A domain-driven distributed reference architecture for big data systems

1st Pouya Ataei
School of Engineering, Computer and Mathematical Science
Auckland University of Technology
Auckland, New Zealand
pouya.ataei@aut.ac.nz

2nd Alan Litchfield
Service and Cloud Computing Research Lab
Auckland University of Technology
Auckland, New Zealand
alan.litchfield@aut.ac.nz

***Abstract*—The proliferation of digital devices, rapid development of software, and the availability of infrastructure augment the capability of users to generate data at an unprecedented rate. The accelerated growth of data has precipitated a shift in the industry which we perceive as the era of big data. The era of big data began when the variety, velocity and volume of data overwhelmed traditional systems, and induced a paradigm shift in data engineering. While companies have attempted to harness the benefits of big data, success rates are still low. Due to challenges such as rapid technology change, organizational culture, complexities of data engineering, impediments to system development, and a lack of effective big data architectures, it has been estimated that only 20% of companies have successfully institutionalized big data. By introducing a domain-driven distributed big data reference architecture, this study addresses issues in big data architecture, data engineering, and system development. The reference architecture is empirically grounded and evaluated through deployment in a real-world scenario as an instantiated prototype, solving a problem in practice. The results of the evaluation demonstrate utility and applicability with thought-provoking architectural tradeoffs and challenges.**

***Index Terms*—Big data architecture, data architecture, big data, domain-driven distributed big data systems**

I. Introduction

The ubiquity of digital devices and software applications allow users to generate data at an unprecedented rate. Almost all aspects of human life is integrated with some sort of software system that is performing computational processes on data. The rapid expansion and evolution of data from a structured element that is passively stored in the database to something that is used to support proactive decision making for business competitive advantage, have dawned a new era, the era of Big Data (BD). The BD era emerged when the velocity, variety, and volume of data overwhelmed existing system capability and capacity effectively and efficiently process and store data [1], [2].

BD is the practice of crunching large sets of heterogenous data to discover patterns and insights for business competitive advantage [3]. Since the inception of the term, ideas have ebbed and flowed along with the rapid advancements of technology, and many strived to harness the power of BD. Nevertheless, there are many failed attempts, for example, as at 2021 only 13% of organizations surveyed succeeded in delivering on their data strategy [4] and 20–24% successfully adopted BD [5], [6]. Therefore, successful adoption of BD is scarce and apparent challenges include "cultural challenges of becoming data-driven", "BD architecture", "data engineering complexities", "rapid technology change", and "lack of sufficiently skilled data engineers" [7].

A big data system is motivated by an array of functional requirements and quality goals. But if this system is to be successful, it must achieve these functional requirements within an acceptable performance, availability, modifiability and cost parameters. The software architecture is the key ingredient in determining whether these goals are attainable, before colossal amount of resources are committed to it. The initial design, development and deployment of a BD system does not mean success. As system grows larger, data providers and data consumers increase, data variety expands, data velocity extends, and metadata becomes increasingly more challenging to handle. This means, only a handful of hyper-specialized data engineers would be able to understand the system internals, resulting in silos, burnt out and potential friction.

This creates a perfect ground for immature architectural decisions that result in hard-to-main and hard-to-scale systems, and raise high-entry blockage. Since ad-hoc BD designs are undesirable and leaves many engineers and architects in the dark, novel architectures that are specifically tailored for BD are required. To contribute to this goal, we explore the notion of reference architectures (RAs) and presented a domaindriven distributed software RA for BD systems.

II. Background

In this section, the current state of BD knowledge, identification of the study focus, and study objectives are provided. Two important elements are discussed, the current state of BD architectures and why RA is needed.

A. Overview of BD architectures

In this overview, three generations of BD architecture are presented. A key issue apparent in all these approaches is the underlying monolithic data pipeline architecture that channels all data without any consideration for data quality. Thus, data logically linked to specific domains are now lumped together and processed in one architectural quantum, creating impediments to scalability and maintainability.

Enterprise Data Warehouse Generally built around a monolithic data warehouse, Extract-Transform-Load (ETL) processes, and data analysis and visualization/presentation software. Enterprise Data Warehouses are typically designed against specific requirements that limits how BD characteristics may be exploited effectively. As the Enterprise Data Warehouse expands and data consumers and data providers increase, then ETL processes become increasingly difficult to maintain and manage. This results in slower transaction processing and increased dependency on a small group of specialized staff. Such dependency creates silos that creates friction within and between teams and organizational boundaries. Furthermore, the monolithic architecture makes scaling, maintenance, and the extension of non-functional scope (for example, the addition of new transformations or data structures) difficult and time consuming and thus, expensive.

Data Lake to address some of the issues with data warehouses in general, a new BD ecosystem emerged as the Data Lake. In this approach, there isn't much data transformation before data are written into the Data Lake and retrieved when needed. While Data Lake architectures address some issues with data warehouse architectures, such as the capability of handling data variety, optimization features may be lacking. For example, if the number of data consumers and providers increase beyond the system's capacity to efficiently and effectively handle queries and manage data, then a data swamp may be created. Furthermore, data quality decreases over time firstly because no data owners are linked to stored data, and secondly because the BD stack is managed by specialized data engineers that may become siloed with ever-increasing backlogs. Data engineers may have little awareness of the semantics and value of the data they are processing, how data are useful to the business, and which domain data belongs to. Consequently, data quality reduces making maintenance and scaling a daunting task.

Cloud Based Solutions Considering the sheer cost of running a data engineering cluster on-premise, the current talent gap faced in the market, and complexity of provisioning the infrastructure of ever-increasing data processing loads [3], companies may choose to deploy on the Cloud for their BD solutions. The current technological generation leans towards architectures that provide Lambda or Kappa for stream processing or batch processing, or frameworks that unify the two like Databricks or Apache Beam. Whereas this generation of BD architectures might bring reduced cost and complexity for data architects, it still suffers from the same architectural challenges of having no clearly defined data domains, siloed specialized data engineers, and a monolithic pipeline BD architecture.

To address these issues, we explore a domain-driven distributed RA for BD systems and propose an RA that addresses some of the challenges. The RA is inspired by the advances in software engineering architectures, and in specific microservices, domain-driven design, and reactive systems.

B. Why a Reference Architecture?

Software architecture is an artefact that aims to satisfy business objectives through a software solution that is adaptable, cost-efficient, maintainable, and scalable. In addition, it allows for the capture of design issues at an early stage in the development process. Whereas this practice can be applied to any class of system, it is particularly useful in the design and development of complex system such as BD [1]. Despite the known complexity of BD systems, the development, analysis, and design of an RA incorporate best practices, techniques, and patterns that supports the achievement of BD undertakings is possible. Therefore engineers and architects can better absorb the complexity of BD system development and make it tractable. This approach to system development is not new for practitioners of complex systems.

In Software Product Line (SPL) development, RAs provide generic artefacts that are instantiated and configured for specific system domains [8]. Also, To address complex and unique system design challenges in software engineering, Information Technology (IT) giants like IBM refer to RAs as the 'best of best practices' [9].

Furthermore, RAs are used to standardize an emerging domain, a good example of this is the *BS ISO/IEC 18384-1 RA for service-oriented architectures* [10] and in diverse environments like NASA space data systems [11].Therefore, RAs are effective for addressing complex BD system development, because: 1) RAs promote adherence to best practice, patterns, and standards, 2) RAs can endow the architecture team with increased openness and interoperability, incorporating architectural patterns that provide desirable quality attributes, 3) RAs serve as the locus of communication, bringing various stakeholders together.

III. Research Methodology

The study is comprised of two phases: the first is the identification of high level requirements of the artefact, and second applies a well-established methodology for building the RA. This section describes the process of the two phases and approaches were made.

A. Requirement Specification

Prior to RA modelling and design, the desired properties of the artefact are defined as requirements. System and software requirements range from a sketch on a napkin to formal (mathematical) specifications. Therefore, the kind of requirements for the purpose of this study are defined in an exploration of the body of evidence. An analysis of existing RAs developed for BD systems offers general requirements, the spectrum of BD RAs, and how they are designed [12].

Defining and classifying software and system requirements is a common subject of debate. Sommerville [13] classify requirements as three levels of abstraction; user requirements,

system requirements, and design specifications. These are mapped against user acceptance testing, integration testing, and unit testing. In this study, a more general framework provided by Laplante [14] is adopted. The adopted approach provides three categories as functional, non-functional, and domain requirements. The objective in this phase is to define the high-level requirements of BD systems, thus non-functional requirements are not fully explored.

After clarifying the type of requirements, prior research provides general requirements for BD systems. The 5Vs of Velocity, Veracity, Volume, Variety and Value are frequently referenced [15]–[17]. In an extensive effort, NIST Big Data Public Working Group embarked on a large scale study to extract requirements from variety of application domains such as Healthcare, Life Sciences, Commercial, Energy, Government, and Defense [18]. The result of this study was the formation of general requirements under seven categories. Volk et al. [19] categorizes nine use cases of BD projects sourced from published literature with a hierarchical clustering algorithm. Bashari et al. [20] focus on security and privacy requirements for BD systems, Yu et al. present modern components of BD systems [21], using goal oriented approaches, Eridaputra et al. [22] create a generic model for BD requirements, and Aljaroodi et al. [23] investigate general requirements to support BD software development.

By analyzing these studies and by evaluating the design and requirement engineering required for BD RAs, a set of high-level requirements based on BD characteristics is established. A rigorous approach to present software and system requirements that offers informal methods of model verification is identified because such methods are well established in the industry and academia [24]. The approach for representing functional requirements follows the guidelines in *ISO/IEC/IEEE standard 29148* [25]. The requirements representation is organized in system modes, where the major components of the system and then the requirements are described. This approach is inspired by the requirement specification expressed for NASA Wide-field InfraRed Explorer (WIRE) system [14] and the Software Engineering Body of Knowledge (SEBoK) [26].

The requirements are categorized by the major characteristics of BD (Table I); value, variety, velocity, veracity, volume [27], and security and privacy [20].

B. Artefact Development Methodology

There are several approaches to the systematic development of RAs. Cloutier et al [9] demonstrate a high-level model for the development of RAs through the collection of contemporary architectural patterns and advancements. Bayer et al [28] introduce a method for the creation of RAs for product line development called PuLSE DSSA. Stricker et al [29] present the idea of pattern-based RA for service-based systems and use patterns as first class citizens. Similarly, Nakagaw et al [30] present a four-step approach to the design and development of RAs. Influenced by ISO/IEC 26550 [31], Derras et al [8] present a four-phase approach for practical RA development in the context of domain engineering and SPL.

Additionally, Galster [32] and Avgeriou [33] propose a 6-step methodology with two primary concepts: empirical foundation and empirical validity. Taking all these into consideration, an empirically-grounded RA methodology proposed by Galster and Avgeriou provides the most appropriate methodology to meet the purposes of this study. Reasons being that this methodology for RA development is adopted more than any other, and that the methodology supports the objectives of this study.

Nevertheless, this methodology suffered from a few limitations and we had to augment several areas of it with other approaches in order to arrive at the desired rigour and relevance. For instance, we could not find comprehensive guidelines on how to collect empirical data in step 3 of the methodology. We did not know what kind of theory we should collect and how should we model the data collected. Thereby, we employed Nakagawa et al's information source investigation guidelines and the idea of RAModel [39].

Also, the methodology does not describe how to evaluate the RA, thus, a more systematic and stronger evaluation approach is required. To address this, an instantiation the RA is deployed in a real world practice, then the Architecture Tradeoff Analysis Method (ATAM) [34] is applied to evaluate the artefact. The methodology constitutes 6 steps: 1) Decision on 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.

1. **Decision on type of the RA** Prior to the development of the RA, the type needs to be identified. The type determines the structure, the data to be collected, and the objective of the RA. To determine the type of RA, a classification framework provided by Angelov et al [35] is applied and a Systematic Literature Review (SLR) [2]. In this classification framework, RAs are categorized as standardization RAs and facilitation RAs. The domain-driven distributed RA chosen for the purposes of this study addresses two essential goals: 1) openness and interoperability between heterogenous components of a BD ecosystem and 2) facilitation of BD system development and data engineering. Accordingly, the output artefact is classified as 'classical standardization RA designed to be implemented in multiple organizations'.
2. **Selection of Design Strategy** According to Galster et al [32], RAs can have two major design strategies to them; 1) RAs that are based on existing patterns, principles and architectures, and 2) RAs that are developed from scratch. Designing RAs from scratch is scarce and usually happens in areas that have received no attention (could be BD in 2004). On the other hand, RAs are proven more successful when built from the proven practices [9].
   The RA developed for the purposes of this study is a research-based RA based on existing RAs, concrete architectures, patterns, standards and best practices.

TABLE I: BD system requirements

Volume	**Vol-1)** System needs to support asynchronous, streaming, and batch processing to collect data from centralized, distributed, and other sources, **Vol-2)** System needs to provide a 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 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 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.

3. **Empirical Acquisition of Data** Due to limitations witnessed by the Galster et al's methodology, we have augmented this phase to increase transparency and systematicity, by employing a SLR for data collection.
   This phase is comprising of three major undertakings; 1) identification of data sources, 2) capturing data from the selected sources and 3) synthesis of the selected data.
   Identification of Data Sources: For this sub-phase, we employed the ProSA-RA's 'information sources investigation'. To unearth the essential architectural quanta, we selected our sources as 'publications'. In order to capture the essence of existing body of knowledge from publications, we conducted a SLR. This SLR is an extension of Ataei et al work [2] which covered all the RAs by 2020. We followed the exact methodology as described in [2], but this time for the years 2021 and 2022. The main objective of this SLR is to find common architectural constructs among existing BD architectures, and highlight limitations.
   Capturing data from the selected sources: After having pooled quality literature, the study embarked in the process of capturing data. We used to software Nvivo for coding, labelling and classifying studies.
   Synthesizing Data: Codes highlighted patterns and patterns resulted in themes, which grounded the design theories necessary to build this RA. Our aim was to capture the body of knowledge and project it into our RA.
4. **Construction of the RA** Based on the requirements, themes, patterns and theories emerged from the previous step, the design and development of the RA initiated. Integral to this phase is the components that the RA should contain, how they should be integrated, how they should communicate, how do they address the requirements and what are the architectural tradeoffs.
   To describe the RA, we followed ISO/IEC/IEEE 42010 standard [36]. As suggested by the standard, we used Archimate to model our architecture.
5. **Enabling RA with variability** One of integral elements that help with instantiation of the RA is variability. This enables RA to remain useful as a priori artefact when it comes down to organization-specific regulations, and regional policies that may constrain the architect's freedom in design decisions. For the purposes of this study, we chose to represent variability by the means of "annotation" as recommended as on of the three accepted approaches by Galster et al [32].
6. **Evaluation of the RA** This last phase of the methodology is to ensure that RA has achieved its goal, and to test its effectiveness and usability. Based on Galster et al's [32] work, a quality of the RA can be deduced based on its utility, correctness and how efficiently it can be adopted and instantiated. Howbeit, the evaluation of the RAs is a known challenge among researchers [33], and while there are many well-established methods for assessing concrete architectures, many of these methodologies fall short in evaluating RAs.
   This is due to the inherent properties of RAs such as level of abstraction, and lack of clearly defined group of stakeholders. Various authors have attempted to solve this issue; for instance, Angelov et al's [37] attempt on modifying ATAM and extended it to resonate well with RAs. To this end, we first instantiated the RA, and then evaluate the prototype in practice, using ATAM.

IV. The Artefact

The RA is built underlying several principles: 1) Domain-driven: to address data quality, siloed teams, data swamp issues and communication issues, affecting velocity, variety, and veracity requirements; 2) Distributed: to address the challenges of scaling monolithic data systems affecting velocity, and volume requirements; 3) Data as a service: to allow for increased discoverability of data and autonomy of various analysis and data science teams without frictions with data

engineers affecting value, variety, veracity, security and privacy requirements; 4) Governance through a federated service: the prevent team-based and rather immature decisions that may not be in-line with global organizational visions, policies, standards and procedures, affecting all requirements; 5) Event driven: to address point-to-point communication issues that arises in distributed systems, affecting velocity requirement.

Lastly, it is worth mentioning that many of our designs have had inspirations from microservices architectural patterns [38]. Based on these principles, the RA designed for the purposes of this study is presented in 1. We have chosen this artefact to be domain-driven and distributed in order to shift from collecting data in monolithic data lakes and data warehouses to converging data through a decentalized and distributed mesh of data products communicating through standard interfaces. This is directly addressing the limitations discussed with current BD architectures.

This RA is made up of 11 main components and 9 variable components, these components are briefly discussed as below:

1) **Ingress Service:** this service is responsible for controlling traffic into the system. Depending on the nature of the request, this service will load balance either into a batch processing controller or a stream processing controller. Ingress is an asynchronous load balancer designed to eliminate choke points, handle SSL termination, and provide with extra features such as named-based virtual hosting. This component addresses the requirements Vol-1, Vol-2, Var-1, Var-3, Var-4, Val-1, Val-3, Val-4, SaP-1 and SaP-2.
2) **Batch Processing Controller:** this controller is responsible for handling batch processes. That is, it is responsible for receiving request for batch processing, and communicating it to the event broker. Due to the batch nature of the requests, the controller can decide to achieve this in a bulk and asynchronous manner. This component addresses the requirements Vel-1, Val-1, and Val-2.
3) **Stream Processing Controller:** This controller achieves similar thing to the batch one, with a difference that it has to handle a different nature of requests. Stream events are synchronous in nature and require hight through-put. Having a specific service for stream processing requirements promote tailored customization that best suit the varying nature of stream events. This component addresses the requirements Vol-1, Vel-1, Vel-2, Vel-4, Vel-5, Val-2,
4) **Event Broker:** event broker is an important architectural construct designed to achieve 'inversion of control'. As the system grows, more nodes and services are added, communication channels increase, and there is a need fore new events to be dispatched. As each service communicates through the event backbone, each service will be required to implement its own event handling module. This can easily turn into a spaghetti of incompatible implementations by different teams, and can even result in unexpected behaviors. To address this issue, an event broker is introduced to each service which has one main responsibility; communication with event backbone. This component indirectly addresses the requirements Val-1, and Ver-1.
5) **Event Backbone:** Event backbone is the heartbeat of the system, facilitating communication between all services. Where as this service is displayed as one technology service in the Archimate diagram, we recommend the event backbone to be designed underlying distributed paradigms itself. This is to ensure scalability as the number of topics and events grows. Event backbone and its relations to other nodes is analogous to a dance troupe; in a dance troupe, the members respond to the rhythm of music by moving according to their specific roles; the same happens here, with a difference that this time, event backbone is the music and services are the members of the dance troupe. This implies that services are only responsible for dispatching events in a 'dispatch and forget' model, subscribing to topics they are interested. 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 Service:** This services is responsible for providing necessary APIs to the consumers of the system, third parties or other BD systems. This allows for the openness of the architecture, and lets data scientist and business analyst easily request the data necessary for their work-loads. This also promotes the idea of self-serve-data through service discovery, data catalogue and product domains. This component can also be tuned for QoS networking, and other low computational functions if needs be. This component addresses the requirements Vel-2, Vel-4, Val-3, Val-4, SaP-1, and SaP-2.
7) **Product Domain Service Mesh:** Driven by the idea of domain-driven design, every product has it is own bounded context and ubiquitous language, and is technically governed by a service mesh. Every service mesh is made up of a batch ingress, stream ingress, BD storage, BD processing framework, domain's data service, together with the control tower and the side cars. These components provides the necessary means for the domain to achieve its ends in regards to BD processing. This is to enable high cohesiveness, low coupling and clear interfaces among services. This component indirectly addresses Vol-1, Vel-3, Vel-4, Vel-5, Var-1, Var-2, Var-3, Val-1, Val-2, Val-3, Val-4, Sap-1, SaP-2, Ver-1, Ver-2, and Ver-3.
8) **Federated Governance Service:** Given the distributed nature of the architecture and sheer number of moving parts with varying life-cycles; there is a need for some global contextual standards and policies that are designed to streamline processes and avoid losses. This is not to limit the autonomy of teams, but to inject them with best practices and organizational policies that tend to reflect the capability framework, regional limitations, and legal matters that can cause sever damage to the

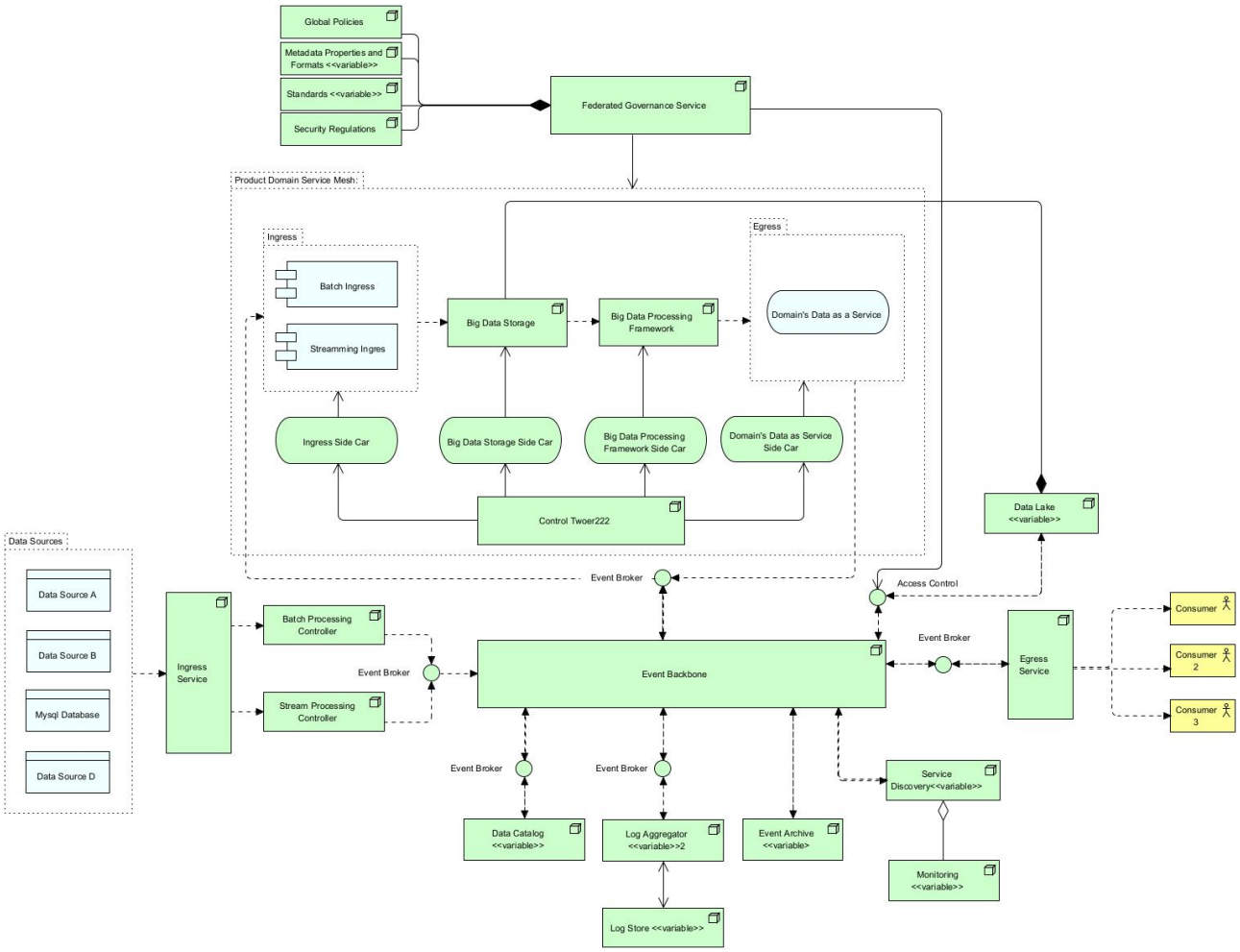

Fig. 1: domain-driven Distributed RA

business. This component can indirectly affect all requirements.

9) **Data catalog:** As data products increase in the system, more data become available, interoperability increases, and thus services have to know who provides what data. Data catalog is responsible for keeping a catalog of all data available among services with relative paths to fetch those data. This component addresses the requirements Vel-4, Var-1, Var-3, and Var-4.
10) **Log Aggregator and Log Store:** Operating underlying a distributed paradigms, requires a shift in a way that logging occurs. This means system cannot rely only one applications reporting logs in a single environment, but there's a need for a distributed tracing that shows a lifecycle of a process and how it went through different services. Therefore this RA benefits from the popular log aggregator pattern initially released by the microservices community, to allow fo graceful scaling of system's logging strategy. This component indirectly addresses the requirements Vol-1, Vel-1, Val-1, and Ver-1.
11) **Event Archive:** One of the main challenges of this architecture is it is reliance on event backbone. Whereas event backbone itself is recommended to be distributed and fault tolerant, event archive further solidifies the service recovery from unexpected events. This implies that, if the event backbone went out of service, the history of events can be stored and retrieved from the event archive to bring various services to the current state of operation. This component indirectly addresses the requirements Vol-1, Vel-1, Val-1, and Ver-1.
12) **Data Lake:** Whereas product domains are demarcated and boundaries are well-defined, we do not find it necessary for each domain to maintain it is own data lake. This is under the assumption that a lot of data are now processed at the time of storage, and is required whenever there is a analytical business case for it. Whereas there isn't a data lake per domain, different domains can have a quota in the data lake that is owned and handled by access control. This component addresses the requirements Vol-2, Vel-1, Var-1, Var-3, Var-4, Val-3.
13) **Service Discovery:** In a distributed setup like the one

portrayed, services need to find each other in order to communicate their means.
Service discovery solves this issue with primary responsibility of identifying services and answering queries about services. This is achieved by services registering themselves to service discovery on boot up. This component indirectly addresses the requirements Vel-2, Vel-4, Var-2, Var-4, Val-3, Val-4, SaP-2.

14) **Monitoring:** Last, but not least, in order to take proactive measures for the overall health of the system and its considerable moving parts, one needs to actively monitor the state of the individual nodes and the overall flow of things. Services emit large amounts of multi dimensional telemetry data that can be read and analyzed for the supporting actions. Monitoring services help storing these data to fuel proactive actions. This component indirectly addresses all requirements.

The elements of the RA that are annotated with the phrase 'variable' can be modified, adjusted or even omitted based on the architect's decision.

EVALUATION

Of utmost importance to development of the RA, is the evaluation of it. Our aim is to evaluate the correctness and utility of the RA by how it can turn into a context-specific concrete architecture that solves an actual problem. For this purpose ATAM has been chosen. ATAM has been chosen because it's got a good pedigree both in academia and industry, and has been applied to variety of architectures in different scales [39].

Using ATAM increased our confidence by uncovering key architectural tradeoffs, risks, and sensitivity points. While ATAM is usually conducted by an outside team after the architecture is created, we tailored it to the requirements of our study. We first created a prototype of our reference architecture, and then followed the ATAM steps to evaluate our prototype in a real-world setup. This approach helped us understand the consequences of our architectural decisions in a rigorous manner.

It is important to note that, this wasn't a setup in which an outside evaluation team would come to a company to evaluate an architecture in practice, but it was our prototype that we brought into a company to test its utility and relevance. While we could have achieved this with technical action research, we found ATAM to be in-line with our conceptual constructs, which is an architectural construct. ATAM provided us with a framework to discuss architectural concepts in a rigorous way [40].

For instantiation of the RA, We utilized ISO/IEC 25000 SQuaRE standard (Software Product Quality Requirements and Evaluation) [41] for technology selection.

We chose Node JS for all APIs and custom scripting, Nginx as our ingress, AWS Lambdas for stream and batch processing controllers, Kafka for event backbone, Kafka event brokers as the event broker, AWS application load balancer as the egress load balancer, Istio as the control tower, Envoy as the side car, Kubernetes as the container orchestrator, AWS S3 as the BD store and event archive, and Data Bricks for stream and batch processing.

We aimed to incorporate most components of our RA into this instance, however logging, monitoring, service discovery, federated governance service, and data catalog has been omitted. For brevity purposes, we dot not expand on ATAM steps in detail or the benefits of it, and we only explain how the evaluation has been conducted. Some details of this evaluation is omitted to protect the security, and intellectual property of the practice, and some details are modified for academic purposes. These modifications have not affected the integrity of the evaluation.

A. Phase 1:

Evaluation has taken place in a subsidiary of an international large-scale company. The subsidiary company is specialized in practice management software for veterinary professionals visa Software as a Service (SaaS) providing services to hospitals all around the globe, among which is some of the biggest equine hospitals, and universities. The company has several ambitions, one of which is big data an AI.

Following the guidelines of ATAM, the first step was the identification of relevant stakeholders. Our emphasis was on key stakeholders such as lead architects. This was important to ensure that we have not missed anything major in our design. We also opted not to include stakeholders that do not directly correlate with the prototype, such as the UI/UX designer. As a result, we invited two lead development architects, head of product, a product owner responsible for the product in which the artifact is tested, a quality assurance engineer and several developers.

1) Introductory Presentations: During the initial meeting, in step 1, ATAM was presented with clear description of its purposes. In step 2, stakeholders discussed the background of the business and some of the challenges faced, and the current state of affairs, the primary business goals, and architecturally significant requirements. In step 3, the prototype has been presented, our assumptions have been stated, and variability points portrayed.

2) Identifying Architectural Approaches: In this step, we discussed architectural styles in regards to quality attributes. For availability, we discussed Kafka's partitions, Nginx worker connections, Data Lake and Istio. For performance, we discussed Nginx asynchronous processing, Kafka topics and consumers, AWS application load balancer, and Kubernetes deployments. For modifiability, we discussed the concept of domain-driven design, side cars, and event brokers. We then continued to analyze these approaches for tradeoffs, sensitivity points, and potential risks.

3) Utility Tree Elicitation: In order to generate the utility tree, we first needed consensus on the most important quality attributes for this evaluation. We presented our assumptions and after a discussion, despite the fact that there were some concerns over privacy, the members agreed on availability, performance, and maintainability as the most important quality

attributes. Based on that, we created the utility tree with the requirements of; 1) performance: system should be able to process real time streams under 1200 ms, queries from data scientist should not take more than 2 hours 2) availability: load balancer and data bricks cluster being available 99.999% of the time, and 3) modifiability: new product domain should be added with less than 5 person in a month time.

We skipped the preliminary analysis of architectural approaches in phase 1 due to resources constraint, and only conducted it after the scenarios have been prioritized. This has not affected our overall evaluation process negatively.

B. Phase 2:

1) Scenario Prioritisation: Scenarios are the quanta of ATAM, and help capturing stimuli to which the architecture has to respond. Based on this premise, in this step, we asked stakeholders to priorities three class of scenarios namely 1) growth scenarios, 2) use-case scenarios and 3) exploratory scenarios. As a result of this we pooled 20 scenarios, which we then asked stakeholders to vote on. The voting process yielded 5 scenarios, described as two user journeys; 1) The pet owner brings the pet to the veterinary hospital, the pet is diagnosed with cancer. The pet's environmental factors should be studied for potential clues on the root cause of cancer; 2) A pet owner brings the pet to the veterinary hospital, the cat symptoms should be processed for early detection of lyme disease.

2) Analyze Architectural Approaches: After identifying architectural approaches and prioritizing scenarios, we ran the scenarios against our prototype. This is to provide heuristic qualitative analysis and point out sensitivity points and tradeoffs. We initiated this process by creating a custom script that extracts actual data from the company's mySQL database, and send it through the ingress. We created the necessary topics for Kafka, and configured Nginx to pass the requests to responsible lambdas for batch and stream processing. We then followed with event producers, Istio, Envoy, Kubernets, Data Bricks and the rest of the system. We then explained how our architectural decisions contribute to realizing each scenario.

C. Present the Results

In the process of running the scenario simulations against our system, we constantly probed our architectural approaches. Many implementation details arose in the process, and many sensitivity points annotated. We realized the true cost of the system, its trade offs and potential challenges. Based on these premises, and stakeholder feedbacks, we deduced that system quality Q_S, is a function f of the quality attributes performance Q_P, availability Q_A, and modifiability Q_M, as the equation Q_S = fQ(Q_P, Q_A, Q_M).

For performance, we used the cloud stress testing agent called StressStimulus. We ran the stress test against our system a couple of times, which revealed some interesting insights. It became evident that cold start time (100-1000ms) of AWS Lambdas can affect the desired performance. On the other hand, using Data Bricks for stream processing, we opted not to use micro-batch to have an accurate evaluation, we also decided not to configure the fair scheduling pool, so as to test the worst case scenario.

After analyzing the prototype with various performance models (periodic data dispatch, large volume of data, many concurrent requests), it became evident to us that latency, input/output, and object mutations were the performance sensitivity points. The event driven nature of the system really shined at handling various simulations. Based on these findings, we characterize system's performance as Q_P = h(l, s, c). That is, system is sensitive to latency (l), side effects (s), and concurrency (c).

Next, we tested our prototype from availability point of view. Since the system is distributed in nature, the failure in one service, if not handled properly, can have a ripple effect on all services. Our prototype could easily recover from such situation through implementation of circuit breakers in event brokers. Our prototype also archived the events before the failure occurred in the event backbone, to bring back the system to a correct state. Moreover, we set up health checks and alarms on pods in the Kubernetes cluster and constantly monitored for system behaviors.

Kubernetes made sure that certain amount of services are always available through special services called deployments and replica sets. This evaluation has also made us realize that our architecture is compliant with twelve factor methodology and can be deemed cloud native. This has positively affected the availability score. Given all, we characterize the system's availability as $Q_{\mathrm{A}} = g(\mu_{\mathrm{C}}, \lambda_{\mathrm{E}}, \mu_{\mathrm{S}})$. That is, system availability is affected by the time it takes for circuit breakers to trip and become available again (μ_{C}), failure of the event backbone (λ_{E}), and the time it takes for the services to recover (μ_{S}), with g being fraction of time that system is operating.

Lastly, we tested our prototype from modifiability point of view. The distributed and domain-driven nature of our architecture allowed us to easily achieve the desired modifiability objectives and even beyond. Adding a new data domain only required us to extend our HCL module written in Terraform for our EKS cluster, and modify it with new Docker images. Brokers were also streamlined, so we could spin up a new broker within minutes. We did not have to worry about certification lifecycle as it was handled by Istio, Local Cert Manager and Let's Encrypt.

Taking all into consideration, we characterize system's modifiability as $Q_{\mathrm{M}} = s(K, K, D)$. That is, system modifiability is sensitive to Kafka provisioning, maintenance and configuration, Kubernetes maintenance, provisioning and configuration, and Databricks provisioning, configuration and maintenance, with s being the skill set required.

1) Tradeoff Points:: After clear analysis, two tradeoff points have been identified; 1) event brokers and the event backbone and 2) service mesh. One area that raised concerns was the event backbone, and how it might turn into a bloated architectural component analogous to ESBs in SOAs. However, given the distributed nature of the system, event archive, and event brokers, we do not find that likely to happen.

This architectural component is designed itself underlying the distributed principles and is responsible only for one function; communication among services. In addition, in the case of service outage, event archive can be utilized to retrieve the order of events, which in turn brings system to the correct state. Event brokers on the other hand, facilitate modifiability by providing native event handling mechanisms, but at a cost of one more layer and potential latency that comes with it. Given these, we deduced that event backbone while affecting performance and maintainability positively, can potentially have negative impact on availability and reliability.

On the other hand, we realized, event brokers have positive effect on modifiability and availability, without having much negative effect on performance. One other area that was discussed heavily was the service mesh. Many developers found a lot to be done before the service mesh can operate and be up and running. We argued that while the initial effort for bringing up a service mesh may sound daunting, modifiability is positively affected longitudinally. Service mesh also promoted the concept of clear interfaces, separation of concerns, and a well-defined bounded context. Based on that, we deduced that service mesh has affected modifiability positively, but it might affect performance negatively due to the network communications required among services.

Two limitations discussed among stakeholders was 1) complexity of implementing the design and 2) tail latency. For the former, we do not think that a distributed big data architecture should be simple; we simply do not recommend organizations to embark on this journey if they do not have the resources necessary to absorb the complexity. For the latter, we do not have a straight forward solution. This is a well known issue in the microservices community as well, and is addressed in our design by the means of fault-tolerant services.

V. Related Work

Application of RAs to solving some of the challenges of data architecture is not a new concept. In one instance, White House announced an initiative for BD research and development with more than $200 million USD. One of the results of this project was NIST BD RA (NBDRA) [18]. In the same vein, IBM [42], Microsoft [43], Oracle [44], SAP [45], ISO [46] published their own BD RAs. In addition, Lambda [47] and Kappa [48] while widely referred to as just BD architectures, are usually at the abstraction level of a RA.

In the realm of academia, there has been numerous efforts including a postgraduate master's dissertation [49] and PhD thesis [50] for creating BD RAs. In addition, few universities have published their own RA. For instance, university of Amsterdam published a BD architecture framework [51].

Last but not least, there has been numerous RAs developed recently for specific domains. These studies have been usually published as short journal papers, and many have promised future publication of the full RA as a book. For instance, Klein et al. [52] developed a BD RA in the national security domain, and Weyrich and Ebert [53] worked on a BD RA in the domain of Internet of Things (IOT). Lastly, there's been some efforts in adopting microservices architecture for BD systems such as Neomycelia [54] and Phi [55].

These RAs are prominent research, with great potential to induce concrete architectures. But with all, they are mostly published as short studies and provide with little information about quality attributes, data quality, metadata management, security, and privacy concerns. In another terms, they are notion or brief discussions on RAs in very particular domains.

Therefore, this study builds on available body of knowledge, and aims to address the limitations of current BD RAs by providing a domain-driven distributed architecture for BD systems. To the best of our knowledge, expect for the works of Ataei et al [54], no other RA has focused on logical separation of data into domains through event-driven communication and with clearly defined boundaries. Here then, we absorb the best of knowledge from both industry and academia and tend to provide with a next-generation BD RA that aims to absorb many advances of software engineering such as microservices, event driven and reactive systems.

VI. Conclusion

BD engineering is sophisticated process, and while there are many good practices institutionalized in software engineering, data engineering domain does not seem to benefit from all of it. This has led to several challenges in development of BD systems, and many companies have failed to bring to light the potential of a data-driven decision making. We aimed at facilitating this process in this study by proposing a BD RA. Nevertheless, there's more and more research required in the area of data processing, reactive event driven data processing system, data engineering and BD architectures. Some areas that need considerable attention is security, privacy, and metadata management for BD architectures.

References

[1] P. Ataei and A. Litchfield, "Neomycelia: A software reference architecturefor big data systems," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2021, pp. 452–462.

[2] P. Ataei and A. T. Litchfield, "Big data reference architectures, a systematic literature review," 2020.

[3] B. B. Rada, P. Ataeib, Y. Khakbizc, and N. Akbarzadehd, "The hype of emerging technologies: Big data as a service," 2017.

[4] M. technology review insights in partnership with Databricks, "Building a high-performance data organization," 2021. [Online]. Available: https://databricks.com/p/whitepaper/mit-technology-review-insights-report

[5] N. Partners, "Big data and ai executive survey 2021," 2021. [Online]. Available: https://www.newvantage.com/thoughtleadership

[6] A. White, "Our top data and analytics predicts for 2019," 2019. [Online]. Available: https://blogs.gartner.com/andrew_white/2019/01/03/our-top-data-and-analytics-predicts-for-2019/

[7] B. B. Rad and P. Ataei, "The big data ecosystem and its environs," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 38, 2017.

[8] M. Derras, L. Deruelle, J.-M. Douin, N. Levy, F. Losavio, Y. Pollet, and V. Reiner, "Reference architecture design: A practical approach," in *ICSOFT*, Conference Proceedings, pp. 633–640.

[9] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone, "The concept of reference architectures," *Systems Engineering*, vol. 13, no. 1, pp. 14–27, 2010.

[10] I. Iso, "Information technology — reference architecture for service oriented architecture (soa ra) — part 1: Terminology and concepts for soa," *International Organization for Standardization*, p. 51, 2016. [Online]. Available: https://www.iso.org/standard/63104.html

[11] NASA, "Reference architecture for space data systems," 2008.

[12] P. Ataei and A. T. Litchfield, "Big data reference architectures, a systematic literature review," 2020.

[13] I. Sommerville, *Software Engineering, 9/E*. Pearson Education India, 2011.

[14] P. A. Laplante, *Requirements engineering for software and systems*. Auerbach Publications, 2017.

[15] M. Bahrami and M. Singhal, *The role of cloud computing architecture in big data*. Springer, 2015, pp. 275–295.

[16] B. B. Rad and P. Ataei, "The big data ecosystem and its environs," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 38, 2017.

[17] H.-M. Chen, R. Kazman, and S. Haziyev, "Agile big data analytics development: An architecture-centric approach," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 2016, Conference Proceedings, pp. 5378–5387.

[18] W. L. Chang and D. Boyd, "Nist big data interoperability framework: Volume 6, big data reference architecture," Report, 2018.

[19] M. Volk, D. Staegemann, I. Trifonova, S. Bosse, and K. Turowski, "Identifying similarities of big data projects–a use case driven approach," *IEEE Access*, vol. 8, pp. 186 599–186 619, 2020.

[20] B. Bashari Rad, N. Akbarzadeh, P. Ataei, and Y. Khakbiz, "Security and privacy challenges in big data era," *International Journal of Control Theory and Applications*, vol. 9, no. 43, pp. 437–448, 2016.

[21] J.-H. Yu and Z.-M. Zhou, "Components and development in big data system: A survey," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 51–72, 2019.

[22] H. Eridaputra, B. Hendradjaya, and W. D. Sunindyo, "Modeling the requirements for big data application using goal oriented approach," in *2014 international conference on data and software engineering (ICODSE)*. IEEE, 2014, pp. 1–6.

[23] J. Al-Jaroodi and N. Mohamed, "Characteristics and requirements of big data analytics applications," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2016, pp. 426–432.

[24] M. Kassab, C. Neill, and P. Laplante, "State of practice in requirements engineering: contemporary data," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 235–241, 2014.

[25] I. 29148:2018, "Iso/iec 29148:2018," 2018. [Online]. Available: https://www.iso.org/standard/72089.html

[26] A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. Tripp, "Software engineering body of knowledge," *IEEE Computer Society, Angela Burgess*, p. 25, 2004.

[27] B. B. Rada, P. Ataeib, Y. Khakbizc, and N. Akbarzadehd, "The hype of emerging technologies: Big data as a service," 2017.

[28] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud, "Pulse: A methodology to develop software product lines," in *Proceedings of the 1999 symposium on Software reusability*, 1999, pp. 122–131.

[29] V. Stricker, K. Lauenroth, P. Corte, F. Gittler, S. De Panfilis, and K. Pohl, "Creating a reference architecture for service-based systems–a pattern-based approach," in *Towards the Future Internet*. IOS Press, 2010, pp. 149–160.

[30] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. Oquendo, "Consolidating a process for the design, representation, and evaluation of reference architectures," in *2014 IEEE/IFIP Conference on Software Architecture*. IEEE, 2014, pp. 143–152.

[31] I. WG, "Iso/iec 26550: 2015-software and systems engineering–reference model for product line engineering and management," *ISO/IEC, Tech. Rep*, 2015.

[32] M. GALSTER and P. AVGERIOU, "Empirically-grounded reference architectures," *Joint ACM*.

[33] P. Avgeriou, "Describing, instantiating and evaluating a reference architecture: A case study," *Enterprise Architecture Journal*, vol. 342, pp. 1–24, 2003.

[34] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No. 98EX193)*. IEEE, Conference Proceedings, pp. 68–78.

[35] S. Angelov, P. Grefen, and D. Greefhorst, "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*. IEEE, 2009, pp. 141–150.

[36] I. International Organization for Standardization (ISO/IEC), "Iso/iec/ieee 42010:2011," 2017. [Online]. Available: https://www.iso.org/standard/50508.html

[37] S. Angelov, J. J. Trienekens, and P. Grefen, "Towards a method for the evaluation of reference architectures: Experiences from a case," in *European Conference on Software Architecture*. Springer, 2008, pp. 225–240.

[38] C. Richardson, *Microservices patterns: with examples in Java*. Simon and Schuster, 2018.

[39] R. K. Len Bass, Dr. Paul Clements, *Software Architecture in Practice (SEI Series in Software Engineering) 4th Edition*. Addison-Wesley Professional; 4th edition, 2021.

[40] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[41] ISO, "Iso/iec 25000:2005," 2014.

[42] D. Quintero, F. N. Lee *et al.*, *IBM reference architecture for high performance data and AI in healthcare and life sciences*. IBM Redbooks, 2019.

[43] B. Levin, "Big data ecosystem reference architecture," *Microsoft Corporation*, 2013.

[44] D. Cackett, "Information management and big data, a reference architecture," *Oracle: Redwood City, CA, USA*, 2013. [Online]. Available: https://www.oracle.com/technetwork/topics/entarch/articles/info-mgmt-big-data-ref-arch-1902853.pdf

[45] "Sap - nec reference architecture for sap hana & hadoop," 2016. [Online]. Available: https://www.scribd.com/document/418835912/Whitepaper-NEC-SAPHANA-Hadoop

[46] I. O. for Standardization (ISO/IEC), "Iso/iec tr 20547-1:2020," 2020. [Online]. Available: https://www.iso.org/standard/71275.html

[47] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 2785–2792.

[48] J. Lin, "The lambda and the kappa," *IEEE Internet Computing*, vol. 21, no. 05, pp. 60–66, 2017.

[49] M. Maier, A. Serebrenik, and I. Vanderfeesten, "Towards a big data reference architecture," *University of Eindhoven*, 2013.

[50] U. Suthakar, "A scalable data store and analytic platform for real-time monitoring of data-intensive scientific infrastructure," Ph.D. dissertation, Brunel University London, 2017.

[51] D. N. B. D. I. Framework, "Draft nist big data interoperability framework: Volume 5, architectures white paper survey," *NIST Special Publication*, 2015.

[52] J. Klein, R. Buglak, D. Blockow, T. Wuttke, and B. Cooper, "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)*. IEEE, Conference Proceedings, pp. 51–57.

[53] M. Weyrich and C. Ebert, "Reference architectures for the internet of things," *IEEE Software*, vol. 33, no. 1, pp. 112–116, 2015.

[54] P. Ataei and A. Litchfield, "Neomycelia: A software reference architecturefor big data systems," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2021, pp. 452–462. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/APSEC53868.2021.00052

[55] A. Maamouri, L. Sfaxi, and R. Robbana, "Phi: A generic microservices-based big data architecture," in *European, Mediterranean, and Middle Eastern Conference on Information Systems*. Springer, 2021, pp. 3–16.