Highlights

Terramycelium: A Complex Adaptive Reference Architecture for Big Data Sytems

Author One, Author Two, Author Three

- Research highlight 1
- Research highlight 2

Terramycelium: A Complex Adaptive Reference Architecture for Big Data Sytems

Author One[a], Author Two[b], Author Three[a,b]

[a]*Department One, Address One, City One, 00000, State One, Country One*
[b]*Department Two, Address Two, City Two, 22222, State Two, Country Two*

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: keyword one, keyword two
PACS: 0000, 1111
2000 MSC: 0000, 1111

1. Introduction

The advent of the internet and widespread use of digital devices have sparked a profound shift in connectivity and data creation, leading to an era marked by a rapid growth in data. This period is characterised by the extensive expansion of data, which presents difficulties for traditional data processing systems and necessitates inventive methods in data architecture [**?** **?**]. The vast amount, variety, and rapid generation of data in the current digital environment necessitate innovative solutions, particularly in the field of Big Data (BD).

Data needs have dramatically evolved, transitioning from basic business intelligence (BI) functions, like generating reports for risk management and compliance, to incorporating machine learning across various organisational facets [**?**]. These range from product design with automated assistants to personalised customer service and optimised operations. Also, as machine

learning becomes more popular, application development needs to change from rule-based, deterministic models to more flexible, probabilistic models that can handle a wider range of outcomes and need to be improved all the time with access to the newest data. This evolution underscores the need to reevaluate and simplify our data management strategies to address the growing and diverse expectations placed on data.

Currently, the success rate of BD projects is low. Recent surveys have identified the fact that current approaches to big data do not seem to be effectively addressing these expectations. According to a survey conducted by [**?**], only 13% of organisations are highly successful in their data strategy. Additionally, a report by NewVantage Partners reveals that only 24% of organisations have successfully converted to being data-driven, and a measly 30% have a well-established big data strategy. These observations, additionally corroborated by research conducted by McKinsey & Company (analytics2016age) and Gartner (Nash), emphasise the difficulties of successfully using big data in the industry. These difficulties include the lack of a clear understanding of how to extract value from data, the challenge of integrating data from multiple sources, data architecture, and the need for skilled data analysts and scientists.

Without a well-established big data strategy, companies may struggle to navigate these challenges and fully leverage the potential of their data. One effective artefact to overcome some of these challenges is Reference Architectures (RAs) [**?**]. RAs extract the essence of the practice as a series of patterns and architectural constructs and manifest it through high-level semantics. This allows stakeholders to refrain from reinventing the wheel and instead focus on utilising existing knowledge and best practices to harness the full potential of their data. While there are various BD RAs available to help practitioners design their BD systems, these RAs are overly cetnralised, lack attention to cross-cutting concerns such as privacy, security, and metadata, and may not effectively handle the proliferation of data sources and consumers.

To this end, this study presents TerrMycelium, a distributed RA designed specifically for BD systems with a focus on domain-driven design. TerrMycelium seeks to surpass the constraints of current RAs by utilising domain-driven and distributed approaches derived from contemporary software engineering. This method aims to improve the ability of BD systems to scale, be maintained, and evolve, surpassing the constraints of traditional monolithic data architectures.

The paper is structured as follows: Section 2 provides an overview of the foundational concepts and technologies pertinent to BD reference architecture, aiming to forge a conceptual framework that is required for this paper. An overview of the existing research on the topic is presented in Section 3. The significance of reference architectures in the context of big data is explored in Section 5. Section 6 details the software and system requirements necessary for implementing the proposed architecture. Section 7 delves into the theoretical foundation underpinning the challenges in contemporary big data systems. The design and development of the TerrMycelium artifact are described in Section 8. Section 9 examines the evaluation findings, their implications, limitations, and relevance to existing and future research. Finally, Section 10 summarizes the main contributions of the study, its practical implications, and suggests directions for future research.

2. Background

This section offers fundamental definitions necessary for understanding the intricacies of the research. This chapter seeks to establish the conceptual framework required for comprehending the terminology utilised in the thesis.

2.1. Definition of Big Data

Various academic definitions have been reviewed to define BD in this research. According to Kaisler et al. (2013), Big Data (BD) refers to data that exceeds the capacity of current technologies to store, manage, and process effectively. The authors Srivastava (2018) define big data as the use of extensive data sets for managing the gathering or reporting of data that aids multiple recipients in decision-making. The phrase "Big Data" (BD) is defined by **?**] as referring to extensive data sets characterised by their huge size, diverse and intricate structure, and the challenges associated with storing, analysing, and visualising them for subsequent processes or outcomes. In this research, Big Data (BD) is defined as the process of identifying patterns from large datasets to achieve progress, management, and predictive analysis in specialised fields.

2.2. Significance of Big Data

The importance and worth obtained from Big Data are clear. Extensive discussions on the topic are prevalent in publications, statistics, research, and conferences [**?**]. Prominent corporations such as Google, Facebook, Netflix,

and Amazon have significantly boosted this trend by major investments in BD projects (Rada, 2017).

The Netflix Prize recommender system is a striking example of the real benefits of BD. The system utilised a wide range of data sources such as user queries, ratings, search phrases, and demographic factors to enhance its performance (Amatriain, 2013). Netflix saw a significant rise in TV series viewership and some series even quadrupled their audience by using Big Data-powered recommendation algorithms.

The Taiwanese government effectively integrated its national health insurance database with customs and immigration statistics as part of a Big Data plan in the healthcare sector (Wang, 2020). The real-time notifications during clinical visits, based on clinical symptoms and travel history, helped identify possible COVID-19 cases proactively. Strategic data-driven initiatives greatly enhanced Taiwan's ability to manage the outbreak. Shell utilises Big Data in the field of energy exploration to enhance the decision-making process and minimise exploration expenses (Marr, 2016).

By analysing and contrasting data from different drilling sites worldwide, judgements are directed towards places that resemble those with verified abundant resources. Before BD's integration, identifying energy resources was difficult. Conventional exploration techniques, based on interpreting seismic vibrations passing through the earth's crust, were both inaccurate and costly and time-consuming. Rolls Royce leverages Bangladesh's potential by gathering detailed performance data from sensors installed on its aircraft products.

Wirelessly transmitted data offers valuable insights into crucial operating stages, including take-off and maintenance. By utilising this abundance of information, Rolls Royce can more precisely identify deterioration, improve diagnostic and prognosis accuracy, and efficiently decrease false alarms. subsectionReference Architectures Reference Architectures (RAs) are crucial components in modern system development, providing guidance for building, maintaining, and evolving complex systems citeCloutier2010.

They provide a precise representation of the fundamental elements of a system and the interactions required to achieve broad goals. This clarity encourages the development of digestible modules, each focusing on certain parts of complicated issues, and offers a sophisticated platform for stakeholders to participate, contribute, and work together. The importance of RAs in IT is highlighted by the success of widely used technologies such as OAuth (OATH) and ANSI-SPARC architecture (ANSI), which have their roots in

well-organized RAs.

RAs define the characteristics of a system and influence its development. RAs stand apart by emphasising abstract features and higher levels of abstraction, which are present in any system's architecture. They strive to encapsulate the core of practice and incorporate proven patterns into unified frameworks, covering elements, attributes, and interconnections. RAs play a crucial role in BD by managing communication, controlling complexity, handling knowledge, reducing risks, promoting architectural visions, establishing common ground, improving understanding of BD systems, and enabling additional analysis.

2.3. Microservices and Decentralised, Distributed Architectures

Microservices architecture is a modern approach in software engineering that organises applications as a set of independent services that are loosely connected [**?**]. This method, derived from the wider idea of Service Oriented Architectures (SOA), concentrates on creating small, self-sufficient modules that work together to create a complete application.

According to **?** , microservices improve scalability, enable continuous deployment, and promote a more agile development environment. They allow teams to design, deploy, and scale components of a system separately, enhancing system resilience and enabling quick adaptability to changing requirements. Decentralised and distributed architectures are essential in today's computer environment, featuring systems that are spread out among several nodes, typically in various geographic regions.

The architectural style, as emphasised by Richards (2015), addresses the drawbacks of conventional monolithic architectures by providing improved scalability, fault tolerance, and flexibility. Distributed systems involve the distribution of data and processing among several nodes that collaborate to execute tasks, as explained by Coulouris (2005). Decentralisation in this context is the absence of a central controlling node, choosing a more democratic and robust network structure instead. The integration of microservices in these systems signifies an advancement in software engineering.

The shift is towards systems that are dispersed, modular, and flexible. This architectural style is well-suited for current requirements for systems that can scale, withstand challenges, and take use of the decentralised structure of modern computing environments. Implementing microservices in decentralised, distributed architectures marks a new phase in software development, emphasising the importance of flexibility, scalability, and resilience.

3. Related Work

This section reviews significant works in BD RAs, detailing their emphasis, research methodologies, and inherent limitations, to endorse the distinctive approach of this study's domain-driven distributed RA, *Terramycelium*. The Lambda design by Kiran (2015) and the Kappa architecture by Kreps (2014) are important industrial advancements in real-time analytics for big data, setting foundational principles for data processing. The architectures have faced criticism for their insufficient data management techniques, particularly concerning data quality, security, and metadata (AtaeiACIS).

This gap is evident in domain-specific Regulatory Authorities (RAs) as those proposed by Quintero (2019) for healthcare. These RAs, while concentrating on certain domain requirements, sometimes overlook broader concerns such as privacy and interoperability. Academic research, like Viana (2014) and Paakkonen (2015), has concentrated on enhancing the conceptual comprehension of Big Data systems through proposed Reference Architectures to provide broader views on data analytics ecosystems. Yet, these recommendations frequently overlook the dynamic and pervasive features of modern data environments, particularly concerning scalability and adaptability. Ataei et al. (2022) highlighted the limitations of current BD RAs, emphasising a common trend: a substantial reliance on monolithic data pipeline architectures.

Existing RAs' reliance is apparent due to their inefficiency in managing data quality, security, privacy, and metadata. Additionally, the inflexible framework of these systems often results in difficulties with scalability and adaptation, making it challenging for them to accommodate changes in data and technology settings. Within this framework, *Terramycelium* is presented as a novel regulating agent for biological defence mechanisms. *Terramycelium* addresses the major limitations of current RAs by implementing a domain-driven, distributed method. *Terramycelium* advocates for decentralised data stewardship and a modular architecture, as opposed to traditional systems that usually have data management in rigid, monolithic structures.

This approach enhances scalability and adaptability by including overarching issues such as security, privacy, and data quality into the system's architecture. The term "Terramycelium" is a significant departure from traditional BD RAs. By focusing on domain-driven design and distributed processing, it offers a scalable and adaptable framework that aligns well with current data management needs and contemporary software architecture concepts. Terramycelium not only addresses the limitations of present RAs but

also emerges as a trailblazer in the progress of BD system design, paving the way for future research and development in this vital field.

4. Research Methodology

Various methods exist for the structured creation of RAs. **?** present a sophisticated strategy for creating RAs by gathering current architectural trends and innovations. citeauthorbayer1999pulse present a technique named PuLSE DSSA for generating RAs in product line development. The concept of a pattern-based Runtime Adaptation for service-based systems is introduced by Stricker et al. (2010), who emphasise the usage of patterns as primary entities. The authors Nakagawa et al. present a four-step method for creating and advancing RAs.

Guided by ISO/IEC 26550 **?** and **?** provide a four-phase method for creating practical risk assessment in the domain engineering and software product line environment. The authors of GALSTER2011Empirically suggest a 6-step process based on two main concepts: empirical foundation and empirical validity. Considering all these factors, an empirically-based research technique is the most suitable approach for this study. This process for RA development is preferred over others and aligns with the objectives of this study.

However, additional approaches must be incorporated into the methodology to achieve the appropriate level of rigour and relevance. Specific instructions for gathering empirical data in step 3 are absent. We were unsure how to proceed with data collection, synthesis, and modelling. **?** presented research guidelines and introduced the RAModel concept. A more systematic and robust evaluation technique is needed as the methodology lacks details on how to assess the RA. To solve this issue, a case-mechanism experiment and expert opinion were used to assess the artefact.

4.1. Step 1: Determination of RA Type

The first step in creating the RA was choosing its kind using Angelov et al.'s classification framework, which divides RAs into standardisation RAs and facilitation RAs. This decision is fundamental as it directs the following stages of information gathering and risk assessment development.

The classification framework, which considers context, aims, and design dimensions, was crucial in determining the most suitable RA type for the study's objectives. The method takes a systematic approach by employing

key interrogatives such as 'When', 'Where', 'Who' for context, 'Why' for goals, and 'How' and 'What' for design to efficiently categorise RAs.

The study improved Angelov's classification by incorporating insights from a recent Systematic Literature Review (SLR) on Big Data Recommender Systems (BD RAs) cited as AtaeiACIS. The updated classification includes current instances, which can be found in Appendix A under the heading RA-classification. The selected Research Assistant for this project specialises in domain-driven distributed Big Data Research Analysis. The goal is to facilitate Big Data system development and enhance an efficient, adaptable data architecture. The standardisation RA is intended to be adaptable in many organisational settings.

4.2. Step 2: Design Strategy Selection

The design approach for the RA was influenced by the frameworks proposed by Angelov et al. (2008) and Galster et al. (2011), which describe two main methodologies: practice-driven (creating RAs from the beginning) and research-driven (building RAs based on pre-existing ones). Practice-driven RAs are uncommon and usually found in emerging areas, while research-driven RAs, which combine current designs, models, and best practices, are more common in established sectors.

This study chooses a research-driven strategy based on these ideas. The RA was created by utilising existing RAs, concrete structures, and documented best practices. This method allows for the development of a detailed design theory that combines and expands on the existing knowledge in the topic.

4.3. Step 3: Empirical Data Collection

Due to limitations in the study technique, we have enhanced this phase by increasing the systematicity and transparency of data gathering and synthesis using academic approaches like SLR. For this purpose we have adopted the findings from a SLR on current state of BD RAs presented by **?** .

4.4. Step4: Construction of the RA

The construction phase of the RA was guided by the findings and elements identified in the earlier stages of the research. Based on the ISO/IEC/IEEE 42010 standard, the construction process involved selectively integrating its components.

This phase prominently featured the implementation of the Archimate modelling language, which is part of the ISO/IEC/IEEE 42010 standard. Archimate's service-oriented strategy efficiently connected the application, business, and technological layers of the RA. The incorporation of themes, theories, and patterns from the empirical data gathering phase, as detailed in sections **??** and **??**, guaranteed that the research analysis addressed the specific demands stated in the study.

Using Archimate's various viewpoints offered a comprehensive perspective on the RA, encompassing technical, business, and consumer context views. This method, in accordance with the ideas presented by Cloutier et al. and Stricker et al., facilitated a thorough comprehension of the RA, guaranteeing its congruence with the study's goals and context.

4.4.1. Step 5: Activating RA with variability

Integrating variability into RAs is crucial to ensure its application within organizational-specific legislation and regional policies. Variability management is essential in Business Process Management (BPM) and Software Product Line Engineering (SPLE) to customise business processes and software artefacts according to specific contextual requirements.

Accurate identification and clear communication of variability are essential to encourage stakeholder discussion, maintain decision traceability, and aid the decision-making process [**?**]. Data collected in previous analytical steps informs variability decision points. **?** outline three approaches to include variability into RAs: annotating the RA, constructing variability views, and forming variability models.

Current literature lacks in-depth information on the criteria for choosing a method to facilitate variability. This study utilises Archimate annotations to incorporate variability into the RA, following the method proposed by Rurua (2019). The technique involves two stages: first, developing a customised layer to encompass fundamental variability principles, and second, annotating the RA. The aim is not to list every possible variability point, but to highlight the main architectural variabilities associated to the system for architects to consider in order to improve the design and make it easier to implement the RA.

4.5. Step 6: Evaluation of the RA

Evaluating the RA is essential to ensure it achieves its developmental goals, especially in terms of efficacy and usefulness (Galster, 2011). Assessing

a Research Article (RA) presents distinct difficulties because of its elevated level of abstraction, varied stakeholder groups, and emphasis on architectural features.

Common evaluation techniques for concrete architecture, like SAAM, ALMA, PASA, and ATAM, are not suitable for abstract architectural representations (RAs) because they heavily depend on stakeholder participation and scenario-based assessment. This requires a tailored method for evaluating RA.

This work utilises a revised evaluation method based on techniques tailored for RAs by Angelov et al. (2008) and the expanded SAAM methodology by Graaf et al. (2005). The approach includes developing a prototype of the RA inside for a case-mechanism experiment and expert opinion as presented by **?** .

5. Why Reference Architectures

Viewing the system as a RA aids in comprehending its main elements, behaviour, structure, and development, which subsequently impact quality characteristics like maintainability, scalability, and performance (Cloutier). RAs can serve as a valuable standardisation tool and a means of communication that leads to specific architectures for BD systems.

They also offer stakeholders a common set of elements and symbols to facilitate discussions and advance BD projects. Utilising RAs for system conceptualization and as a standardisation object is a common technique among those dealing with complex systems. Software Product Line (SPL) development uses RAs as generic artefacts that are customised and configured for a specific system domain. IBM and other top IT companies have continually supported the use of RAs as exemplary techniques for solving complex system design difficulties in software engineering.

RAs often act as instruments in establishing uniformity in new areas within international standards. The BS ISO/IEC 18384-1 RA for service-oriented architectures showcases the effectiveness of RAs in establishing standardised frameworks in particular domains. This study utilises a 6-phase technique for creating the RA as outlined by Galster (2011). The six phases are as follows: 1) Determination of the kind of the RA 2) Develop a design strategy 3) Gather empirical data 4) Construct the Risk Assessment 5) Activate RA with variability 6) Assessment of the RA.

The term "empirically grounded" means that the risk assessment (RA) should be based on known principles and then assessed for applicability and validity. These principles are not exclusive to the methods of Galster (2011); other researchers like Cloutier and Derras have also advocated for them.

6. Software and System Requirements

According to **?** [**?**], 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.

? 's 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 study is made up of the following phases:

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

6.1. Determining the type of the requirements:

Defining and classifying software and system requirements is a common subject of debate. **?** [**?**] classify 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. Nevertheless, in this study, a more general framework provided by **?** 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.

6.2. Determining the relevant requirements:

In an extensive effort, the NIST Big Data 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 Defense [**?**]. The result of this study is the formation of general requirements under seven categories. In addition, **?** categorize nine use cases of BD projects sourced from published literature using a hierarchical clustering algorithm.

? [**?**] focus on security and privacy requirements for BD systems, **?** present modern components of BD systems, using goal-oriented approaches, **?** created a generic model for BD requirements, and **?** investigate general requirements to support BD software development.

By analyzing the result of the first SLRs, the studies discussed above and by evaluating the design and requirement engineering required for BD RAs, a set of high-level requirements based on BD characteristics is established.

6.3. Identifying the right approach for categorisation of the requirements:

After clarifying the type of requirements and the relevant requirements, current BD RAs and their requirements have been assessed to increase understanding of the available BD requirement categorisation methods. A common theme among these studies connected the dot towards a common approach to classifying requirements. This approach is categorised through BD characteristics such as velocity, veracity, volume, variety, value, security and privacy [**?** **?** **?** **?**].

6.4. Determining the appropriate strategy for presenting the requirements:

A systematic strategy for presenting software and system requirements that includes informal model verification techniques is recognised due to its established use in both business and academics (Kassab, 2014). The method for illustrating functional requirements adheres to the principles stated in the ISO/IEC/IEEE standard 29148. The requirements representation is structured based on system modes, detailing the primary components of the system followed by the requirements. This technique is based on the requirement specification published for the NASA Wide-field InfraRed Explorer (WIRE) system and the Software Engineering Body of Knowledge (SEBoK). The requirements are outlined in Table 1.

Table 1: Terramycelium 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, standardization, and normalization 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
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.

7. Theory

An inflection point in mathematics is a critical point where the curvature of a curve changes direction, indicating a shift from one behaviour to another. This significant moment is characterised by the breaking down of the previous structure and the appearance of a new shape. Currently, there are concrete indicators and factors that suggest a significant change is imminent. The 2023 New Vantage Partners research indicates that investments in data are increasing and staying robust, despite possible economic challenges. However, the survey indicates that only 23.9% of organisations describe themselves as data-driven. Prior to delving into these design theories, it is crucial to establish a precise communication mechanism for architectural structures. This study utilises the architecture definition defined in ISO/IEC 42010, which defines architecture as the underlying concepts or properties of a system within its context, manifested in its elements, relationships, and design and evolution principles.

7.1. The monolith

The techniques and methods of data engineering have experienced rapid growth, similar to the Cambrian explosion, yet the fundamental assumptions guiding data designs have not been significantly questioned.

As per Richards (2020), architectures can be classified into two main categories: monolithic, deployed as a single entity, and distributed, consisting of individually deployed sub-components. Currently, the majority of data architectures fall under the first type.

Beginning with a monolith may be a straightforward and effective method for constructing a data-intensive system, but it becomes inadequate as the system grows. Although this notion is being questioned in the software engineering field, data engineering continues to be influenced by monolithic designs. Enabler technologies like data warehouses and data lakes support these concepts. Furthermore, numerous groups and publications embrace the concept of a "single source of truth."

7.2. The data chasm

Analytical data and operational data are two separate categories of data utilised in corporate operations. Operational data is utilised for daily corporate operations, whereas analytical data is employed for strategic decision-making by recognising patterns and trends in previous data.

The problems of current Big Data systems stem from the basic assumption of segregating operational and analytical data. Operational and analytical data have distinct characteristics and processing methods. Moving operational data distant from its source can harm its integrity, lead to organisational silos, and result in data quality problems.

These two distinct data planes are often managed inside separate organisational hierarchies. Data scientists, business intelligence analysts, machine learning engineers, data stewards, and data engineers typically work under the guidance of the Chief Data and Analytics Officer (CDAO) to generate business value from data. Software developers, product owners, and quality assurance engineers typically collaborate with the Chief Technology Officer (CTO).

As a consequence, there are now two separate technology systems in place, requiring significant resources to connect them. Two distinct topologies and delicate integration designs have emerged due to this gap, facilitated by ETLs (Figure 1). Data extraction from operational databases is often accomplished through a batch ETL process. These ETL processes typically lack a clearly defined agreement with the operational database and solely consume its data. This emphasises the vulnerability of this system, as modifications made to operational databases can impact analytical applications downstream. As time passes, the complexity of ETL jobs grows, making them harder to maintain and leading to a decline in data quality.

Many technologies developed over the years are based on this notion. Although these technologies excel in managing the amount, speed, and diversity

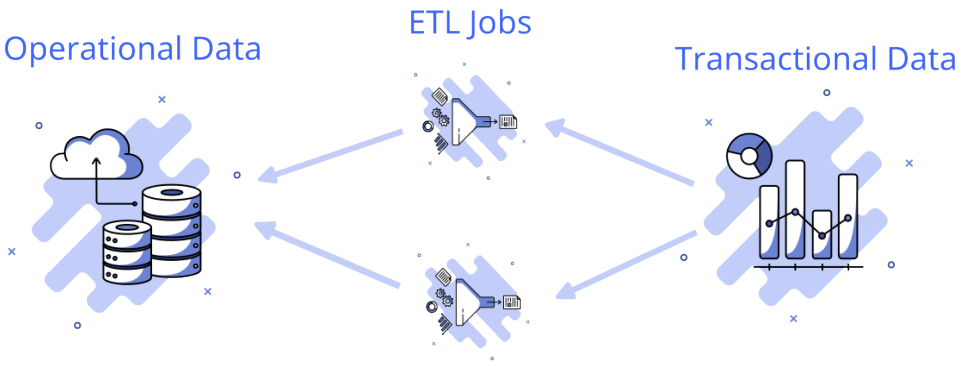

Figure 1: The great divide of data

of data, current data difficulties revolve around the increase in data sources, data quality, and data architecture.

Information is typically gathered and unified through multiple platforms. Some of this data may exceed the boundaries of the organisation. Therefore, considering the points mentioned earlier and the difficulties outlined in Section **??**, it is suggested that modern data architectures should move away from centralising data in a single large analytical database and instead focus on connecting analytical data from various sources.

The artefact created for this study utilises earlier solutions to improve and overcome their limitations. This artefact tries to move away from highly centralised and rigid data architectures that serve as a bottleneck for coordination. This artefact aims to connect the point of data generation with its utilisation, streamlining the process. This artefact is designed to enhance agility in response to growth and successfully adapt to organisational changes. The initial architectural style of Metamycelium is as described:

7.2.1. Localized autonomy through domain-driven decentralisation

Modern enterprises are facing significant complexity. A typical business consists of diverse domains with distinct structures. These domains exhibit varying rates of change and are typically segregated from one another. The business's overall synergy is determined by the relationship and evolution of various domains.

The core of these synergies is characterised by volatility, frequent mar-

ket changes, and a growing number of rules. How do modern organisations handle the effects of these developments on their data? Should they consistently update ETL tasks, develop new ETL pipelines, and merge data into operational repositories? How can firms generate reliable and high-quality data efficiently? It comes down to accepting the change in the current data environment.

To address this complexity, one approach is to synchronise technology with business objectives. Businesses divide their challenge into smaller components within each domain, integrating technology and data into these areas. This method is well recognised in microservices architectures (Ataei, 2023).

7.2.2. Domain-driven design

Central to Metamycelium is the dispersal and decentralisation of services into distinct areas with well-defined boundaries. One of the most difficult aspects of designing a distributed system is determining the architectural components on which the system should be divided. This matter has been frequently deliberated among proponents of microservices architecture, as evidenced in the citation from DataMesh. Metamycelium, influenced by domain-driven desigDataMesh places data in close proximity to the product domain it pertains to. This suggests that data is within the product domain and is a component of it [**?**].

Most businesses nowadays are structured around their goods, which is the fundamental driving force behind this. These items represent the business's talents that are divided into different domains. These domains often establish their defined scope, progress at varying speeds, and are managed by interdisciplinary teams [**?**]. Integrating data into these specific areas can provide a synergistic effect that enhances the handling of ongoing changes.

Communication can occur on a micro level between application developers and data engineers over user data collection in layered data structures, or on a macro level when application developers consider rebuilding their Graphql schema in a way that impacts downstream analytical services. Thus, the study integrates the concept of Domain-Driven Design (DDD) to enhance communication and improve the acceptance, precision, and significance of Metamycelium.

Domain-Driven Design (DDD) is a software development methodology that emphasises comprehending and representing the problem domain of a software application. The objective of Domain-Driven Design (DDD) is to

develop software that mirrors the language, concepts, and behaviours of the problem domain, prioritising these aspects over technical factors.

DDD can assist Metamycelium by offering a structured method for modelling and handling data that is well-matched with the application's issue domain. Domain-Driven Design (DDD) can assist data architects in obtaining a more profound comprehension of the necessary data and its structure by concentrating on the language, concepts, and behaviours of the problem domain (eridaputra 2014 modelling). DataMesh Effective communication is crucial in software development projects, as it facilitates the sharing of vital knowledge [**?**]. Data developers and business stakeholders frequently do not connect directly. Domain knowledge is conveyed through intermediaries like business analysts or project managers into a set of activities to be completed (Khrononov, 2021). This indicates the need for at least two translations from two distinct ontologies.

Each translation results in the loss of crucial subject expertise, posing a danger to the overall data quality. During a data engineering process, the requirements can become twisted, and the data engineer may lack awareness of the specific business domain or problem being addressed.

Data engineering challenges are typically complex and wide-ranging, rather than simple mathematics problems or puzzles. An organisation may choose to enhance workflows and processes by consistently making decisions based on data. However, a rigid and inflexible centralised data architecture can increase the likelihood of project failure.

7.2.3. Complex adaptive systems

Metamycelium architectures have characteristics similar to those of complex adaptive systems (Holland, 1992). The artefact created for this thesis is inspired by the concept that influential groups might form based on basic rules that control individual agents in a certain area. Reynolds (1987) examined a coordinated group of starling birds during the autumn season. The study revealed that each starling bird adheres to three basic rules: alignment, separation, and cohesion. The rules can be mathematically articulated as follows:

Alignment:

$$v_i(t+1) = v_i(t) + \frac{1}{k}\sum_{j=1}^{N}(v_j(t) - v_i(t))$$

where $v_i(t)$ is the velocity vector of bird i at time t, k is a normalization

factor, and N is the number of neighbouring birds.

Cohesion:

$$v_i(t+1) = v_i(t) + \frac{1}{k}(c_i(t) - p_i(t))$$

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 normalization factor.

Separation:

$$v_i(t+1) = v_i(t) + \sum_{j=1}^{N} \frac{(p_i(t) - p_j(t))}{d_{ij}^2}$$

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 require a centralised orchestrator to form this intricate adaptive system. Metamycelium aims to encourage a domain-driven allocation of data ownership. This architecture is designed to offer not just operational data but also analytical data through a standard interface. The practice management software for veterinarians includes operational APIs for updating animal attributes and analytical interfaces for retrieving animal data within a specific time frame. Each domain possesses its own data.

The domain can retrieve data from other domains using a discovery mechanism, process the data, and augment its own data. Aggregate domains are created to collect data from different domains and make it available for a certain purpose.

This aims to eliminate vertical dependency and enable teams to have local autonomy while being equipped with the appropriate amount of discovery and APIs. This architecture emphasises the concept of coequal nodes working together to accomplish the system's overarching objective, rather than relying on a centralised database managed by individuals lacking domain expertise. This idea is influenced by Domain-Driven Design (DDD) as described by Evans (2004), data mesh by Dehghani (2020), and microservices architecture by Newman (2015).

7.2.4. Event driven services

Metamycelium's decentralised and distributed structure poses difficulties in service communication as the network expands. At first, basic point-to-point communication via REST API calls is adequate, but it becomes inefficient as the system grows. Synchronous interactions can result in a 'dis-

tribution tax,' where one service's blocked state, generally caused by costly procedures, leads to delays in dependent services [**?**].

Distributed systems' extensive network requirements can lead to complications such as tail latency, context switching, and gridlocks. This strong interconnection contradicts the distributed system's objectives of independence and robustness.

Metamycelium uses asynchronous event-driven communication to address these concerns. This approach allows services to publish and respond to events, so separating their interactions. Within the "publish and forget" paradigm, services broadcast events to designated topics and proceed without waiting for immediate reactions. Restaurant personnel respond to environmental signals rather than direct directives to enhance operational efficiency.

Event-driven architectures usually provide eventual consistency, which may not be ideal for some real-time stream processing situations that need immediate consistency. However, it is generally safe to assume that most data engineering tasks can effectively function within an event-driven framework.

As per Richards (2020), event-driven architectures consist of two primary topologies: 1) broker topology and 2) mediator topology. The concept of streaming platforms is further explained in the writings of Stopford (2018). Metamycelium, as a BD architecture designed for processing analytical data, eliminates several issues associated with accomplishing ACID transactions. Metamycelium utilises the architectural principles of distributed asynchronous event-driven systems with a mixed topology. Metamycelium is incorporating aspects from the broker topology, mediator topology, and several notions from streaming platforms. Metamycelium's CAP theorem prioritises partition tolerance and availability as key assurances.

The hybrid topology consists of five main architectural components: the event, the event consumer, the event producer, the event backbone, and the eventing interface. Events are triggered by the event producer and sent to the relevant topic. The event is processed by the event broker and stored in a queue-like indexed data structure for later retrieval. Interested event consumers will listen to the topic of interest using the eventing interface. The event backbone is an internally distributed system composed of numerous event brokers.

Event brokers are services created and provided to enable event communication via Metamycelia. The brokers are synchronised with a decentralised service coordinator. Brokers also enable the replication of subjects. Additionally, the event backbone includes a specialised event archive to ensure

fault tolerance and recoverability. The purpose of this archive is to retain all events processed by the brokers for potential restoration in case of a failure.

The components collaborate to establish a distributed, fault-tolerant, and scalable data system capable of managing both batch and stream processing, as seen in Figure 2.

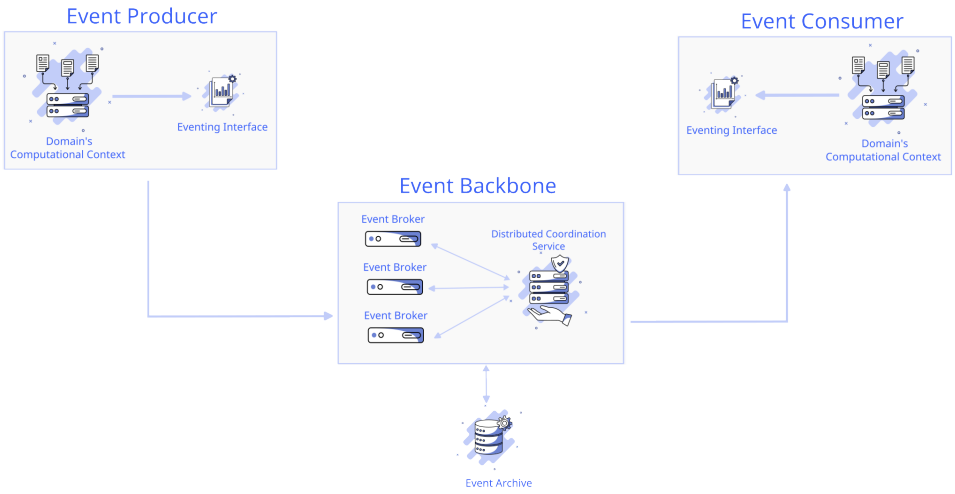

Figure 2: Metamycelium Event-driven Architecture

7.3. Architectural characteristics

The Metamycelium architecture is distinguished by its emphasis on maintainability, scalability, fault tolerance, elasticity, and deployability. It is in line with contemporary engineering methodologies like automated deployment and continuous integration. The architecture focuses on utilising microservices, which are autonomous components that can be deployed and maintained separately.

This architecture excels in maintainability. Event-driven microservices architecture enables modular development and independent scaling, facilitating the maintenance and updating of individual components without impacting the entire system. The design also enables automated deployment processes, which help streamline upgrades and minimise manual interaction.

Scalability and elasticity are significant aspects as well. The architecture allows for horizontal scalability, enabling the addition or removal of services

according to demand. This adaptability guarantees that the system can efficiently manage different workloads.

The architecture also demonstrates strong fault tolerance. Interservice communication can affect fault tolerance, however a redundant and scalable design of Metamycelium, combined with service discovery techniques, helps alleviate this problem. Microservices' standalone and single-purpose design typically results in a high level of fault tolerance.

Metamycelia's minimal deployment components and decoupled design highlight deployability. The design facilitates evolutionary transformation to correspond with contemporary business practices that demand agility and adaptability. Modular and autonomous units provide quicker updates and iterations, ensuring alignment with the ever-changing business needs.

The architecture might be rated lower in terms of cost and simplicity. The decentralised structure of Metamycelium and the possibility of higher communication overhead can create challenges in cost management and optimisation. Implementing strategies like intelligent data caching and replication can help overcome performance issues related to network calls, but managing costs is still a continuous concern.

The Metamycelium design emphasises the advantages of microservices, focusing on maintainability, scalability, fault tolerance, elasticity, and deployability. The text recognises the difficulties that come with distributed architectures and provides methods to reduce their impact. Architects need to comprehend the principles of architecture in order to skillfully traverse and capitalise on its advantages. An overview of architectural characteristics is portrayed in Table 2

8. Artifact

After exploring several kernel and design theories, a solid theoretical foundation is established for creating and developing the artefact. Metamycelium is generated using Archimate and mostly showcases the RA within the technological layer. The architect has the responsibility to determine the flow and application that should be present in each node when showcasing these services in the technology layer. To ensure thoroughness, a basic business process is assumed as every software is created to fulfil a business requirement. Metamycelium should possess the flexibility needed to accommodate alternative business models, notwithstanding potential variations in the business layer across different settings.

Table 2: Metamycelium Architecture Characteristics

Characteristic	**Score**
Maintainability	★★★
Scalability	★★★★
Fault Tolerance	★★★
Elasticity	★★★★
Deployability	★★★★
Cost	★★
Simplicity	★
Performance	★★★
Support for Modern Engineering Practices	★★★★

The BD RA does not depict the architecture of any particular BD system. It functions as a versatile tool for describing, discussing, and constructing system-specific designs based on standardised principles. Metamycelium enables in-depth and insightful talks about the needs, frameworks, and functions of BD systems. The system stays vendor-neutral, providing flexibility in choosing products or services, and avoids imposing strict solutions that restrict innovation.

To ensure thoroughness, we have assumed a basic BD business process, as all software is created to fulfil a specific business requirement. Metamycelium consists of 15 primary components and 5 variable components, as seen in Figure 3. The lowercase "a" in the top left corner of the diagram represents the auxiliary view, while the letter "m" represents the master view. When the same entity is utilised in various models, the auxiliary view is employed. This indicates that the entity already exists 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 4.

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. Its responsibilities 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 bal-

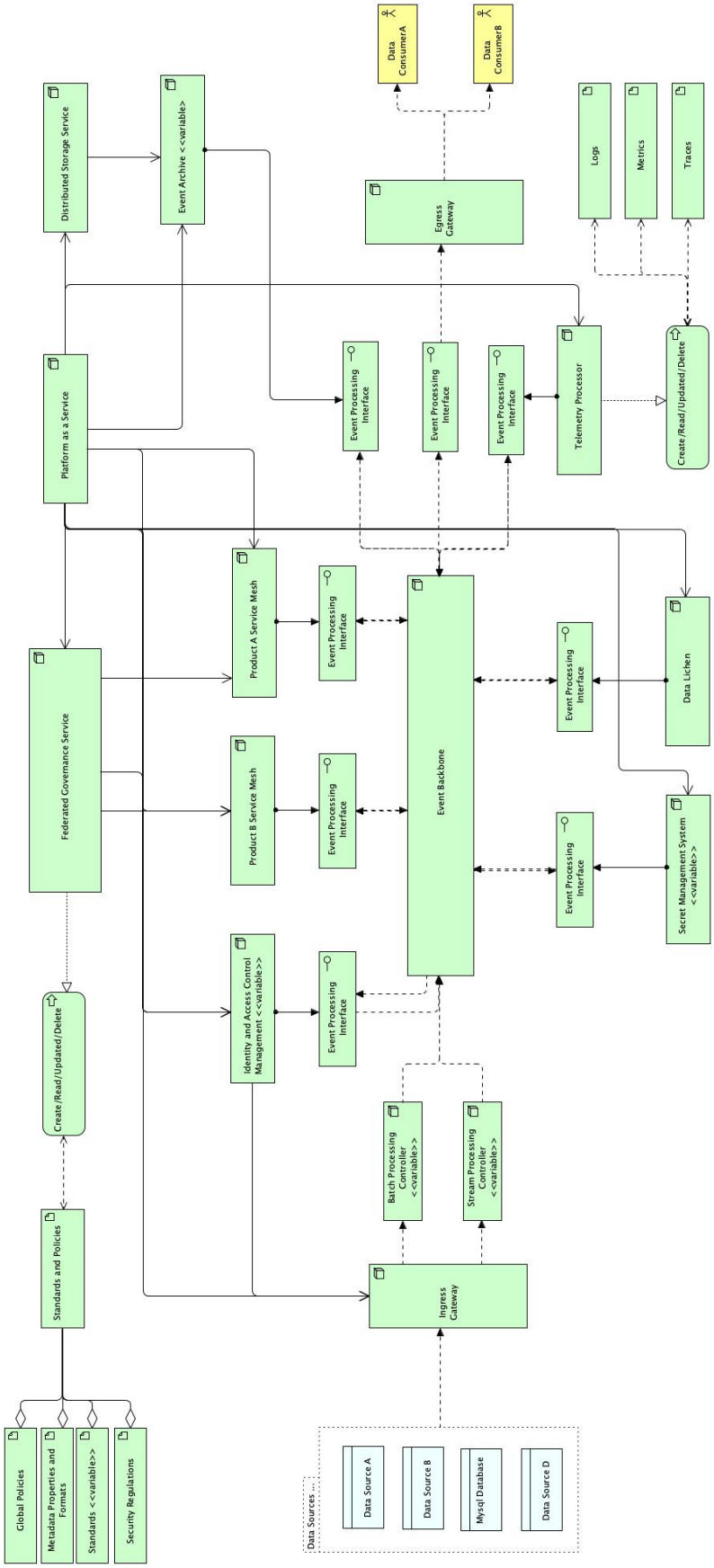

Figure 3: Metamycelium

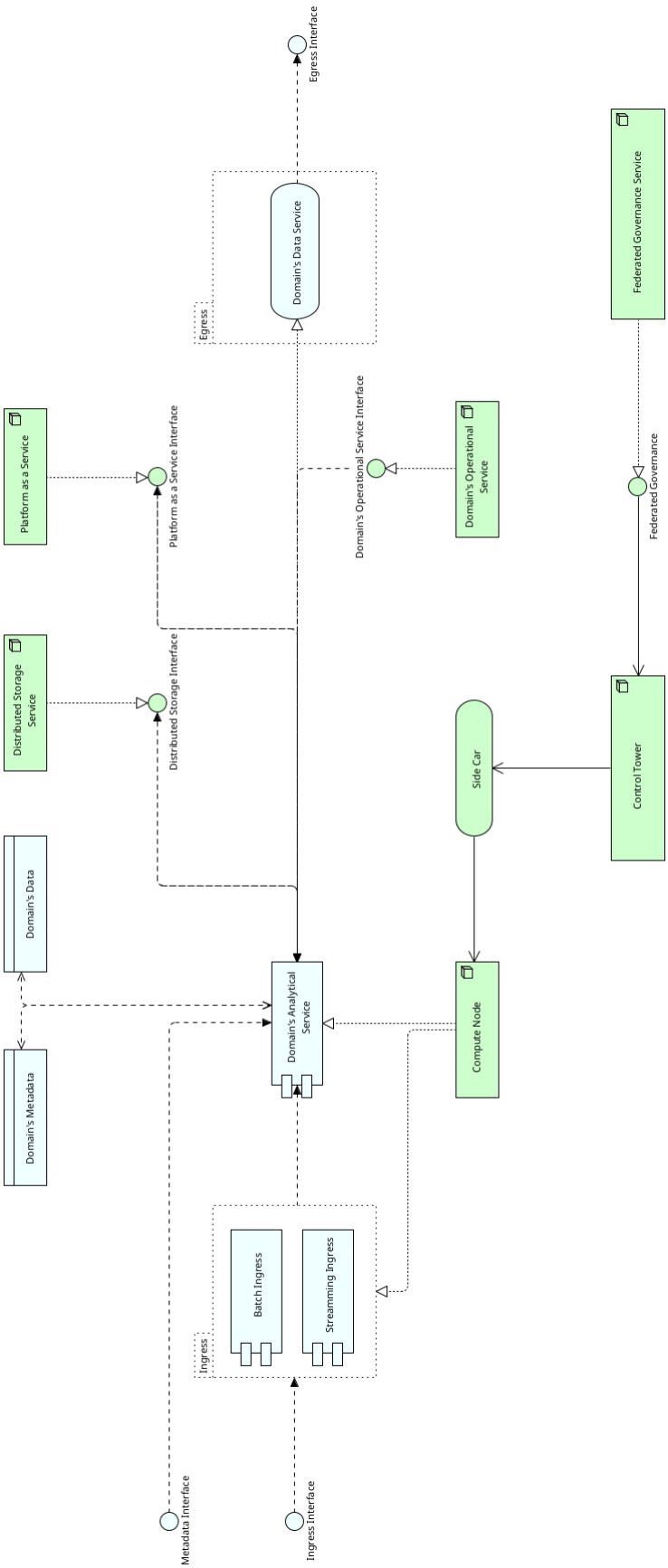

Figure 4: Service Mesh

ances 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 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 customizations 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 pivotal role in the architecture contributes to its overall robustness, security, scalability, performance, flexibility, and ease of monitoring.

2. **Batch Processing Controller:** The batch processing controller main job is to handles batch events by dispatching them to the event back-

bone. 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 customization and flexibility in producing batch events. It can perform additional tasks beyond compute-intensive operations, such as data scrubbing or adding custom headers. This customization 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 optimization. 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 optimized 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 optimize 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 exam-

ple, 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 analyze 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, optimize 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 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 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. 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:** The inclusion of an egress gateway in Metamycelium 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 authorisation component. This centralised approach brings several ben-

efits 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 authorisation 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 authorized 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 authorisation, 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 authorisation 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 de-

sign 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 optimizations 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).

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. 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 [**?**].
   (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 [**?**] 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.
10. **Telemetry Processor:** If all services employ the idea of localized 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 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 **??** 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 analyze logs, metrics, and traces from various sources. This centralization 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 optimization by leveraging data from dif-

ferent 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, centralizing 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.

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 deallocation 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 authorisation. It provides robust mechanisms to authenticate users, components, and services within the architecture, ensuring that only authorized 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 centralization 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 5.

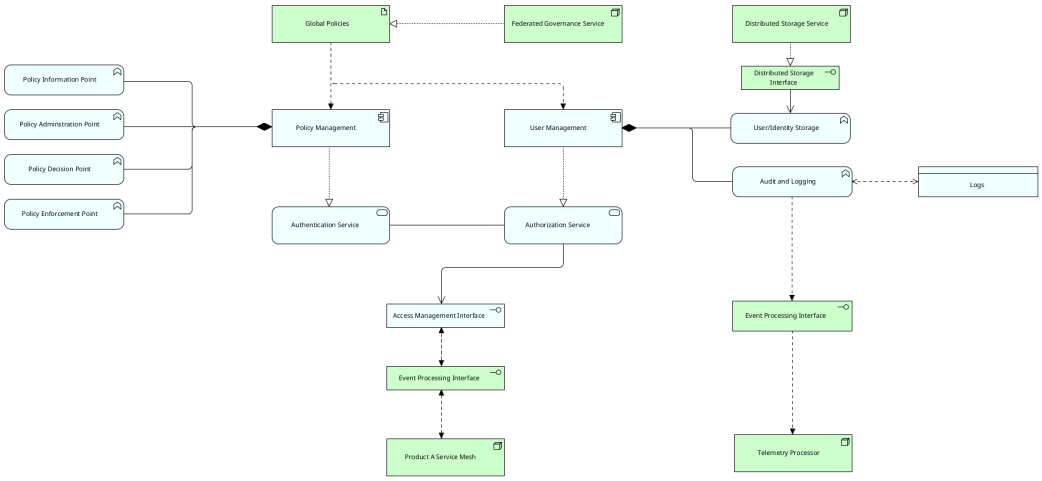

Figure 5: Identity and Access Control Management System

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 authorized 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 authorized 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 var-

ious data domains. This integration value facilitates secure and convenient access to secrets for authorized components, ensuring smooth operation and reducing friction in the development and deployment processes. This component 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.

9. Discussion

10. Conclusion