might raise questions about the sustainability and scalability of such solutions in real-world implementations.

10.7 Future Directions

The exploration of BD RAs suggests a diverse range of research paths. An emerging area of interest is the exploration of decentralised, domain-driven BD systems. These systems, while not the only solution, present an alternative to current centralised models, potentially addressing scalability, data quality, and interoperability issues.

A key area for future research is the empirical validation of these decentralised BD systems. Real-world case studies and applications are necessary to evaluate their practicality and effectiveness. Such empirical evidence will provide a clearer understanding of the benefits and limitations of decentralised architectures.

The role of metadata in BD systems is another important research area. Investigating robust metadata management within architectures like Metamycelium could address challenges in data privacy, security, lineage, and provenance. Effective metadata management could lead to more streamlined and efficient data management practices in BD systems.

Additionally, exploring the sociotechnical dynamics of adopting decentralised BD systems is crucial. Research should focus on the necessary technological solutions and organisational changes for effective implementation, considering the integration of these systems into diverse organisational contexts.

Another important research direction is alternatives to monolithic data pipeline architectures in BD systems. Investigating architectural models that segregate data processing by domain could offer more manageable and scalable solutions, particularly for handling data across various domains.

While there are standards in distributed systems, their application in the BD context is less defined. Future research should focus on developing standard data communication approaches tailored for BD systems. Establishing such standards could enhance interoperability and efficiency.

Moreover, the exploration of privacy and security in BD RAs remains a critical area. With the increasing focus on data privacy regulations like GDPR, developing secure components for data scrubbing and enhancing pipeline security is essential.

Finally, future BD RA research should strive to address the biases and limitations in current architectures influenced by specific technological offerings. Embracing a more inclusive approach and exploring a wide range of technological solutions could avoid limiting architectural choices and promote a more versatile and adaptive BD ecosystem.

In conclusion, future research in BD RAs should explore decentralised, domain-driven systems, metadata management, data communication standards, and privacy and security issues. This holistic approach is essential for advancing BD management strategies, offering balanced and multifaceted solutions to the complex challenges in the field.

10.8 Conclusion

The research focused on the Metamycelium architecture highlights a shift in the domain of BD systems towards decentralised architectures. These architectures address the complexities of the contemporary data landscape, making them increasingly relevant. The novel design of the Metamycelium framework is well-suited to meet these evolving data management needs.

The methodological rigour of this study, which included literature reviews and feedback from experts, enhanced the validity of its findings. This approach ensured a detailed and objective evaluation of the Metamycelium framework, affirming its applicability.

Nevertheless, it is essential to recognise the inherent limitations and potential biases within the research methodology. Acknowledging these aspects is crucial for a balanced understanding of the study's contributions and for pinpointing areas that necessitate

further research.

Reflecting on the research process, several key insights emerge. The BD systems landscape is in a state of constant evolution, necessitating continuous innovation and refinement. While the Metamycelium architecture marks a step forward in this landscape, ongoing exploration and adaptation are imperative. Empirical validation of the framework in diverse real-world contexts is essential to confirm its effectiveness and identify areas for improvement. Future research efforts, benefiting from collaborative approaches, will likely yield richer perspectives and more comprehensive outcomes.

Looking ahead, the BD systems field presents a range of opportunities and challenges. Insights from this research can guide future studies, enabling them to navigate these challenges effectively and contribute to the development of more dynamic, efficient, and resilient BD systems.

References

2022 gartner® magic quadrant™ for analytics and business intelligence platforms (Tech. Rep.). (2022). Gartner. Retrieved from https://www.tableau.com/asset/gartner-magic-quadrant-2022 (Accessed: 2023-06-03)

Aboulafia, M. (1991). *Philosophy, social theory, and the thought of george herbert mead*. SUNY Press.

Abran, A., Moore, J. W., Bourque, P., Dupuis, R. & Tripp, L. (2004). Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*, 25.

Ainsworth, S. & Jones, T. M. (2020). Prefetching in functional languages. *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*. doi: 10.1145/3381898.3397209

Akhtar, P., Frynas, J. G., Mellahi, K. & Ullah, S. (2019). Big data-savvy teams' skills, big data-driven actions and business performance. *British Journal of Management*, *30*(2), 252–271.

Aksakalli, I. K., Çelik, T., Can, A. B. & Tekinerdogan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, *180*, 111014.

Al-Jaroodi, J. & Mohamed, N. (2016). Characteristics and requirements of big data analytics applications. In *2016 ieee 2nd international conference on collaboration and internet computing (cic)* (pp. 426–432).

Almaatouq, A., Shmueli, E., Nouh, M., Alabdulkareem, A., Singh, V. K., Alsaleh, M., ... Alfaris, A. (2016). If it looks like a spammer and behaves like a spammer, it must be a spammer: analysis and detection of microblogging spam accounts [Journal Article]. *International Journal of Information Security*, *15*(5), 475-491. doi: 10.1007/s10207-016-0321-5

Amatriain, X. (2013). Beyond data: from user information to business value through personalized recommendations and consumer science [Conference Proceedings]. In (p. 2201-2208). ACM. doi: 10.1145/2505515.2514691

Amazon Web Services. (2024). *Reference architecture 2 - analytics lens.* https://docs.aws.amazon.com/wellarchitected/latest/analytics-lens/reference-architecture-2.html. (Accessed: 2024-03-23)

Amazon Web Services, I. (2023). *Big data analytics options on aws* (White Paper). Author Retrieved from https://docs.aws.amazon.com/

pdfs/whitepapers/latest/big-data-analytics-options/big-data-analytics-options.pdf#welcome

Americas conference on information systems,amcis. (2024). Retrieved from https://aisnet.org/page/AMCIS

Angelov, S. & Grefen, P. (2008a). An e-contracting reference architecture. *Journal of Systems and Software*, *81*(11), 1816–1844.

Angelov, S. & Grefen, P. (2008b). An e-contracting reference architecture [Journal Article]. *Journal of Systems and Software*, *81*(11), 1816-1844. doi: 10.1016/j.jss.2008.02.023

Angelov, S., Grefen, P. & Greefhorst, D. (2009). A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 joint working ieee/ifip conference on software architecture & european conference on software architecture* (pp. 141–150).

Angelov, S., Grefen, P. & Greefhorst, D. (2012). A framework for analysis and design of software reference architectures. *Information and Software Technology*, *54*(4), 417–431.

Angelov, S., Trienekens, J. J. & Grefen, P. (2008). Towards a method for the evaluation of reference architectures: Experiences from a case. In *European conference on software architecture* (pp. 225–240).

ANSI, A. (1975). X3/sparc study group on dbms, interim report [Journal Article]. *SIGMOD FDT Bull*, *7*(2).

Apache. (2023a). *Apache airflow: Platform to programmatically author, schedule, and monitor data pipelines.* Retrieved from https://airflow.apache.org/

Apache. (2023b). *Apache projects directory.* Retrieved from https://projects.apache.org/ (Accessed: 2023-06-03)

Apache. (2023c). *Hadoop: Open-source software for reliable, scalable, distributed computing.* Retrieved from https://hadoop.apache.org/

Apollo GraphQL. (2023). *Apollo federation.* https://www.apollographql.com/docs/federation/. (Accessed: May 10, 2023)

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., ... others (2016). Scone: Secure linux containers with intel sgx. In *Osdi* (Vol. 16, pp. 689–703).

Asur, S. & Huberman, B. A. (2010). Predicting the future with social media. In *2010 ieee/wic/acm international conference on web intelligence and intelligent agent technology* (Vol. 1, pp. 492–499). doi: 10.1109/wi-iat.2010.63

Ataei, P. & Litchfield, A. (2020). Big data reference architectures: A systematic literature review. In *2020 31st australasian conference on information systems (acis)* (pp. 1–11). doi: 10.5130/acis2020.bf

Ataei, P. & Litchfield, A. (2021a). Neomycelia: A software reference architecturefor big data systems. In *2021 28th asia-pacific software engineering conference (apsec)* (pp. 452–462).

Ataei, P. & Litchfield, A. (2021b, dec). Neomycelia: A software reference architecturefor big data systems. In *2021 28th asia-pacific software engineering conference (apsec)* (p. 452-462). Los Alamitos, CA, USA: IEEE Computer Society. Retrieved from https://doi.ieeecomputersociety.org/10.1109/

APSEC53868.2021.00052 doi: 10.1109/APSEC53868.2021.00052

Ataei, P. & Litchfield, A. (2022). The state of big data reference architectures: A systematic literature review. *IEEE Access*, *10*, 113789-113807. doi: 10.1109/ACCESS.2022.3217557

Ataei, P. & Litchfield, A. (2023). Towards a domain-driven distributed reference architecture for big data systems. In *Amcis 2023.* Minneapolis, Minnesota, USA.

Ataei, P. & Staegemann, D. (2023). Application of microservices patterns to big data systems. *Journal of Big Data*, *10*(1), 1–49.

Australian Bureau of Statistics. (2021). *ANZSCO: Australian and new zealand standard classification of occupations.* https://www.abs.gov.au/statistics/classifications/anzsco-australian-and-new-zealand-standard-classification-occupations/latest-release. (Accessed: March 25, 2023)

Avgeriou, P. (2003a). Describing, instantiating and evaluating a reference architecture: A case study [Journal Article]. *Enterprise Architecture Journal*, *342*, 1-24.

Avgeriou, P. (2003b). Describing, instantiating and evaluating a reference architecture: A case study. *Enterprise Architecture Journal*, *342*, 1–24.

Azure architecture center. (2024). https://docs.microsoft.com/en-us/azure/architecture/. (Accessed: April 9, 2023)

Bahrami, M. & Singhal, M. (2015). The role of cloud computing architecture in big data [Book Section]. In *Information granularity, big data, and computational intelligence* (p. 275-295). Springer. doi: 10.1201/9781315155678-8

Baltes, S. & Ralph, P. (2022). Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering*, *27*(4), 1–31.

Bashari Rad, B., Akbarzadeh, N., Ataei, P. & Khakbiz, Y. (2016). Security and privacy challenges in big data era [Journal Article]. *International Journal of Control Theory and Applications*, *9*(43), 437-448.

Bass, L., Clements, P. & Kazman, R. (2003). *Software architecture in practice* [Book]. Addison-Wesley Professional. doi: 10.1109/wicsa.2008.12

Bass, L., Weber, I. & Zhu, L. (2015). *Devops: A software architect's perspective* [Book]. Addison-Wesley Professional.

Bates, D. W., Saria, S., Ohno-Machado, L., Shah, A. & Escobar, G. (2014). Big data in health care: using analytics to identify and manage high-risk and high-cost patients [Journal Article]. *Health Affairs*, *33*(7), 1123-1131. doi: 10.1377/hlthaff.2014.0041

Baum, T., Münch, J. & Ramler, R. (2016, May). Experiment-driven development: A review of empirical studies. *IEEE Transactions on Software Engineering*, *42*(5). doi: 10.1109/TSE.2015.2487331

Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., ... DeBaud, J.-M. (1999). Pulse: A methodology to develop software product lines. In *Proceedings of the 1999 symposium on software reusability* (pp. 122–131).

Benbasat, I. & Zmud, R. W. (1999). Empirical research in information systems: The practice of relevance. *MIS quarterly*, 3–16.

Bengtsson, P. & Bosch, J. (n.d.). Scenario-based software architecture reengineering

[Conference Proceedings]. In *Proceedings. fifth international conference on software reuse (cat. no. 98tb100203)* (p. 308-317). IEEE. doi: 10.1109/icsr.1998.685756

Bengtsson, P. & Bosch, J. (1998). Scenario-based software architecture reengineering. In *Proceedings. fifth international conference on software reuse (cat. no. 98tb100203)* (pp. 308–317).

Berger, P. L., Luckmann, T. et al. (1966). *The social construction of reality: A treatise in the sociology of knowledge*. Anchor.

Bhadani, A. K. & Jothimani, D. (2016). Big data: challenges, opportunities, and realities. *Effective big data management and opportunities for implementation*, 1–24.

Bilal, M., Oyedele, L. O., Akinade, O. O., Ajayi, S. O., Alaka, H. A., Owolabi, H. A., ... Bello, S. A. (2016). Big data architecture for construction waste analytics (cwa): A conceptual framework. *Journal of Building Engineering*, *6*, 144–156.

Bird, R. & Wadler, P. (1988). An introduction to functional programming. *ACM Computing Surveys (CSUR)*, *21*(3), 471–524.

Bogner, J., Wagner, S. & Zimmermann, A. (2019). Using architectural modifiability tactics to examine evolution qualities of service- and microservice-based systems: An approach based on principles and patterns. *SICS Software-Intensive Cyber-Physical Systems*, *34*(2-3), 141–149. doi: 10.1007/s00450-019-00402-z

Bohlouli, M., Dalter, J., Dornhöfer, M., Zenkert, J. & Fathi, M. (2015). Knowledge discovery from social media using big data-provided sentiment analysis (somabit) [Journal Article]. *Journal of Information Science*, *41*(6), 779-798. doi: 10.1177/0165551515602846

Borrego, M., Foster, M. J. & Froyd, J. E. (2014). Systematic literature reviews in engineering education and other developing interdisciplinary fields. *Journal of Engineering Education*, *103*(1), 45–76.

Bosch, J. (2000). *Design and use of software architectures: adopting and evolving a product-line approach*. Pearson Education.

Brackenbury, W., Liu, R., Mondal, M., Elmore, A. J., Ur, B., Chard, K. & Franklin, M. J. (2018). Draining the data swamp: A similarity-based approach. In *Proceedings of the workshop on human-in-the-loop data analytics* (pp. 1–7).

Braun, P. et al. (2021). Tackling consistency-related design challenges of distributed data-intensive systems – an action research study. *ar5iv*.

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77–101.

Brewer, E. (2012). Cap twelve years later: how the [Journal Article]. *Computer*, *45*(2), 23-29. doi: 10.1109/mc.2012.37

Brewer, E. A. (2000). Towards robust distributed systems. In *International conference on dependable systems and networks* (pp. 398–405).

Brikman, Y. (2017). *Terraform: Up & Running*. O'Reilly Media, Inc. ([Print])

Brooks Jr, F. P. (1996). The computer scientist as toolsmith ii. *Communications of the ACM*, *39*(3), 61–68.

Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V. & Sadovykh, A. (2020). Microservices. *Science and Engineering. Springer*.

Bughin, J. (2016). Big data, big bang? [Journal Article]. *Journal of Big Data*, *3*(1), 2. doi: 10.1186/s40537-015-0014-3

Building a high-performance data and ai organization (Tech. Rep.). (2023). MIT Technology Review Insights. Retrieved from `https://www.databricks.com/resources/whitepaper/mit-technology-review-insights-report` (Accessed: 2023-06-03)

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. (2008). *Pattern-oriented software architecture: A system of patterns, volume 1* (Vol. 1). John wiley & sons.

Cackett, D. (2013). Information management and big data, a reference architecture. *Oracle: Redwood City, CA, USA*. Retrieved from `https://www.oracle.com/technetwork/topics/entarch/articles/info-mgmt-big-data-ref-arch-1902853.pdf`

Capterra. (2023). *Latest software categories.* Retrieved from `https://www.capterra.com/categories` (Accessed: 2023-06-03)

Carroll, J. M. (1995). *Scenario-based design: Envisioning work and technology in system development*. Wiley.

Chainey, S., Tompson, L. & Uhlig, S. (2008). The utility of hotspot mapping for predicting spatial patterns of crime [Journal Article]. *Security journal*, *21*(1-2), 4-28. doi: 10.1057/palgrave.sj.8350066

Chang, W. L. & Boyd, D. (2018). *Nist big data interoperability framework: Volume 6, big data reference architecture* (Technical Report). Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST).

Chang, W. L., Grady, N. et al. (2015). *Nist big data interoperability framework: volume 1, big data definitions* (Tech. Rep.). Gaithersburg, MD, USA: National Institute of Standards and Technology.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., ... others (2024). A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, *15*(3), 1–45.

Chen, H., Chiang, R. H. & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact [Journal Article]. *MIS quarterly*, *36*(4), 1165. doi: 10.2307/41703503

Chen, H.-M., Kazman, R., Garbajosa, J. & Gonzalez, E. (2017). Big data value engineering for business model innovation. *HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*.

Chen, H.-M., Kazman, R. & Haziyev, S. (2016a). Agile big data analytics development: An architecture-centric approach [Conference Proceedings]. In *2016 49th hawaii international conference on system sciences (hicss)* (p. 5378-5387). IEEE. doi: 10.1109/hicss.2016.665

Chen, H.-M., Kazman, R. & Haziyev, S. (2016b). Agile big data analytics for web-based systems: An architecture-centric approach [Journal Article]. *IEEE Transactions on Big Data*, *2*(3), 234-248. doi: 10.1109/tbdata.2016.2564982

Chen, H.-M., Kazman, R., Haziyev, S. & Hrytsay, O. (2015). Big data system development: An embedded case study with a global outsourcing firm [Conference Proceedings]. In *Proceedings of the first international workshop on big data software engineering* (p. 44-50). IEEE Press. doi: 10.1109/bigdse.2015.15

Chen, H.-M., Kazman, R. & Matthes, F. (2016). Demystifying big data adoption: Beyond it fashion and relative advantage [Conference Proceedings]. In *Proceedings of pre-icis (international conference on information system) digit workshop.* doi: 10.1109/hicss.2016.631

Chen, H.-M., Schütz, R., Kazman, R. & Matthes, F. (2017). How lufthansa capitalized on big data for business model renovation [Journal Article]. *MIS Quarterly Executive*, *16*(1). doi: 10.24251/hicss.2017.713

Chen, L. & Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, *53*(4), 344–362.

Chen, M., Yang, J., Zhou, J., Hao, Y., Zhang, J. & Youn, C.-H. (2018). 5g-smart diabetes: Toward personalized diabetes diagnosis with healthcare big data clouds [Journal Article]. *IEEE Communications Magazine*, *56*(4), 16-23. doi: 10.1109/mcom.2018.1700788

Chen, Y.-S. (2018). E-business and big data strategy in franchising [Book Section]. In *Encyclopedia of information science and technology, fourth edition* (p. 2686-2696). IGI Global. doi: 10.4018/978-1-5225-2255-3.ch234

Cherryholmes, C. H. (1992). Notes on pragmatism and scientific realism. *Educational researcher*, *21*(6), 13–17.

Chua, W. F. (1986). Radical developments in accounting thought. *Accounting review*, 601–632.

Clandinin, D. J. & Connelly, F. M. (2004). *Narrative inquiry: Experience and story in qualitative research*. John Wiley & Sons.

Clements, P., Garlan, D., Little, R., Nord, R. & Stafford, J. (2003). Documenting software architectures: views and beyond. In *25th international conference on software engineering, 2003. proceedings.* (pp. 740–741).

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E. & Bone, M. (2010a). The concept of reference architectures [Journal Article]. *Systems Engineering*, *13*(1), 14-27.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E. & Bone, M. (2010b). The concept of reference architectures [Journal Article]. *Systems Engineering*, *13*(1), 14-27. doi: 10.2514/6.2017-5118

Coblenz, M., Sunshine, J., Aldrich, J., Myers, B. A., Weber, S. & Shull, F. (2016). Exploring language support for immutability. *Proceedings of the 38th International Conference on Software Engineering*. doi: 10.1145/2884781.2884798

Cohen, J. (1960). A coefficient of agreement for nominal scales [Journal Article]. *Educational and Psychological Measurement*, *20*(1), 37–46. doi: 10.1177/001316446002000104

Collaborative, N. (2023). *U.s. climate resilience toolkit: Climate explorer.* (Available from: `https://crt-climate-explorer.nemac.org/`)

Company, T. M. D. (2019-2024). *The modern data company - homepage.* Retrieved from https://www.moderndata.com/

Comuzzi, M. & Patel, A. (2016). How organisations leverage big data: A maturity model [Journal Article]. *Industrial management and Data systems*, *116*(8), 1468-1492. doi: 10.1108/imds-12-2015-0495

Confluent. (2024). *Apache kafka documentation: Monitoring data latency.* https://docs.confluent.io/platform/current/kafka/monitoring.html. (Accessed: 2023-06-05)

Cook, G. A. (1993). *George herbert mead: The making of a social pragmatist*. University of Illinois Press.

Corbin, J. & Strauss, A. (2014). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications.

Coulouris, G., Dollimore, J. & Kindberg, T. (2005). *Distributed systems: Concepts and design* (4th ed.). Addison-Wesley.

Crampton, J. W. (2015). Collect it all: National security, big data and governance [Journal Article]. *GeoJournal*, *80*(4), 519-531. doi: 10.1177/2053951716661366

Creswell, J. W. & Creswell, J. D. (2017). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.

Creswell, J. W., Hanson, W. E., Clark Plano, V. L. & Morales, A. (2007). Qualitative research designs: Selection and implementation. *The counseling psychologist*, *35*(2), 236–264.

Critical Appraisal Skills Programme. (2023). *Casp checklists.* https://casp-uk.net/casp-tools-checklists/.

Crotty, M. (1998). The foundations of social research: meaning and perspective in the research process sage. *Thousand Oaks, CA Google Scholar*.

Cruzes, D. S. & Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement* (pp. 275–284).

Cumpston, M., Li, T., Page, M. J., Chandler, J., Welch, V. A., Higgins, J. P. & Thomas, J. (2019). Updated guidance for trusted systematic reviews: a new edition of the cochrane handbook for systematic reviews of interventions. *Cochrane Database Syst Rev*, *10*(10.1002), 14651858.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K. & Wasowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of the sixth international workshop on variability modeling of software-intensive systems* (pp. 173–182).

Davenport, T. H., Barth, P. & Bean, R. (2012). *How'big data'is different* [Book]. MIT Sloan Management Review. doi: 10.15358/9783800648153-85-1

Davis, F. D. (1985). *A technology acceptance model for empirically testing new end-user information systems: Theory and results* (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Dehghani, Z. (2019). *How to move beyond a monolithic data lake to a distributed data mesh.* martinfowler.com. Retrieved from https://martinfowler.com/articles/data-monolith-to-mesh.html

Dehghani, Z. (2020). Data mesh: A new paradigm for data-driven organizations. *ThoughtWorks*.

Dehghani, Z. (2022). *Data mesh: Delivering data-driven value at scale*. O'Reilly Media. Retrieved from https://www.amazon.com/Data-Mesh-Delivering-Data-Driven-Value/dp/1492092398/ref=sr_1_1?crid=5YWKS24L2BK4&keywords=data+mesh&qid=1647136984&sprefix=dat%2Caps%2C646&sr=8-1

Demchenko, Y., Grosso, P., De Laat, C. & Membrey, P. (2013). Addressing big data issues in scientific data infrastructure [Conference Proceedings]. In *2013 international conference on collaboration technologies and systems (cts)* (p. 48-55). IEEE. doi: 10.1109/cts.2013.6567203

Denning, P. J. (1997). A new social contract for research. *Communications of the ACM*, *40*(2), 132–134.

Denzin, N. K. (2004). Symbolic interactionism. *A companion to qualitative research*, 81–87.

Denzin, N. K. & Lincoln, Y. S. (2011). *The sage handbook of qualitative research*. sage.

Derras, M., Deruelle, L., Douin, J. M., Levy, N., Losavio, F., Pollet, Y. & Reiner, V. (2018a). Reference architecture design: a practical approach. In *13th international conference on software technologies (icsoft)* (pp. 633–640).

Derras, M., Deruelle, L., Douin, J. M., Levy, N., Losavio, F., Pollet, Y. & Reiner, V. (2018b). Reference architecture design: a practical approach. In *13th international conference on software technologies (icsoft)* (pp. 633–640).

Dewey, J. et al. (1917). The need for a recovery of philosophy. *Creative intelligence: Essays in the pragmatic attitude*, 3–69.

Dezfouli, M. B., Shahraki, M. H. N. & Zamani, H. (2018). A novel tour planning model using big data [Conference Proceedings]. In *2018 international conference on artificial intelligence and data processing (idap)* (p. 1-6). IEEE. doi: 10.1109/idap.2018.8620933

Dodds, P. S., Harris, K. D., Kloumann, I. M., Bliss, C. A. & Danforth, C. M. (2011). Temporal patterns of happiness and information in a global social network: hedonometrics and twitter [Journal Article]. *PLoS One*, *6*(12), e26752. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/22163266 doi: 10.1371/journal.pone.0026752

Durahim, A. O. & Coşkun, M. (2015). iamhappybecause: Gross national happiness through twitter analysis and big data [Journal Article]. *Technological Forecasting and Social Change*, *99*, 92-105. doi: 10.1016/j.techfore.2015.06.035

Dwork, C. (2006). Differential privacy. In *Automata, languages and programming: 33rd international colloquium, icalp 2006, venice, italy, july 10-14, 2006, proceedings, part ii 33* (pp. 1–12).

Dybå, T. & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, *50*(9-10), 833–859.

Ehtisham Zaidi, G. D. S. (2019). *Augmented data catalogs: Now an enterprise must-have for data and analytics leaders* (Tech. Rep.). Gartner.

Elmasri, R. (2017). *Fundamentals of database systems* [Book]. doi: 10.1007/978-1-4614-8265-9_80674

Engelsman, W., Quartel, D., Jonkers, H. & van Sinderen, M. (2011). Extending enterprise architecture modelling with business goals and requirements. *Enterprise information systems*, *5*(1), 9–36.

Eridaputra, H., Hendradjaya, B. & Sunindyo, W. D. (2014). Modeling the requirements for big data application using goal oriented approach. In *2014 international conference on data and software engineering (icodse)* (pp. 1–6).

Eryurek, E., Gilad, U., Lakshmanan, V., Kibunguchy-Grant, A. & Ashdown, J. (2021). *Data governance: The definitive guide: People, processes, and tools to operationalize data trustworthiness*. O'Reilly Media.

Estdale, J. & Georgiadou, E. (2018). Applying the iso/iec 25010 quality models to software product. In *European conference on software process improvement* (pp. 492–503).

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Fetterman, D. M. (2019). *Ethnography: Step-by-step*. Sage publications.

Firouzi, F., Rahmani, A. M., Mankodiya, K., Badaroglu, M., Merrett, G. V., Wong, P. & Farahani, B. (2018). *Internet-of-things and big data for smarter healthcare: From device to architecture, applications and analytics* (Vol. 78). Elsevier. doi: 10.1016/j.future.2017.09.016

Fischer, C., Winter, R. & Wortmann, F. (2010). Design theory [Journal Article]. *Business and Information Systems Engineering*, *2*(6), 387-390. doi: 10.1007/s12599-010-0128-2

Ford, N., Parsons, R. & Kua, P. (2022). *Building evolutionary architectures: Support constant change* (2nd ed.). O'Reilly Media. Retrieved from https://www.amazon.com/Building-Evolutionary-Architectures-Automated-Governance/dp/1098100639

Ford, N., Richards, M., Sadalage, P. & Dehghani, Z. (2021). *Software architecture: The hard parts*. " O'Reilly Media, Inc.".

for Standardization (ISO/IEC), I. O. (2020). *Iso/iec tr 20547-1:2020.* Retrieved from https://www.iso.org/standard/71275.html

Foundation, A. S. (2021). *Apache beam.* https://beam.apache.org/. (Accessed: April 8, 2023)

Fowler, M. (2012). *Patterns of enterprise application architecture*. Addison-Wesley.

G2. (2023). *G2 software categories.* Retrieved from https://www.g2.com/categories (Accessed: 2023-06-03)

Galster, M. & Avgeriou, P. (2011a). Empirically-grounded reference architectures: a proposal. *Joint ACM*. doi: 10.1145/2000259.2000285

Galster, M. & Avgeriou, P. (2011b). Empirically-grounded reference architectures: a proposal [Conference Proceedings]. In *Proceedings of the joint acm sigsoft conference–qosa and acm sigsoft symposium–isarcs on quality of software architectures–qosa and architecting critical systems–isarcs* (p. 153-158). ACM. doi: 10.1145/2000259.2000285

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., ... others (2019). An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems* (pp. 3–18).

Gartner. (2021). *Magic quadrant.* `https://www.gartner.com/en/research/methodologies/research-methodologies-and-processes/magic-quadrants-research`. (Accessed: March 26, 2023)

Geerdink, B. (2013). A reference architecture for big data solutions introducing a model to perform predictive analytics using big data technology. In *8th international conference for internet technology and secured transactions (icitst-2013)* (pp. 71–76).

GitHub. (2023). *Explore github.* Retrieved from `https://github.com/explore` (Accessed: 2023-06-03)

GitOps. (2023). `https://www.gitops.tech/`. (Accessed: April 19, 2023)

Gohil, A., Modi, H. & Patel, S. K. (2013). 5g technology of mobile communication: A survey [Conference Proceedings]. In *2013 international conference on intelligent systems and signal processing (issp)* (p. 288-292). IEEE. doi: 10.1109/issp.2013.6526920

Goldman, O. (2024). *Effective software architecture: Building better software faster* (1st ed.). United States: Addison-Wesley Professional.

Gorelik, A. (2019). *The enterprise big data lake: Delivering the promise of big data and data science*. O'Reilly Media.

Gorton, I. & Klein, J. (2015). Distribution, data, deployment [Journal Article]. *STC 2015*, 78.

Graaf, B., Van Dijk, H. & Van Deursen, A. (2005). Evaluating an embedded software reference architecture-industrial experience report. In *Ninth european conference on software maintenance and reengineering* (pp. 354–363).

Greefhorst, D. (1999). Een applicatie-architectuur voor het web bij de bank—de pro's en contra's van toestandsloosheid. *Software Release Magazine*, *2*.

Greefhorst, D. & Gehner, P. (2006). Achmea streamlines application development and integration. *Via Nova Architectura*.

Gregg, B. (2014). *Systems performance: enterprise and the cloud*. Pearson Education.

Guo, L. & Vargo, C. (2015). The power of message networks: A big-data analysis of the network agenda setting model and issue ownership [Journal Article]. *Mass Communication and Society*, *18*(5), 557-576. doi: 10.1080/15205436.2015.1045300

Haller, P. & Axelsson, L. (2017). Quantifying and explaining immutability in scala. *Electronic Proceedings in Theoretical Computer Science*, *246*, 21-27. doi: 10.4204/eptcs.246.5

Han, J., Haihong, E., Le, G. & Du, J. (2011). Survey on nosql database [Conference Proceedings]. In *2011 6th international conference on pervasive computing and applications* (p. 363-366). IEEE. doi: 10.1109/icpca.2011.6106531

HashiCorp. (2021). *Terraform.* https://www.terraform.io/docs/modules/index.html. ([Online; accessed 23-April-2023])

HashiCorp. (2023a). *Terraform.* Retrieved from https://www.terraform.io/

HashiCorp. (2023b). *Vault helm chart.* Retrieved from https://artifacthub.io/packages/helm/hashicorp/vault

Hechler, E., Weihrauch, M. & Wu, Y. (2023). Data fabric and data mesh research areas. In *Data fabric and data mesh approaches with ai: A guide to ai-based data cataloging, governance, integration, orchestration, and consumption* (pp. 375–392). Springer.

Heilig, L. & Voß, S. (2017). Managing cloud-based big data platforms: a reference architecture and cost perspective. In *Big data management* (pp. 29–45). Springer.

Henke, N., Bughin, J., Chui, M., Manyika, J., Saleh, T., Wiseman, B. & Sethupathy, G. (2016). *The age of analytics: Competing in a data-driven world* (Tech. Rep.). McKinsey & Company. Retrieved from https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/the-age-of-analytics-competing-in-a-data-driven-world (Accessed: [insert date here])

Hevner, A. & Chatterjee, S. (2010). Design science research in information systems [Book Section]. In *Design research in information systems* (p. 9-22). Springer. doi: 10.1007/978-1-4419-5653-8_2

Hevner, A. R., Chatterjee, S. & Galletta, D. F. (2010). Research in information systems: An empirical study of diversity in the discipline and its journals. *Journal of Management Information Systems*, *27*(1), 273–308. doi: 10.2753/MIS0742-1222270111

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 75–105.

Hoffman, R. R. (1997). The cognitive psychology of expertise and the domain of interpreting. *Interpreting*, *2*(1-2), 189–230.

Holland, J. H. (1992). Complex adaptive systems. *Daedalus*, *121*(1), 17–30.

Hollingsworth, D. & Hampshire, U. (1995). Workflow management coalition: The workflow reference model. *Document Number TC00-1003*, *19*(16), 224.

Huberty, M. (2015). Awaiting the second big data revolution: from digital noise to value creation [Journal Article]. *Journal of Industry, Competition and Trade*, *15*(1), 35-47.

IEEE access. (2024). *IEEE Access.* Retrieved from https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=6585654

Iglesias, C. A., Favenza, A. & Carrera, Á. (2020). A big data reference architecture for emergency management. *Information*, *11*(12), 569.

Inc., D. (2021). *Databricks.* https://databricks.com/. (Accessed: April 8, 2023)

International Organization for Standardization (ISO/IEC), I. (2017). *Iso/iec/ieee 42010:2011.* Retrieved from https://www.iso.org/standard/50508.html

ISO. (2019). Iso 19115-1:2014. *International Organization for Standardization*.

Iso, I. (2011). Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models [Journal Article]. *International Organization for Standardization*, *34*, 2910.

ISO, I. (2016). Information technology — reference architecture for service oriented architecture (soa ra) — part 1: Terminology and concepts for soa. *International Organization for Standardization*, 51. Retrieved from `https://www.iso.org/standard/63104.html`

ISO/IEC. (2018). *Iso/iec 29148:2018. systems and software engineering — life cycle processes — requirements engineering* [Standard]. Retrieved from `https://www.iso.org/standard/72089.html`

Iso/iec 25000:2005. software engineering — software product quality requirements and evaluation (square) — guide to square [Standard]. (2014).

ISO/IEC 26550: 2015-software and systems engineering–reference model for product line engineering and management [Standard]. (2015).

Istio. (2018). *Istio: An open platform to connect, manage, and secure microservices.* `https://istio.io/`.

Jagadish, H., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R. & Shahabi, C. (2014). Big data and its technical challenges [Journal Article]. *Communications of the ACM*, *57*(7), 86-94. doi: 10.1145/2611567

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R. & Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, *57*(7), 86–94.

James, W. (1981). Pragmatism: A new name for some old ways of thinking (b. kuklick, ed.). *Indianapolis, IN: Hackett.(Original work published 1907)*.

Jin, S., Lin, W., Yin, H., Yang, S., Li, A. & Deng, B. (2015). Community structure mining in big data social media networks with mapreduce [Journal Article]. *Cluster computing*, *18*(3), 999-1010. doi: 10.1007/s10586-015-0452-x

Jin, X., Wah, B. W., Cheng, X. & Wang, Y. (2015). Significance and challenges of big data research. *Big data research*, *2*(2), 59–64.

Johnson, M. (2019). *Principles of logic in computer science*. TechPress.

Josey, A. (2016). *Togaf® version 9.1-a pocket guide*. Van Haren.

Josey, A., Lankhorst, M., Band, I., Jonkers, H. & Quartel, D. (2016). An introduction to the archimate® 3.0 specification. *White Paper from The Open Group*.

JSON Schema. (2019). `https://json-schema.org/`. (Accessed: April 30, 2023)

Kaisler, S., Armour, F., Espinosa, J. A. & Money, W. (2013). Big data: Issues and challenges moving forward [Conference Proceedings]. In *2013 46th hawaii international conference on system sciences* (p. 995-1004). IEEE. doi: 10.1109/hicss.2013.645

Kakivaya, G., Xun, L., Hasha, R., Ahsan, S. B., Pfleiger, T., Sinha, R., ... others (2018). Service fabric: a distributed platform for building microservices in the cloud. In *Proceedings of the thirteenth eurosys conference* (pp. 1–15).

Kalil, T. (2012). *Big data is a big deal.* Retrieved from `https://`

obamawhitehouse.archives.gov/blog/2012/03/29/big-data-big-deal

Kallio, H., Pietilä, A.-M., Johnson, M. & Kangasniemi, M. (2016). Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *Journal of advanced nursing*, *72*(12), 2954–2965.

Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., ... others (2023). Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, *103*, 102274.

Kazman, R., Bass, L., Abowd, G. & Webb, M. (1994). Saam: A method for analyzing the properties of software architectures. In *Proceedings of 16th international conference on software engineering* (pp. 81–90).

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H. & Carriere, J. (1998a). The architecture tradeoff analysis method [Conference Proceedings]. In *Proceedings. fourth ieee international conference on engineering of complex computer systems (cat. no. 98ex193)* (p. 68-78). IEEE.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H. & Carriere, J. (1998b). The architecture tradeoff analysis method [Conference Proceedings]. In *Proceedings. fourth ieee international conference on engineering of complex computer systems (cat. no. 98ex193)* (p. 68-78). IEEE. doi: 10.21236/ada350761

Khine, P. & Wang, Z. (2019). A review of polyglot persistence in the big data world. *Information*, *10*, 141. doi: 10.3390/info10040141

Khine, P. P. & Wang, Z. (2019). A review of polyglot persistence in the big data world. *Information*, *10*(4), 141.

Khrononov, S. (2021). *Learning domain-driven design: Aligning your architecture with the business using context maps, strategic design, and agile techniques*. Birmingham, UK: Packt Publishing. Retrieved from https://www.amazon.com/Learning-Domain-Driven-Design-Aligning-Architecture/dp/1098100131

Kimball, R. & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling*. Wiley.

Kiran, M., Murphy, P., Monga, I., Dugan, J. & Baveja, S. S. (2015a). Lambda architecture for cost-effective batch and speed big data processing. In *2015 ieee international conference on big data (big data)* (pp. 2785–2792).

Kiran, M., Murphy, P., Monga, I., Dugan, J. & Baveja, S. S. (2015b). Lambda architecture for cost-effective batch and speed big data processing [Conference Proceedings]. In *2015 ieee international conference on big data (big data)* (p. 2785-2792). IEEE. doi: 10.1109/bigdata.2015.7364082

Kitchenham, B. A., Budgen, D. & Brereton, P. (2015). *Evidence-based software engineering and systematic reviews* (Vol. 4). CRC press.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K. & Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering*, *28*(8), 721–734.

Klein, J., Buglak, R., Blockow, D., Wuttke, T. & Cooper, B. (n.d.). A reference

architecture for big data systems in the national security domain [Conference Proceedings]. In *2016 ieee/acm 2nd international workshop on big data software engineering (bigdse)* (p. 51-57). IEEE.

Klein, J., Buglak, R., Blockow, D., Wuttke, T. & Cooper, B. (2016). A reference architecture for big data systems in the national security domain. In *2016 ieee/acm 2nd international workshop on big data software engineering (bigdse)* (pp. 51–57).

Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems* [Book]. " O'Reilly Media, Inc.". doi: 10.1007/978-3-319-77525-8_197

Kohler, J. & Specht, T. (2019). Towards a secure, distributed, and reliable cloud-based reference architecture for big data in smart cities. In *Big data analytics for smart and connected cities* (pp. 38–70). IGI Global.

Kreps, J. (2014a). *Questioning the lambda architecture.* Blog post. Retrieved from https://www.oreilly.com/radar/questioning-the-lambda-architecture/

Kreps, J. (2014b). Questioning the lambda architecture. *Online article, July*, *205*. Retrieved from https://www.oreilly.com/radar/questioning-the-lambda-architecture/

Kreps, J. (2023). *Benchmarking apache kafka: 2 million writes per second (on three cheap machines).* https://www.confluent.io/blog/benchmarking-apache-kafka-2-million-writes-per-second-on-three-cheap-machines/. (Accessed: 2023-06-05)

Krippendorff, K. (1970). Estimating the reliability, systematic error and random error of interval data [Journal Article]. *Educational and Psychological Measurement*, *30*(1), 61–70. doi: 10.1177/001316447003000106

Krishnan, K. (2013). *Data warehousing in the age of big data*. Morgan Kaufmann.

Kubernetes Special Interest Group, S. (2023). *kind - kubernetes in docker.* Retrieved from https://kind.sigs.k8s.io/

Laigner, R., Zhou, Y., Salles, M. A. V., Liu, Y. & Kalinowski, M. (2021a). Data management in microservices: State of the practice, challenges, and research directions. *Proceedings of the VLDB Endowment*, *14*(13), 3348–3361. doi: 10.14778/3484224.3484232

Laigner, R., Zhou, Y., Salles, M. A. V., Liu, Y. & Kalinowski, M. (2021b). Data management in microservices: State of the practice, challenges, and research directions. *arXiv preprint arXiv:2103.00170*.

Lankhorst, M. (2013). A language for enterprise modelling. In *Enterprise architecture at work* (pp. 75–114). Springer.

Lankhorst, M. M., Proper, H. A. & Jonkers, H. (2010). The anatomy of the archimate language. *International Journal of Information System Modeling and Design (IJISMD)*, *1*(1), 1–32.

Laplante, P. A. (2017). *Requirements engineering for software and systems*. Auerbach Publications.

La Rosa, M., van der Aalst, W. M., Dumas, M. & Ter Hofstede, A. H. (2009).

Questionnaire-based variability modeling for system configuration. *Software & Systems Modeling*, *8*(2), 251–274.

Lee, A. (2000). Systems thinking, design science, and paradigms: Heeding three lessons from the past to resolve three dilemmas in the present to direct a trajectory for future research in the information systems field,"keynote address. In *Eleventh international conference on information management, taiwan.*

Lee, K. & Wang, L. (2020). Ensuring data consistency in big data systems. In *Proceedings of the 2020 international conference on big data* (p. 456-467).

Len Bass, R. K., Dr. Paul Clements. (2021). *Software architecture in practice (sei series in software engineering) 4th edition* [Book]. Addison-Wesley Professional; 4th edition.

Leonard, E. M. (2011). *Design and implementation of an enterprise data warehouse*. Marquette University.

Levin, B. (2013). Big data ecosystem reference architecture. *Microsoft Corporation*.

Li, S.-H., Yen, D. C., Lu, W.-H. & Wang, C. (2012). Identifying the signs of fraudulent accounts using data mining techniques [Journal Article]. *Computers in Human Behavior*, *28*(3), 1002-1013. doi: 10.1016/j.chb.2012.01.002

Liao, C., Squicciarini, A. & Griffin, C. (2015). Epidemic behavior of negative users in online social sites [Conference Proceedings]. In *Proceedings of the 5th acm conference on data and application security and privacy* (p. 143-145). ACM. doi: 10.1145/2699026.2699129

Lin, J. (2017). The lambda and the kappa. *IEEE Internet Computing*, *21*(05), 60–66.

Lin, K., Xia, F., Wang, W., Tian, D. & Song, J. (2016). System design for big data application in emotion-aware healthcare [Journal Article]. *IEEE Access*, *4*, 6901-6909. doi: 10.1109/access.2016.2616643

Lincoln, Y. S. & Guba, E. G. (1985). *Naturalistic inquiry*. sage.

Lincoln, Y. S., Lynham, S. A., Guba, E. G. et al. (2011). Paradigmatic controversies, contradictions, and emerging confluences, revisited. *The Sage handbook of qualitative research*, *4*(2), 97–128.

LinkedIn. (2020). *Linkedin emerging jobs report.* Retrieved from `https://business.linkedin.com/content/dam/me/business/en-us/talent-solutions/emerging-jobs-report/Emerging_Jobs_Report_U.S._FINAL.pdf`

Luo, J., Wu, M., Gopukumar, D. & Zhao, Y. (2016). Big data application in biomedical research and health care: a literature review. *Biomedical informatics insights*, *8*, BII–S31559.

Lycett, M. (2013). *'datafication': making sense of (big) data in a complex world* (Vol. 22). Taylor & Francis. doi: 10.1057/ejis.2013.10

Maamouri, A., Sfaxi, L. & Robbana, R. (2021). Phi: A generic microservices-based big data architecture. In *European, mediterranean, and middle eastern conference on information systems* (pp. 3–16).

Magnani, L. (2011). *Abduction, reason and science: Processes of discovery and explanation*. Springer Science & Business Media.

Maier, M., Serebrenik, A. & Vanderfeesten, I. (2013). Towards a big data reference architecture [Journal Article]. *University of Eindhoven*.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. & Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity [Journal Article]. , *3*. doi: 10.1186/s40537-015-0014-3

March, S. T. & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, *15*(4), 251–266.

Markus, M. L., Majchrzak, A. & Gasser, L. (2002a). A design theory for systems that support emergent knowledge processes [Journal Article]. *MIS quarterly*, *43*, 179-212. doi: 10.1016/j.dss.2006.09.005

Markus, M. L., Majchrzak, A. & Gasser, L. (2002b). A design theory for systems that support emergent knowledge processes. *MIS quarterly*, 179–212.

Márquez, G. & Astudillo, H. (2018). Actual use of architectural patterns in microservices-based open source projects. In *2018 25th asia-pacific software engineering conference (apsec)* (pp. 31–40).

Marquez, G. & Astudillo, H. (2018). Actual use of architectural patterns in microservices-based open source projects. In *2018 25th asia-pacific software engineering conference (apsec)* (pp. 31–40). IEEE. doi: 10.1109/APSEC.2018.00017

Marr, B. (2016). *Big data in practice: how 45 successful companies used big data analytics to deliver extraordinary results* [Book]. John Wiley and Sons. doi: 10.1109/bigdata.2018.8622333

Martinez-Prieto, M. A., Cuesta, C. E., Arias, M. & Fernández, J. D. (2015). The solid architecture for real-time management of big semantic data. *Future Generation Computer Systems*, *47*, 62–79.

Marz, N. & Warren, J. (2015). *Big data: Principles and best practices of scalable real-time data systems* [Book]. New York; Manning Publications Co. doi: 10.1109/tcss.2020.2995497

McAfee, A. & Brynjolfsson, E. (2012). Big data: the management revolution [Journal Article]. *Harv Bus Rev*, *90*(10), 60-6, 68, 128. Retrieved from `https://www.ncbi.nlm.nih.gov/pubmed/23074865`

McKinsey, G. et al. (2011). Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, 158-184. doi: 10.7591/9781501734328-007

Meadows, D. H. (2008). *Thinking in systems: A primer*. Chelsea Green Publishing.

Mehta, N. & Pandit, A. (2018). Concurrence of big data analytics and healthcare: A systematic review [Journal Article]. *Int J Med Inform*, *114*, 57-65. Retrieved from `https://www.ncbi.nlm.nih.gov/pubmed/29673604` doi: 10.1016/j.ijmedinf.2018.03.013

Merriam, S. B. & Grenier, R. S. (2019). *Qualitative research in practice: Examples for discussion and analysis*. John Wiley & Sons.

Mertens, D. M. (2008). *Transformative research and evaluation*. Guilford press.

Mertens, D. M. (2019). *Research and evaluation in education and psychology: Integrating diversity with quantitative, qualitative, and mixed methods*. Sage publications.

Mikalef, P., Pappas, I. O., Krogstie, J. & Giannakos, M. (2018). Big data analytics capabilities: a systematic literature review and research agenda [Journal Article]. *Information Systems and e-Business Management*, *16*(3), 547-578. doi: 10.1109/educon.2018.8363273

Miles, M. B. & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. sage.

Montesi, F. & Weber, J. (2016). Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv:1609.05830*.

Morris, K. (2016). *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, Inc. ([Print])

Moses, B., Gavish, L. & Vorwerck, M. (2022). *Data quality fundamentals: A practitioner's guide to building trustworthy data pipelines* (1st ed.). O'Reilly Media.

Moustakas, C. (1994). *Phenomenological research methods*. Sage publications.

Muller, G. (2008). A reference architecture primer. *Eindhoven Univ. of Techn., Eindhoven, White paper*.

Munn, Z., Barker, T. H., Moola, S., Tufanaru, C., Stern, C., McArthur, A., ... Aromataris, E. (2020). Methodological quality of case series studies: an introduction to the jbi critical appraisal tool. *JBI evidence synthesis*, *18*(10), 2127–2133.

Murdoch, T. B. & Detsky, A. S. (2013). The inevitable application of big data to health care [Journal Article]. *JAMA*, *309*(13), 1351-2. Retrieved from `https://www.ncbi.nlm.nih.gov/pubmed/23549579` doi: 10.1001/jama.2013.393

Myers, M. D. & Avison, D. (2002). *Qualitative research in information systems: a reader*. Sage.

Nadal, S., Herrero, V., Romero, O., Abelló, A., Franch, X., Vansummeren, S. & Valerio, D. (2017). A software reference architecture for semantic-aware big data systems. *Information and software technology*, *90*, 75–92.

Nadal, S., Herrero, V., Romero, O., Abelló, A., Franch, X., Vansummeren, S. & Valerio, D. (2017). A software reference architecture for semantic-aware big data systems [Journal Article]. *Information and software technology*, *90*, 75-92.

Nakagawa, E. Y., Guessi, M., Maldonado, J. C., Feitosa, D. & Oquendo, F. (2014). Consolidating a process for the design, representation, and evaluation of reference architectures. In *2014 ieee/ifip conference on software architecture* (pp. 143–152).

Nakagawa, E. Y., Martins, R. M., Felizardo, K. R. & Maldonado, J. C. (2009). Towards a process to design aspect-oriented reference architectures [Conference Proceedings]. In *Xxxv latin american informatics conference (clei) 2009*.

Nakagawa, E. Y., Oquendo, F. & Becker, M. (2012). Ramodel: A reference model for reference architectures. In *2012 joint working ieee/ifip conference on software architecture and european conference on software architecture* (pp. 297–301).

Nakagawa, E. Y., Oquendo, F. & Maldonado, J. C. (2014). Reference architectures. *Software Architecture 1*, 55–82.

Nash, H. (2015). Cio survey 2015 [Journal Article]. *Association with KPMG*.

Nasser, T. & Tariq, R. (2015). Big data challenges. *J Comput Eng Inf Technol 4: 3. doi: http://dx. doi. org/10.4172/2324*, *9307*(2).

Neuman, L. W. (2007). *Social research methods, 6/e*. Pearson Education India.

Newman, I., Benz, C. R. & Ridenour, C. S. (1998). *Qualitative-quantitative research methodology: Exploring the interactive continuum*. SIU Press.

Newman, S. (2015a). *Building microservices: Designing fine-grained systems*. O'Reilly Media, Inc.

Newman, S. (2015b). *Building microservices: Designing fine-grained systems*. Sebastopol, CA: O'Reilly Media. Retrieved from https://www.amazon.com/Building-Microservices-Designing-Fine-Grained-Systems/dp/1492034029

NewVantage. (2021). *Big data and ai executive survey 2021*. NewVantage Partners. Retrieved from https://www.supplychain247.com/paper/bi_data_and_ai_executive_survey_2021/pragmadik

Noblit, G. W., Hare, R. D. & Hare, R. D. (1988). *Meta-ethnography: Synthesizing qualitative studies* (Vol. 11). sage.

Nogueira, I. D., Romdhane, M. & Darmont, J. (2018). Modeling data lake metadata with a data vault. In *Proceedings of the 22nd international database engineering & applications symposium* (pp. 253–261).

Norman, D. A. (2013). *The design of everyday things*. New York, NY: Basic Books.

Norta, A. H. (2007). *Exploring dynamic inter-organizational business process collaboration* (Vol. 68) (No. 04).

Ntentos, E., Zdun, U., Plakidas, K., Schall, D., Li, F. & Meixner, S. (2019). Supporting architectural decision making on data management in microservice architectures. In T. Bures, L. Duchien & P. Inverardi (Eds.), *Software architecture* (Vol. 11681, pp. 20–36). Cham: Springer International Publishing. doi: 10.1007/978-3-030-29983-5{\textunderscore}2

Nunamaker Jr, J. F., Briggs, R. O., Mittleman, D. D., Vogel, D. R. & Pierre, B. A. (1996). Lessons from a dozen years of group support systems research: A discussion of lab and field findings. *Journal of management information systems*, *13*(3), 163–207.

Nunamaker Jr, J. F., Chen, H., Purdin, T. D., Sprague Jr, R. H. & Takeda, H. (2009). Design science research in information systems. *MIS quarterly*, *33*(1), 1–22.

Nye, J. S. (2013). Governance in the information age [Journal Article]. *Governance*, *113*, 19-22. doi: 10.1525/curh.2014.113.759.19

OATH. (2007). Oath reference architecture, release 2.0 initiative for open authentication [Journal Article]. *OATH*. Retrieved from https://openauthentication.org/wp-content/uploads/2015/09/ReferenceArchitectureVersion2.pdf

Ochs, P. (1998). Peirce, pragmatism and the logic of scripture. *Cambridge: Cambridge*.

Open Web Application Security Project. (2017). *Owasp top 10 - 2017*. https://owasp.org/Top10/. (Accessed on April 23, 2023)

Oracle Corporation. (2023). Mysql: The world's most popular open source database [Computer software manual]. Retrieved from https://www.mysql.com/ (Accessed: 2023-03-01)

Orlikowski, W. J. & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information systems research*, *2*(1), 1–28.

Overflow, S. (2022). *Stack overflow developer survey 2022.* Retrieved from https://survey.stackoverflow.com/2022/

Pääkkönen, P. & Pakkala, D. (2015). Reference architecture and classification of technologies, products and services for big data systems. *Big data research*, *2*(4), 166–186.

Pääkkönen, P. & Pakkala, D. (2020). Extending reference architecture of big data systems towards machine learning in edge computing environments. *Journal of Big Data*, *7*(1), 1–29.

Page, M. J., Moher, D., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., ... others (2021). Prisma 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews. *Bmj*, *372*.

Partners, N. V. (2023). *New vantage partners annual report* (Tech. Rep.). Retrieved from https://www.newvantage.com/_files/ugd/e5361a_247885043758499ba090f7a5f510cf7c.pdf

Patton, M. Q. (2002). Qualitative research and evaluation methods. thousand oaks. *Cal.: Sage Publications*, *4*.

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45–77.

Peirce, C. S. (1878). How to make our ideas clear. *Popular Science Monthly*, *12*, 286–302.

Phillips, D. C., Phillips, D. C. & Burbules, N. C. (2000). *Postpositivism and educational research*. Rowman & Littlefield.

Piñeiro, C., Morales, J., Rodríguez, M., Aparicio, M., Manzanilla, E. G. & Koketsu, Y. (2019). Big (pig) data and the internet of the swine things: a new paradigm in the industry. *Animal frontiers*, *9*(2), 6–15.

Pohl, K., Böckle, G. & Van Der Linden, F. (2005). *Software product line engineering: foundations, principles, and techniques* (Vol. 1). Springer.

Polyhistor. (2023a). *helmchartsandapplicationsformetamycelium.* https://github.com/Polyhistor/helmChartsAndApplicationsForMetamycelium. GitHub.

Polyhistor. (2023b). *Infrastructureformetamycelium.* https://github.com/Polyhistor/InfrastructureForMetamycelium. GitHub.

Pope, A. L. (1998). *The corba reference guide: understanding the common object request broker architecture*. Addison-Wesley Longman Publishing Co., Inc.

Popescu, F., Iskandaryan, R. & Weber, T. (2019). Big data and international accreditations in higher education: A dutch–russian case study [Journal Article]. *Proceedings of the 11th International Conference on Computer Supported Education - Volume 1: CSEDU,*, 207-214. doi: 10.5220/0007572102070214

Popovič, A., Hackney, R., Tassabehji, R. & Castelli, M. (2018). The impact of big data analytics on firms' high value business performance [Journal Article]. *Information*

Systems Frontiers, *20*(2), 209-222. doi: 10.1016/j.jbusres.2018.12.072

Pourmirza, S., Peters, S., Dijkman, R. & Grefen, P. (2017). A systematic literature review on the architecture of business process management systems. *Information Systems*, *66*, 43–58.

Press, M., Joyner, L. & Malcolm, G. (2002). *Application architecture for. net: Designing applications and services*. Microsoft Press.

Presto. (2023). *Presto: Distributed sql query engine for big data.* Retrieved from `https://prestosql.io/`

Qin, S., Man, J., Wang, X., Li, C., Dong, H. & Ge, X. (2019). Applying big data analytics to monitor tourist flow for the scenic area operation management [Journal Article]. *Discrete Dynamics in Nature and Society*, *2019*, 1-11. doi: 10.1155/2019/8239047

Quintero, D., Lee, F. N. et al. (2019). *Ibm reference architecture for high performance data and ai in healthcare and life sciences*. IBM Redbooks.

Rad, B. B. & Ataei, P. (2017a). The big data ecosystem and its environs [Journal Article]. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 38.

Rad, B. B. & Ataei, P. (2017b). The big data ecosystem and its environs. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 38.

Rad, B. B. & Ataei, P. (2017c). The big data ecosystem and its environs [Journal Article]. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 38. doi: 10.1007/978-3-319-94301-5_16

Rada, B. B., Ataeib, P., Khakbizc, Y. & Akbarzadehd, N. (2017a). The hype of emerging technologies: Big data as a service. *Int. J. Control Theory Appl*, *9*(43), 1–18.

Rada, B. B., Ataeib, P., Khakbizc, Y. & Akbarzadehd, N. (2017b). The hype of emerging technologies: Big data as a service [Journal Article]. *Int. J. Control Theory Appl*.

Rahimi, S. K. & Haug, F. S. (2010). *Distributed database management systems: A practical approach*. John Wiley & Sons.

Rainer, R. K., Prince, B., Sánchez-Rodríguez, C., Splettstoesser-Hogeterp, I. & Ebrahimi, S. (2020). *Introduction to information systems*. John Wiley & Sons.

Ranjan, J. (2019). The 10 vs of big data framework in the context of 5 industry verticals [Journal Article]. *Productivity*, *59*(4), 324-342. doi: 10.32381/prod.2019.59.04.2

Reis, J. & Housley, M. (2022). *Fundamentals of data engineering*. " O'Reilly Media, Inc.".

Rethlefsen, M. L., Kirtley, S., Waffenschmidt, S., Ayala, A. P., Moher, D., Page, M. J. & Koffel, J. B. (2021). Prisma-s: an extension to the prisma statement for reporting literature searches in systematic reviews. *Systematic reviews*, *10*(1), 1–19.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on computer graphics and interactive techniques* (pp. 25–34).

Richards, M. & Ford, N. (2020). *Fundamentals of software architecture: an engineering approach*. O'Reilly Media.

Richardson, C. (2018). *Microservices patterns: with examples in java*. Simon and Schuster.

Richardson, C. (2022). *A pattern language for microservices.* https://microservices.io. Retrieved 07.08.2022, from `https://microservices.io/patterns/index.html`

Rittel, H. (1984). Planning problems are wicked problems. *Developments in design methodology*, 135–144.

Rodríguez-Mazahua, L., Rodríguez-Enríquez, C.-A., Sánchez-Cervantes, J. L., Cervantes, J., García-Alcaraz, J. L. & Alor-Hernández, G. (2016). A general perspective of big data: applications, tools, challenges and trends. *The Journal of Supercomputing*, *72*, 3073–3113.

Rohling, A. J., Neto, V. V. G., Ferreira, M. G. V., Dos Santos, W. A. & Nakagawa, E. Y. (2019). A reference architecture for satellite control systems. *Innovations in Systems and Software Engineering*, *15*(2), 139–153.

Rorty, R. (1990). Pragmatism as anti-representationalism. *Pragmatism: From Peirce to Davidson*, 1–6.

Rosemann, M. & Van der Aalst, W. M. (2007). A configurable reference modelling language. *Information systems*, *32*(1), 1–23.

Runeson, P., Andersson, C., Thelin, T., Andrews, A. & Berling, T. (2006). What do we know about defect detection methods?[software testing]. *IEEE software*, *23*(3), 82–90.

Rurua, N., Eshuis, R. & Razavian, M. (2019). Representing variability in enterprise architecture. *Business & Information Systems Engineering*, *61*(2), 215–227.

Russom, P. (2014). Evolving data warehouse architectures. *The Data Warehousing Institute (TDWI)*.

Sabt, M., Achemlal, M. & Bouabdallah, A. (2015). Trusted execution environment: what it is, and what it is not. In *2015 ieee trustcom/bigdatase/ispa* (Vol. 1, pp. 57–64).

Sadalage, P. J. & Fowler, M. (2013). *Nosql distilled: a brief guide to the emerging world of polyglot persistence* [Book]. Pearson Education. doi: 10.1109/aiccsa.2015.7507130

Sagiroglu, S. & Sinanc, D. (2013). Big data: A review [Conference Proceedings]. In *2013 international conference on collaboration technologies and systems (cts)* (p. 42-47). IEEE. doi: 10.1109/cts.2013.6567202

Sang, G. M., Xu, L. & Vrieze, P. d. (2017). Simplifying big data analytics systems with a reference architecture. In *Working conference on virtual enterprises* (pp. 242–249).

SAP. (2016). *Sap - nec reference architecture for sap hana & hadoop* (Tech. Rep.). Author. Retrieved from `https://www.scribd.com/document/418835912/Whitepaper-NEC-SAPHANA-Hadoop`

schema.org. (2011). `https://schema.org/`. ([Online; accessed 1 May 2023])

Serra, J. (2024a). *Deciphering data architectures: Choosing between a modern data warehouse, data fabric, data lakehouse, and data mesh* (1st ed.). O'Reilly Media.

Serra, J. (2024b). *Deciphering data architectures: From data warehouse to data lakehouse*. Oreilly & Associates Inc. Retrieved from https://www.amazon.com/Deciphering-Data-Architectures-Warehouse-Lakehouse/dp/1098150767

Sevilla, C. G. (1992). *Research methods*. Rex Bookstore, Inc.

Shamseer, L., Moher, D., Clarke, M., Ghersi, D., Liberati, A., Petticrew, M., ... Stewart, L. A. (2015). Preferred reporting items for systematic review and meta-analysis protocols (prisma-p) 2015: elaboration and explanation [Journal Article]. *Bmj*, *349*.

Sharpe, R., Van Lopik, K., Neal, A., Goodall, P., Conway, P. P. & West, A. A. (2019). An industrial evaluation of an industry 4.0 reference architecture demonstrating the need for the inclusion of security and human components. *Computers in industry*, *108*, 37–44.

Shrivastava, S., Srivastav, N., Sheth, R., Karmarkar, R. & Arora, K. (2022). *Solutions architect's handbook: Kick-start your career as a solutions architect by learning architecture design principles and strategies*. Packt Publishing Ltd.

Sievi-Korte, O., Richardson, I. & Beecham, S. (2019). Software architecture design in global software development: An empirical study [Journal Article]. *Journal of Systems and Software*, *158*, 110400.

Sigala, M. (2019). *Big data and innovation in tourism, travel, and hospitality: Managerial approaches, techniques, and applications* [Book]. ieeexplore. doi: 10.1007/978-981-13-6339-9_4

Sikora-Wohlfeld, W., Basu, A., Butte, A. & Martinez-Canales, M. (2014, 09). Accelerating secondary genome analysis using intel big data reference architecture. *Intel*.

Silver, M. S., Markus, M. L. & Beath, C. M. (1995). The information technology interaction model: A foundation for the mba core course. *MIS quarterly*, 361–390.

Simon, H. A. (1969). The sciences of the artificial. *Cambridge, MA*.

Simon, H. A. (1981). *The sciences of the artificial*. MIT Press.

Singh, N., Lai, K.-H., Vejvar, M. & Cheng, T. (2019). Big data technology: Challenges, prospects and realities [Journal Article]. *IEEE Engineering Management Review*.

Sinnema, M., Deelstra, S. & Hoekstra, P. (2006). The covamof derivation process. In *International conference on software reuse* (pp. 101–114).

Skelton, M. & Pais, M. (2019). *Team topologies: Organizing business and technology teams for fast flow*. IT Revolution.

Slife, B. D., Williams, R. N. & Williams, R. N. (1995). *What's behind the research?: Discovering hidden assumptions in the behavioral sciences*. Sage.

Smith, A. & Johnson, B. (2022). High data load thresholds in modern architectures. *Data Warehouse and Analytics Journal*, *47*(3), 256-279.

Smith, J. K. (1983). Quantitative versus qualitative research: An attempt to clarify the issue. *Educational researcher*, *12*(3), 6–13.

Snodgrass, R. T. (1999). *Developing time-oriented database applications in sql*. Morgan Kaufmann Publishers.

Sommerville, I. (2011). *Software engineering, 9/e*. Pearson Education India.

Srinivasan, U. & Arunasalam, B. (2013). Leveraging big data analytics to reduce healthcare costs [Journal Article]. *IT professional*, *15*(6), 21-28. doi: 10.1109/mitp.2013.55

Sriraman, A. & Wenisch, T. F. (2018). μ suite: a benchmark suite for microservices. In *2018 ieee international symposium on workload characterization (iiswc)* (pp. 1–12).

Srivastava, R. (2018). Big data: Issues and challenges [Journal Article]. *International Journal Of Scientific And Innovative Research*(6), 1. doi: 10.1007/978-981-13-8759-3_10

StackShare. (2023). *Explore stackshare.* Retrieved from `https://stackshare.io/stacks` (Accessed: 2023-06-03)

Stake, R. E. (1995). *The art of case study research*. sage.

State of data mesh 2022. (2022). ThoughtWorks. Retrieved from `https://www.thoughtworks.com/about-us/events/state-of-data-mesh-2022`

Stats, I. L. (2019). *Internet live stats.* Retrieved from `https://www.internetlivestats.com/`

Stopford, B. (2018). *Designing event-driven systems*. O'Reilly Media, Inc.

Strauss, A. & Corbin, J. (1998). *Basics of qualitative research techniques*. Thousand oaks, CA: Sage publications.

Stricker, V., Lauenroth, K., Corte, P., Gittler, F., De Panfilis, S. & Pohl, K. (2010a). Creating a reference architecture for service-based systems–a pattern-based approach. In *Towards the future internet* (pp. 149–160). IOS Press.

Stricker, V., Lauenroth, K., Corte, P., Gittler, F., De Panfilis, S. & Pohl, K. (2010b). Creating a reference architecture for service-based systems-a pattern-based approach [Conference Proceedings]. In *Future internet assembly* (p. 149-160).

Sudhakar, G. P. (2012). A model of critical success factors for software projects. *Journal of Enterprise Information Management*.

Suthakar, U. (2017). *A scalable data store and analytic platform for real-time monitoring of data-intensive scientific infrastructure* (Unpublished doctoral dissertation). Brunel University London.

Taibi, D., Lenarduzzi, V. & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. In *Proceedings of the 8th international conference on cloud computing and services science* (pp. 221–232). SCITEPRESS - Science and Technology Publications. doi: 10.5220/0006798302210232

Takeda, H. (2014). *Design science methodology for information systems and software engineering*. Springer.

Talisse, R. B. & Aikin, S. F. (2011). *The pragmatism reader: From peirce through the present*. Princeton University Press.

Tanenbaum, A. S. & Wetherall, D. J. (2011). *Computer networks* (5th ed.). Boston, MA: Pearson.

Team, A. H. (2023). *Artifact hub.* Retrieved from `https://artifacthub.io/`

technology review insights in partnership with Databricks, M. (2021). *Building a high-performance data organization.* Databricks. Retrieved from https://databricks.com/p/whitepaper/mit-technology-review-insights-report

Technology trends outlook (Tech. Rep.). (2023). McKinsey & Company. Retrieved from https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-top-trends-in-tech (Accessed: 2023-06-03)

The Modern Data Company. (2024). *Modern dataos - the data product platform.* https://themoderndatacompany.com/dataos/. (Accessed: 2024-05-13)

ThoughtWorks. (2021). *Thoughtworks technology radar.* Retrieved from https://www.thoughtworks.com/radar

ThoughtWorks. (2023). *Technology radar* (Tech. Rep.). https://www.thoughtworks.com/radar. https://www.thoughtworks.com/radar. (Accessed: [Insert access date here])

Thoughtworks. (2024, April). *Technology radar.* https://www.thoughtworks.com/radar. (Volume 30 | April 2024 Technology Radar An opinionated guide to today's technology landscape)

Tran, P. H., Tran, K. P., Huong, T. T., Heuchenne, C., HienTran, P. & Le, T. M. H. (2018). Real time data-driven approaches for credit card fraud detection [Conference Proceedings]. In *Proceedings of the 2018 international conference on e-business and applications* (p. 6-9). ACM. doi: 10.1145/3194188.3194196

Urquhart, J. (2020). *Streaming architecture: New designs using apache kafka and mapr streams*. O'Reilly Media.

Valdivia, J. A., Lora-González, A., Limón, X., Cortes-Verdin, K. & Ocharán-Hernández, J. O. (2020). Patterns related to microservice architecture: a multivocal literature review. *Programming and Computer Software*, *46*(8), 594–608. doi: 10.1134/S0361768820080253

Vale, G., Correia, F. F., Guerra, E. M., de Oliveira Rosa, T., Fritzsch, J. & Bogner, J. (2022). Designing microservice systems using patterns: An empirical study on quality trade-offs. In *2022 ieee 19th international conference on software architecture (icsa)* (pp. 69–79). IEEE. doi: 10.1109/ICSA53651.2022.00015

van den Driest, F., Sthanunathan, S. & Weed, K. (2016). Building an insights engine [Journal Article]. *Harvard business review*, *94*(9), 15. doi: 10.2501/jar-2016-029

Viana, P. & Sato, L. (2014). A proposal for a reference architecture for long-term archiving, preservation, and retrieval of big data. In *2014 ieee 13th international conference on trust, security and privacy in computing and communications* (pp. 622–629).

Vogel, D. R. & Wetherbe, J. C. (1984). Mis research: a profile of leading journals and universities. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, *16*(1), 3–14.

Vogel, O., Arnold, I., Chughtai, A., Ihler, E., Kehrer, T., Mehlig, U. & Zdun, U. (2009). Software-architektur: Grundlagen–konzepte. *Praxis*, *2*.

Voigt, P. & Von dem Bussche, A. (2017). The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, *10*(3152676), 10–5555.

Volk, M., Bosse, S., Bischoff, D. & Turowski, K. (2019). Decision-support for selecting big data reference architectures. In *International conference on business information systems* (pp. 3–17).

Volk, M., Staegemann, D., Trifonova, I., Bosse, S. & Turowski, K. (2020). Identifying similarities of big data projects–a use case driven approach. *IEEE Access*, *8*, 186599–186619.

Walls, J. G., Widmeyer, G. R. & El Sawy, O. A. (1992). Building an information system design theory for vigilant eis. *Information systems research*, *3*(1), 36–59.

Wamba, S. F., Gunasekaran, A., Akter, S., Ren, S. J.-f., Dubey, R. & Childe, S. J. (2017). Big data analytics and firm performance: Effects of dynamic capabilities [Journal Article]. *Journal of Business Research*, *70*, 356-365. doi: 10.1016/j.jbusres.2016.08.009

Wang, C. J., Ng, C. Y. & Brook, R. H. (2020). Response to covid-19 in taiwan: big data analytics, new technology, and proactive testing. *Jama*, *323*(14), 1341–1342.

Wang, H., Xu, Z., Fujita, H. & Liu, S. (2016). Towards felicitous decision making: An overview on challenges and trends of big data [Journal Article]. *Information Sciences*, *367*, 747-765. doi: 10.1016/j.ins.2016.07.007

Waseem, M., Liang, P., Ahmad, A., Shahin, M., Khan, A. A. & Marquez, G. (2022). Decision models for selecting patterns and strategies in microservices systems and their evaluation by practitioners. In *2022 ieee/acm 44th international conference on software engineering: Software engineering in practice (icse-seip)* (pp. 135–144). IEEE. doi: 10.1109/ICSE-SEIP55303.2022.9793911

Webster, J. & Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, xiii–xxiii.

Weerasinghe, S. & Perera, I. (2022). Taxonomical classification and systematic review on microservices. *International Journal of Engineering Trends and Technology*, *70*(3), 222–233. doi: 10.14445/22315381/IJETT-V70I3P225

Weiser, M., Brown, J. S., Denning, P. & Metcalfe, R. (1998). *Beyond calculation: The next fifty years of computing.* Copernicus.

Weyrich, M. & Ebert, C. (2015). Reference architectures for the internet of things. *IEEE Software*, *33*(1), 112–116.

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*. Springer.

Williams, L. G. & Smith, C. U. (n.d.). Pasasm: a method for the performance assessment of software architectures [Conference Proceedings]. In *Proceedings of the 3rd international workshop on software and performance* (p. 179-189).

Wirth, N. (2008). A brief history of software engineering [Journal Article]. *IEEE Annals of the History of Computing*, *30*(3), 32-39. doi: 10.1109/mahc.2008.33

Wolcott, H. (2008). Ethnography: A way of seeing. 2: nd ed. *Walnut Creek, CA: AltaMira*.

Wolfram Research, Inc. (2021). *Computational Notebook.* `https://www.wolfram`

.com/notebooks/. ([Online; accessed 30 April 2023])

World Wide Web Consortium. (2014). *Data Catalog Vocabulary (DCAT).* https://www.w3.org/TR/vocab-dcat/. ([Online; accessed 30 April 2023])

World Wide Web Consortium. (2017). *Data Quality Vocabulary.* https://www.w3.org/TR/vocab-dqv/. ([Online; accessed 30 April 2023])

Wu, H. (2002). *A reference architecture for adaptive hypermedia applications*. Citeseer.

Yang, C., Liang, P. & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, *111*, 157–184.

Yelp. (2023). *Yelp dataset.* (Available from Yelp: https://www.yelp.com/dataset)

Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5). sage.

Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T. & Chen, E. (2023). A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*.

Yu, C. H. (1994). Abduction? deduction? induction? is there a logic of exploratory data analysis?.

Yu, J.-H. & Zhou, Z.-M. (2019). Components and development in big data system: A survey. *Journal of Electronic Science and Technology*, *17*(1), 51–72.

Zhang, M., Liu, J. & Feng, L. (2019). The application of big data technology in creative travel [Conference Proceedings]. In *2019 international conference on intelligent transportation, big data and smart city (icitbs)* (p. 317-319). IEEE. doi: 10.1109/icitbs.2019.00083

Zhang, Y., Qiu, M., Tsai, C.-W., Hassan, M. M. & Alamri, A. (2015). Health-cps: Healthcare cyber-physical system assisted by cloud and big data [Journal Article]. *IEEE Systems Journal*, *11*(1), 88-95. doi: 10.1109/jsyst.2015.2460747

Zhu, H. (2005). *Software design methodology: From principles to architectural styles*. Elsevier.

Zimmermann, H. (1980). Osi reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*, *28*(4), 425–432.

Appendix A

Expert Opinion Guide

A.1 Introduction

Thanks for your participation. Your opinion is being collected to validate theories in regards to a domain-driven distributed RA that is called Metamycelium. There are no right or wrong answers, and the interest is in your opinion and experiences. This process should take approximately one hour and half depending on the flow of the dialogues.

All your responses will be confidential, and the results of this expert opinion gathering will be presented without mentioning your name. You may decline to answer any question or stop the process at any time and for any reason. Should you wish to not answer any of the questions, you may decline the question. Are there any questions in regards to what I have just explained ?

Note to the reader/researcher: Please not that this guide aims to only encompass the main themes being discussed with the expert and as such does not include the prompts that may have emerged in the process. Some general prompts and close-ended questions are included.

A.2 Establishing Rapport

Before we begin, it would be nice if you could introduce yourself and tell me a bit about your background and your area of interest.

A.3 Candidates background

1. Could you please tell me your job title? (Solution Architect)
2. Could you please tell me how many years of professional experience have you got in software engineering or data engineering? (10 years)

A.4 Familiarity with big data systems

1. Could you please tell how many years of experience have you got related to data engineering or big data? (6 years)
2. Could you please elaborate on your experience/s with big data systems (or any related systems)? expert

A.5 Main Themes

Applicability & Industriallevance

1. Is this artefact directly applicable to current industry needs?
2. How does this architecture align with industry trends and future projections?

A.5.1 Strengths & Distinctive Features

1. What are the core strengths of this architecture?

2. Are there features or components that differentiate this architecture from existing solutions?

A.5.2 Challenges & Potential Barriers

1. What challenges or limitations might organisations face when adopting this architecture?
2. Are there scenarios where this architecture might not be the best fit?

A.6 Closing thoughts

1. Are there any further comments/suggestions/improvements that you have got for our study?

A.7 Expert Opinion Categories

1. I believe the Metamycelium architecture is relevant to the current industry demands.
2. The Metamycelium architecture aligns well with prevailing industry trends.
3. I foresee the Metamycelium architecture being even more relevant in the future industry landscape.
4. The Metamycelium architecture has distinct features that set it apart from existing solutions.
5. The design of Metamycelium promotes efficient data handling and processing.
6. I anticipate challenges in implementing the Metamycelium architecture in certain organisations.

7. The architecture may require significant changes to existing systems to be fully integrated.

8. There might be scenarios where Metamycelium is not the most optimal architecture choice.

9. The design of Metamycelium is robust against diverse data challenges.

10. I believe the Metamycelium architecture can bring substantial benefits to organisations that adopt it.

Appendix B

Reference Architectures Classification

This framework divides RAs into two major classes: standardisation RAs and facilitation RAs.

B.1 Standardisation RAs

1. Type 1: classical, standardisation architectures designed to be implemented in multiple organisations. Examples are:
   (a) WRM (Hollingsworth & Hampshire, 1995)
   (b) OSI RM (Zimmermann, 1980)
   (c) OATH (OATH, 2007)
   (d) COBRA (Pope, 1998)
   (e) Neomycelia (Ataei & Litchfield, 2021b)
   (f) Kappa (Kreps, 2014b)
   (g) Bolster (Nadal, Herrero, Romero, Abelló et al., 2017)
2. Type 2: classical, standardisation architectures designed to be implemented in a single organisation

(a) Fortis Bank Reference Software Architecture (Angelov et al., 2009)

B.2 Facilitation RAs

1. Type 3: classical, facilitation reference architectures for multiple organisations designed by a software organisation in cooperation with user organisations

   (a) Microsoft Application Architecture for .Net (Press, Joyner & Malcolm, 2002)

   (b) IBM PanDOORA

   (c) OATH (OATH, 2007)

   (d) COBRA (Pope, 1998)

2. Type 4: classical, facilitation architectures designed to be implemented in a single organisation

   (a) Achmea Software Reference Architecture (Greefhorst & Gehner, 2006)

   (b) ABN-AMRO Web Application Architecture (Greefhorst, 1999)

3. Type 5: preliminary, facilitation architectures designed to be implemented in multiple organisations

   (a) ERA (Angelov & Grefen, 2008a)

   (b) AHA (Wu, 2002)

   (c) eSRA (Norta, 2007)