DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Assessing the Cost and Benefit of a Microservice Landscape Discovery Method in the Automotive Industry

Nektarios Machner

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Assessing the Cost and Benefit of a Microservice Landscape Discovery Method in the Automotive Industry

Bewertung der Kosten und des Nutzens einer Methode für die automatisierte Rekonstruktion von Microservice Architekturen in der Automobilbranche

Author:	Nektarios Machner, B.Sc.
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Martin Kleehaus, M.Sc.
Submission Date:	15.11.2019

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.11.2019 Nektarios Machner, B.Sc.

Acknowledgments

It has been a long journey, but it has finally come to an end. So at this point, I would like to take the opportunity to thank a few people without whom this thesis would not have been possible. First and foremost, I would like to thank my family and friends for always believing in me and supporting me throughout the duration of writing this thesis. Secondly, I would like to extend my gratitude to my advisor Martin Kleehaus from TUM who provided invaluable guidance in designing and writing the thesis, but also Jan Schäfer and Stefan Volkert from our industry partner who made this thesis possible and helped finding interview partners for the evaluation. Speaking of interview partners, I also want to express my gratitude to these people who took time out of their day to diligently answer all of my questions. Last but not least, I would like to thank the SEBIS chair headed by Prof. Dr. Florian Matthes, which this thesis was written at. And before I close off this section, in a time of rising mental health issues, I believe it is of utmost importance to always love yourself and therefore, I would like to end this section by also thanking myself for having the discipline to see this thesis through to the end.

Abstract

Enterprise Architecture Management is a widespread practice in today's organizations all over the world. Part of this managerial function is to document the current IT landscape in order to provide valuable information, on which decisions are based regarding the future orientation of the enterprise. Due to the rising adoption of trends like DevOps and microservice-based architectures, Enterprise Architecture Documentation (EAD), which is still performed manually to a large extent, faces various challenges that need to be overcome.

Recent research endeavors explored approaches that assist the documentation process in an automated manner through the usage of runtime data in order to minimize the manual effort and improve the overall quality of available information.

In this thesis, a novel approach to leverage the automated extraction of runtime data from an Application Performance Monitoring (APM) tool and reconstruct the as-is IT landscape is implemented in a real world environment situated in the automotive industry. The implemented IT artifact is tested and evaluated through expert interviews with EA practitioners from our industry partner.

The proposed solution approach shows promising results and is able to automatically extract EA-related information from runtime data and provide various visualizations for exploring the reconstructed IT landscape.

Keywords: Enterprise Architecture Management (EAM), Enterprise Architecture Documentation (EAD), Application Performance Monitoring (APM), distributed tracing, automated discovery algorithm, automated runtime data extraction, automotive industry

Contents

1. Introduction

This chapter will introduce the motivation behind this thesis, present the research questions and the underlying research methodology and give an outline of the overall structure of the thesis.

1.1. Motivation & Problem Description

Enterprise Architecture Management (EAM) is a common practice in mid- and large-sized organizations with the purpose of managing the complexity of the enterprise's IT landscape in relation to the business requirements [1]. Companies with a solid foundation are found to have higher profitability, faster time to market, and lower IT costs [2]. Therefore, Enterprise Architecture as a foundation for execution, which depends on a tight alignment between IT capabilities and business objectives, takes on increasing strategic importance [2].

Part of the managerial function of EAM is the documentation and analysis of the organization's as-is landscape which builds the foundation for adapting to upcoming trends and deriving plans for future transformations.

In the recent years, various trends started to gain a lot of popularity. Trends such as microservices and DevOps are considered growing concepts with a comparable rate of growth since the year 2014 [3]. A multitude of well-known companies such as Amazon and Spotify utilize microservice architectures and as a consequence claim to have achieved scalability, agility and reliability [4].

Due to the rising complexity of IT landscapes, Enterprise Architecture Documentation (EAD) faces a set of challenges:

- EAD is performed mostly manually and is therefore a costly and time-consuming task. [1] [5] [6] [7] [8]
- The documentation is incomplete and/or outdated, which results in decisions made on the basis of low quality data. [1] [9]
- There are no clear responsibilities for documentation. [10]
- The IT landscape is constantly changing and evolving, too fast for manual documentation. [3] [6]

Recent research started investigating automated approaches as a chance to meet these challenges and facilitate the documentation process (cf. Related Work in section 2.3). To that effect, a novel approach is being developed at the SEBIS chair of the TUM, which leverages

runtime data from an Application Performance Monitoring (APM) tool and enriches it with information from federated information systems in order to reconstruct the EA landscape and assist its documentation. As part of this thesis, this approach will be implemented in a real world environment within the automotive industry and evaluated by EA practitioners from our industry partner. The aim is to critically assess the benefits and limitations of this solution approach and provide feedback for future research.

1.2. Research Questions

This master's thesis aims at answering the following research questions:

- **RQ1:** Which IT artifacts and their communication relationships can be discovered through runtime data?
- **RQ2:** What requirements need to be fulfilled for the proposed solution approach to function as intended?
- **RQ3:** What are benefits and limitations of the proposed solution approach?

RQ1 aims at identifying all relevant EA-related elements that can be extracted from runtime data in an automated fashion. The goal is to achieve an overview of which elements can be discovered and which ones can not so that further research can be done in exploring means how to close this information gap and provide more valuable information with reduced manual effort over time.

RQ2 is geared towards identifying various requirements that need to be fulfilled so that an automated approach can be successful in reconstructing the IT landscape and assist the documentation process.

RQ3 represents a conclusive evaluation of the proposed solution approach after implementing and testing it in a real world environment. To that effect, a quantitative and a qualitative analysis will be conducted including expert interviews with EA practitioners. As a result, benefits and limitations of the overall solution approach will be presented and assessed and ideas for further research to improve the solution approach will be discussed.

1.3. Research Methodology

Hevner et al. describe seven guidelines for design science in information systems research, which they derive from the principle of design-science research that "knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact." [11]

The first guideline (GL1) states that a viable artifact needs to be produced by the performed research [11]. This thesis complies with GL1 by building upon already existing research and producing a prototypical instantiation of the conceptualized solution approach.

The second guideline (GL2) requires the produced artifact to be relevant with respect to problems or issues encountered by IS practitioners [11]. The herein produced artifact is relevant to multiple problems as introduced in 1.1, e.g. in assisting the process of EA documentation, among other use cases.

The third guideline (GL3) calls for a profound evaluation with suitable methods [11]. As part of a case study, the designed artifact will be integrated into the productive environment of our industry partner from the automotive industry, whereby it will be subjected to a continuous cycle of adaptation and feedback from the corresponding advisors over a period of three months. The case study will be concluded with a quantitative analysis of multiple metrics and a qualitative analysis in the form of multiple expert interviews with practitioners from our industry partner.

The fourth guideline (GL4) requires the conducted research to contribute meaningfully to its respective field [11]. The research within this thesis primarily contributes to the field of IS research and EAM by providing a proof of concept (through construction and implementation in a real-world scenario) and an assessment of benefits and limitations of the proposed solution to be considered for future research.

The fifth guideline (GL5) dictates applying rigor when constructing and evaluating the artifact [11]. This thesis applies rigor by choosing technologies for the implementation of the IT artifact that have been tested and prevailed in the field and selecting relevant stakeholders and metrics for its evaluation.

The sixth guideline (GL6) states that design science needs to be viewed as a continuous search process [11]. In order to apply to this guideline, the implementation of the IT artifact will constantly be critically assessed and updated accordingly in search for the best possible solution.

The seventh guideline (GL7) recommends presenting the research in an effective way to a managerial as well as a technology-affine audience [11]. This thesis adheres to GL7 by introducing the conducted research and presenting the findings in a way that does not require a profound understanding of the problem domain, since the necessary knowledge is outlined within the thesis and enriched with references to the underlying literature for further reading, but by also including a chapter that discusses the technical implementation of the IT artifact in more detail. Additionally, the source code of the implemented artifact is attached for further inspection should technology-savvy readers be interested.

1.4. Outline

The remainder of this master's thesis is organized as follows:

Chapter 2 will cover the theoretical background of this thesis and introduce the conceptual foundation which will serve as a basis on which the proposed solution approach will later be built upon.

Chapter 3 will explain the implemented IT artifact in more detail and discuss a variety of technical matters including design choices and limitations of the proposed solution approach.

Chapter 4 will evaluate the implemented IT artifact within a quantitative analysis with regard to multiple metrics and a qualitative analysis where interviews conducted with experts from our industry partner will be assessed and their respective statements critically reflected upon.

Chapter 5 will summarize the findings, conclusively answer the stated research questions and discuss ideas for future work.

2. Concept

This chapter will cover the theoretical background of this thesis and introduce the underlying concept in more detail. The goal is to explain the foundations on which the implementation in chapter 3 will build upon.

2.1. Theoretical Background

2.1.1. Terminology

While the glossary will cover most of the terms used throughout this thesis, some terms need to be elaborated on more thoroughly for a deeper understanding and clarification.

- Application (software / program)

  In the literature and in the field, the term application is used excessively and very broadly without giving much thought to the meaning of the term. In general, an application can be seen as a logical grouping of functions to fulfill one or a set of tasks. Depending on the context, the actual meaning and provided functionality can differ largely. Therefore, the author will make an effort to always specify what kind of applications the term refers to when used throughout the thesis.

- Application Performance Monitoring (APM)

  APM can denote the terms Application Performance Monitoring and Application Performance Management. Usually, monitoring is part of the management function or in other words, in order to adeqately manage applications, applications need to be monitored first so that relevant information can be identified and collected. Unless otherwise specified, the acronym APM will refer to Application Performance Monitoring.

2.1.2. ArchiMate

ArchiMate is a EA modeling language developed and maintained by The Open Group, which was designed to provide a consistent representation for EA-related diagrams. The latest version of the specification is 3.1.[1]

The core framework is composed of three layers and three aspects which together form 9 cells as depicted in Figure 2.1.

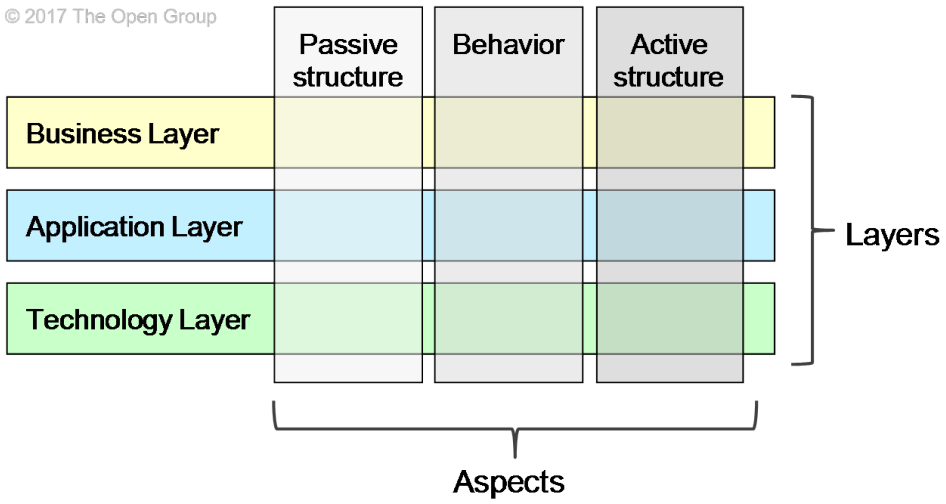


Figure 2.1.: ArchiMate Core Framework, taken from [12]

The aspects are not as important, but the different layers and their color coding are necessary to correctly interpret the models and understand the presented findings later in this thesis and shall be therefore defined and explained hereinafter.

- **Business Layer**

  "The Business Layer depicts business services offered to customers, which are realized in the organization by business processes performed by business actors." [12]

  This layer concerns itself with the overarching business perspective of an organization including elements such as domains and products. The elements found within this layer are understood on a logical level, technical details are abstracted.

- **Application Layer**

  "The Application Layer depicts application services that support the business, and the applications that realize them." [12]

  This layer represents the technical side of things and contains the elements necessary to provide all the business services defined in the business layer. While technical details are not abstracted, they are still understood on a logical level.

[1] pubs.opengroup.org/architecture/archimate3-doc/toc.html

- **Technology Layer**

  "The Technology Layer depicts technology services such as processing, storage, and communication services needed to run the applications, and the computer and communication hardware and system software that realize those services. Physical elements are included for modeling physical equipment, materials, and distribution networks to this layer." [12]

  As the definition states, the elements found within this layer build the necessary foundation on which the elements of the application layer run on.

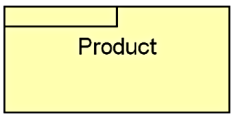
The ArchiMate language offers a variety of different elements, but only the ones used in this thesis will be defined hereinafter.

- **Business Function**

Business Function Notation

"Represents a collection of business behavior based on a chosen set of criteria (typically required business resources and/or competencies), closely aligned to an organization, but not necessarily explicitly governed by the organization" [12]

- **Product**



Product Notation

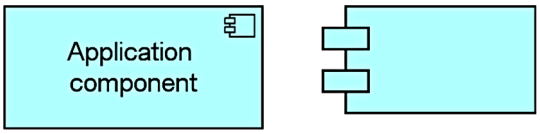
"Represents a coherent collection of services and/or passive structure elements, accompanied by a contract/set of agreements, which is offered as a whole to (internal or external) customers." [12]

- **Business Service**

Business Service Notation

"Represents explicitly defined behavior that a business role, business actor, or business collaboration exposes to its environment." [12]

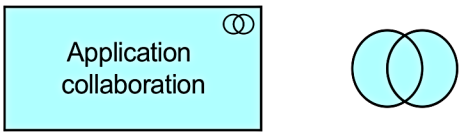
- **Application Component**



Application Component Notation

"Represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable." [12]

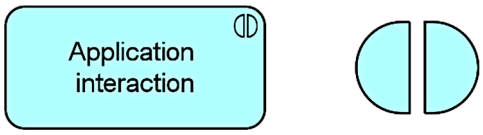
- **Application Collaboration**



Application Collaboration Notation

"Represents an aggregate of two or more application internal active structure elements that work together to perform collective application behavior." [12]

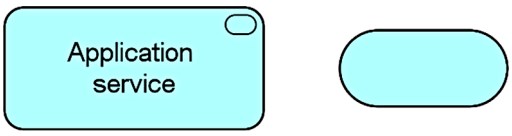
- **Application Interaction**



Application Interaction Notation

"Represents a unit of collective application behavior performed by (a collaboration of) two or more application components." [12]

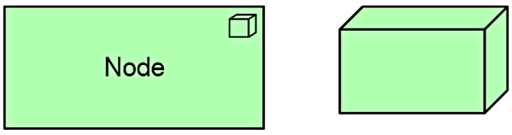
- **Application Service**



Application Service Notation

"Represents an explicitly defined exposed application behavior." [12]

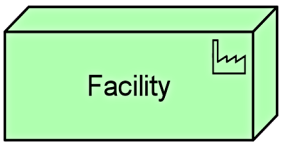
- **Node**



Node

"Represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources." [12]

- **Facility**



Facility

"Represents a physical structure or environment." [12]

2.2. Microlyze

This section will introduce Microlyze in more detail. Microlyze is a novel solution approach to discover and maintain EA-related information in order to assist and possibly one day replace manual EA documentation to some extent. It is designed and conceptualized at the SEBIS chair of the TUM in Munich and is based on microservice-based architectures.

2.2.1. Overview

Figure 2.2 shows an overview of the underlying ideas that will be explained hereinafter.

During the development process a JSON configuration file is created by the product owner and development team that contains static information about a corresponding microservice including references to federated information systems.

When a microservice is ready to be deployed, it is committed to the CI/CD pipeline which contains a test case that validates the included JSON configuration file against a JSON schema globally defined by enterprise/IT architects. The validation ensures the presence of such a configuration file and checks whether all necessary fields have been supplied with the according information. Only if this validation succeeds can the developed microservice be cleared for deployment. In that case, the configuration file is transferred to a centralized reference repository and stored for later use. Otherwise, the development team needs to amend the configuration file and retry the deployment process including the validation step.

Once a microservice is successfully deployed, it is continuously monitored by an Application Performance Monitoring tool that collects a plethora of relevant information in the form of runtime data. This data is stored at a monitoring server that can be queried through its exposed interfaces.

Microlyze includes algorithms that request relevant information from the monitoring server and reconstruct the as-is IT landscape by processing the received runtime data. The discovered IT artifacts are stored in a database for historization purposes.

By means of a graph-based query language it is then possible to query the database for the stored IT artifacts enriched with static information from the associated JSON configuration file that is retrieved from the reference repository and used to bridge the gap between the application layer and the business layer. The queried data can be visualized and displayed to the user through a dedicated frontend within any modern browser application.

2.2.2. Prerequisites

In order for Microlyze to function as intended, a set of prerequisites needs to be fulfilled.

1. Every microservice needs to be monitored with an APM tool.
2. A JSON configuration file needs to be created and maintained for every microservice.
3. Usage of a CI/CD pipeline is necessary to ensure the existence and validate the content of the configuration files by means of a separate test case.

4. Handling and maintenance of the configuration files needs to be delegated to agile teams (product owner and development team).

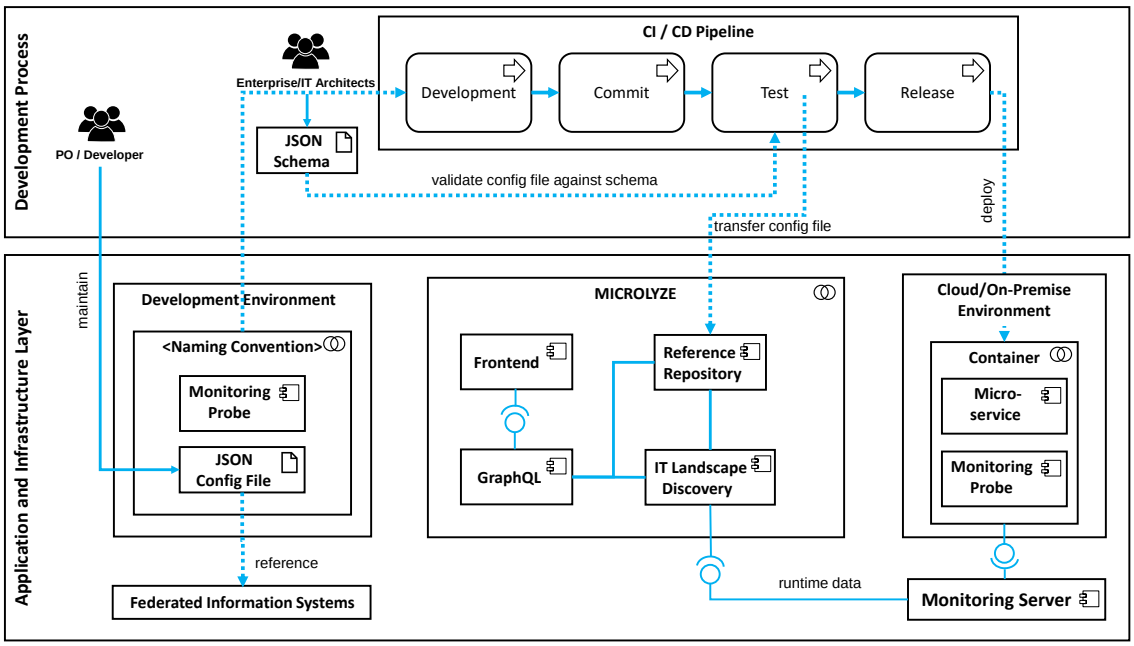


Figure 2.2.: Microlyze Overview, adapted from [13]

2.2.3. Theoretical Discovery Algorithms

Two algorithms were designed and formally described in [14]. These algorithms will build the basis for the implementation of the Automated Architecture Discovery Algorithm in chapter 3 and shall therefore be covered briefly in this section.

Assume the Enterprise Architecture $A(E, C)$ is a directed graph with a finite set of runtime artifacts E and communications C, whereby $C \subseteq E \times E$.

The backward discovery algorithm recursively retrieves historical tracing data (tD) by starting at t_0 and iterating through the past in pre-defined timeframes (T) until no further data is available due to storage limitations from the APM tool in use. Afterwards, the forward discovery algorithm is triggered and continuously processes new runtime data.

The overall idea is to incrementally reconstruct the as-is landscape by adding newly discovered elements to the EA graph and storing the data in a database for historization. Since runtime data can only contain information on what has been observed, there can be no guarantee that the discovered set of IT artifacts is complete. However, running these algorithms for a sufficiently long enough period of time should provide eventual consistency. For further information on the formal definition of the discovery algorithms please refer to [14].

Algorithm 1 Backward Discovery, taken from [14]

Require: $T > 0$

function BACKWARDDISCOVERY(A, t_0, T)
  if $A = \emptyset$ **then**
    $rD(E) \leftarrow$ REPOSITORYDATA
    $A\prime \leftarrow A(rD(E), C)$
  $t_1 \leftarrow t_0$
  $t_0 \leftarrow t_1 - T$
  $tD(E, C) \leftarrow$ TRACEDATA(t_0, t_1)
  if $tD \neq \emptyset$ **then**
    for all $e \in tD(E)$ **do**
      if $e \in rD(E)$ **then**
        $A\prime\prime \leftarrow A\prime(E, C \cup tD(C_e))$
    BACKWARDDISCOVERY$(A\prime\prime, t_0, T)$
  else
    return $A\prime$

Algorithm 2 Forward Discovery, taken from [14]

Require: $T > 0$, $\tau > 0$

function FORWARDDISCOVERY(A, τ, T)
  $t_1 \leftarrow t_0 + T$
  $rD(E) \leftarrow$ REPOSITORYDATA
  $tD(E, C) \leftarrow$ TRACEDATA(t_0, t_1)
  $A\prime \leftarrow A(E \cap rD(E), C)$
  $A\prime\prime \leftarrow A(E\prime, C \cup tD(C))$
  for all $c \in A\prime\prime(C\prime)$ **do**
    if $c(lastSeen) + \tau \leq t_0$ **then**
      $c(deleted) \leftarrow true$
  $V(i+1) \leftarrow A\prime\prime$
  return $A\prime\prime$

2.3. Related Work

This section will briefly cover relevant related work regarding the usage of runtime data and automation to reconstruct the IT landscape.

- Buschle et al. make use of an Enterprise Service Bus (ESB) to automatically extract EA-related information for documentation purposes [15]. This data is compared against the metamodel proposed by ArchiMate, whereby the achieved coverage is measured. While the approach succeeds in covering up to 75% of the application layer and up to 50% of the technology layer, the business layer was found to be covered by only up to 20%.

- Holm et al. utilize network scanning for automatic data collection [8]. The gathered data is mapped to ArchiMate models but is not able to bridge the gap to the business layer, thereby only providing information on the application and technology layer. Furthermore, the extracted data is reportedly too detailed for EA purposes and lacks information on communication relationships.

- Similar to the previous approach, Alegria et al. also monitor and analyze network traffic in order to infer EA relevant information [16]. The apporach suffers from the same shortcomings as [8].

- Cuadrado et al. combine static information extracted by analyzing source code directly and dynamic information extracted by a profiling tool from runtime data to reconstruct the architecture [17]. While this approach is similar to Microlyze in combining static with dynamic information, it lacks the ability to identify communication relationships and is also limited to Java code/software. Furthermore, it is not possible to export the extracted data to a commonly used data format.

- Van Hoorn et al. developed a framework called "Kieker" that utilizes runtime data from distributed tracing for further analysis of monitored applications [18] [19]. While the extracted traces can be used to reconstruct the IT landscape, the focus is primarily on improving the continuous monitoring approach and enhancing the analyses based upon the monitoring records.

- Valja et al. successfully derive EA models by combining different data sources including runtime data extracted from a network scanner and transforming the different sources to a common language but also lack the ability to extract communication relationships. [20] [21].

- Farwick et al. proposed an automated approach to integrate information about running infrastructure instances in the cloud into an EAM view. While making use of runtime data, it lacks the measures in place to historize data or to identify communication relationships [1].

- Fittkau et al. use monitoring data to improve consistency between EA models and reality. The extracted information is used to visualize relationships between applications but is not stored for historization purposes. The approach is also unable to bridge the gap between the application and the business layer [22].

2.4. Delimitation

This thesis uses *Microlyze* as conceptual foundation and implements the proposed solution approach in a real-world environment situated in the automotive industry. It represents a proof of concept through instantiation. While the concept of microservices is universal and does not differ much depending on the environment it is applied to, the developed IT artifact being implemented and evaluated in the automotive industry might impact the generalizability of the findings.

Due to restrictions imposed by our industry partner, it is not possible to implement the JSON configuration files within a CI/CD pipeline. This part is simulated through a manually maintained document that contains static information from federated information systems similar to the information that would otherwise be contained in said configuration files in order to retain the full capability of the proposed solution approach in reconstructing the as-is landscape and bridging the gap between application/technology and business layer. In this way, it is possible to "paint the big picture" and thus, it is possible to evaluate the entire solution approach.

The implementation will primarily focus on the extraction and usage of runtime data generated through distributed tracing. Special attention will thereby be paid to identifying and extracting communication relationships between microservices, which is scarcely covered by other mentioned solution approaches.

3. Implementation

The previous chapter focused on the concept and theoretical background of the proposed solution approach including the introduction to *Microlyze*. This chapter will cover the actual implementation in more detail.

Please note:
To keep this thesis as easily readable as possible, actual code will only be shown in an exemplary manner wherever the author deems it necessary. Otherwise, code fragments and implementation details will be generalized and explained in text. Should one be interested in the actually implemented code as a whole, it can be found attached to the thesis as digital supplement.

3.1. Proposed Solution - Overview

The implemented IT artifact is mainly written in TypeScript[1], a typed superset of JavaScript which compiles to plain JavaScript. The provided functionality can be roughly divided into two parts: data extraction and data visualization.

The data extraction part consists of automatically connecting to a supported Application Performance Monitoring (APM) tool, extracting relevant EA-related information, mapping it to our data model and storing it in a database.

The data visualization part builds upon the extracted data set and provides a variety of visualizations directed at different EA stakeholders.

Figure 3.1[2345678910] provides a high-level overview of the implementation of the proposed solution including the most important technologies in use and serves as a reference for the following sections of this chapter in which the individual parts will be presented and explained in more detail.

[1] typescriptlang.org
[2] dynatrace.com
[3] graphql.org
[4] nodejs.org
[5] mongodb.com
[6] developers.google.com/web/tools/puppeteer
[7] google.com/chrome
[8] reactjs.org
[9] yworks.com/products/yfiles-for-html
[10] products.office.com/excel

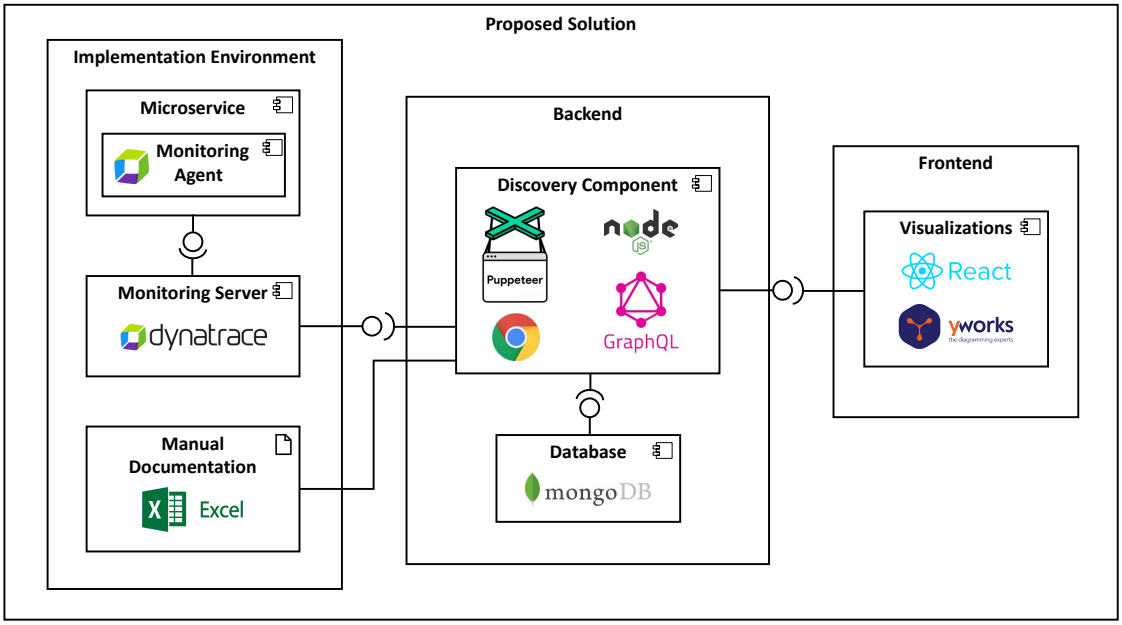


Figure 3.1.: Proposed Solution - Overview

3.2. Implementation Environment

As mentioned before, this thesis was conducted in cooperation with a large German enterprise in the automotive industry. To offer their products and services, this enterprise contains a division responsible for all IT-related concerns. This thesis was conducted at the department for vehicle data connectivity, which is mainly responsible for a variety of IT products and applications necessary to provide the backend for multiple services in the context of connected cars.

3.2.1. Requirements Analysis

In order to identify what requirements the proposed solution approach should fulfill (i.e. which IT artifacts should be documented in an automated fashion), the advisors at our industry partner were questioned by means of a questionnaire in which they should rate various IT artifacts along several criteria, which are presented hereinafter.

- **Relevance**

  How relevant is the documentation of the IT artifact?

  This question aims at determining how important it is to document a given artifact. IT artifacts that are not perceived as very relevant also do not necessarily need to be documented automatically.

- **Completeness**

  How complete is the current documentation of the IT artifact?

  This question aims at identifying how complete the already existing documentation of a given artifact is. Already completely documented IT artifacts have a lower need for automated documentation than incompletely documented ones.

- **Rate of Change**

  How often does the IT artifact change?

  This question aims at determining how often an artifact is subject to change and therefore how often its documentation needs to be updated accordingly. More frequently changing IT artifacts have a higher need for automated documentation since the documentation also needs to be updated more frequently whereas less frequently changing IT artifacts might as well be documented manually since the effort to keep the documentation up-to-date is not as high.

- **Actuality**

  How current (up-to-date) is the documentation of the IT artifact?

  This question aims at identifying the actuality of the documentation of a given artifact. An already up-to-date documentation would signify that the current documentation process is sufficient and the need for automation is low.

- **Degree of Automation**

  How high is the degree of automation regarding the current documentation of the IT artifact?

  This question aims at identifying to what extent the documentation of a given artifact is already automated. Should there already be a functioning automated documentation process in place, the need for automation would obviously not be as high as for IT artifacts without automated documentation.

The underlying idea of the questionnaire is to identify which IT artifacts would benefit most from being documented automatically and should therefore be prioritized first during the prototypical implementation. The higher a given IT artifact scored within a category the higher the need for automated documentation is perceived to be. The overall score of a specific IT artifact was determined by summarizing the scores for each category and each participant and then dividing it by the number of participants who answered the questionnaire. Table 3.1 shows the top ten requirements that have the highest need for automated documentation.

Rank	Requirement	Score
1	Data flow and dependencies between applications	18
2	Interfaces / APIs	18
3	Mapping and associations within application layer	18
4	Application Components (logical unit)	17
5	Communication technology (protocols)	17
6	Business Processes	15
7	Mapping and associations within technology layer	15
8	Physical IT resources	14
9	Mapping and associations within business layer	13
10	Use Cases / Scenarios	13

Table 3.1.: Requirements Analysis

The top three identified requirements share the same score and mutually pertain to the application layer. They also collectively share the similarity that they concern themselves with some aspect of interrelationships between microservices, being it the actual data flow and dependencies, the targeted interfaces or the mapping between components in general. Due to their importance, these requirements will be addressed with the highest priority.

3.3. Monitoring - Dynatrace AppMon

The IT applications of our industry partner mostly consist of microservices, which are monitored by Dynatrace Application Monitoring (AppMon). AppMon is an Application Performance Monitoring tool offered by Dynatrace. The version in use during the conduction of this thesis was *AppMon 2018 April*.

3.3.1. AppMon Overview

A typical AppMon deployment in a live environment includes the following core components:[11]

- **AppMon Server**

  The AppMon Server is a backend process and functions as central administration unit. It is responsible for correlating data received from Agents through the AppMon Collector and storing it in the file system together with memory dumps.

- **Frontend Server**

  The Frontend Server handles requests received from clients and needs to run on the same machine as the AppMon Server.

- **Memory Analysis Server**

  The Memory Analysis Server is responsible for processing memory snapshots that would otherwise put too much load on the Frontend Server. It can run on a different machine than the previous two components.

- **Collector**

  AppMon Collectors receive and combine data from their associated Agents and send it to the AppMon server. They need to be close to their Agents in the network but do not need to be close to the server.

- **Performance Warehouse database**

  The AppMon Performance Warehouse is a SQL database server responsible for time-series data gathered by the Agents. This data is stored separately from the rest that is stored in the file system.

- **Client**

  The AppMon Client is a frontend component and serves as user interface to access the relevant information collected and stored by the backend services. The client can either be installed on a user's machine or accessed directly through a web interface by a browser.

[11] dynatrace.com/support/doc/appmon/getting-started/architecture/

- **Application Agents**

  An Agent is a software that is either running on the host or injected into the application process. It gathers performance data and sends them to the AppMon server.[12] Agent Groups are logical groupings of multiple Agents. In the context of our implementation environment, one Agent Group roughly represents one microservice.

In a nutshell: The most important aspects of the monitoring tool concerning our solution approach can be summarized as follows. Agents are injected into application processes thereby collecting information and sending them to collectors. The AppMon server collects all sorts of relevant information about the monitored microservices through the collectors and stores them in its database that can be queried by the users.

3.3.2. Metamodel & Data Structures

To the best of the author's knowledge, no officially documented AppMon metamodel could be found, therefore the author created one as perceived by himself after using the tool for a prolonged time, hence there is no claim to completeness and correctness.

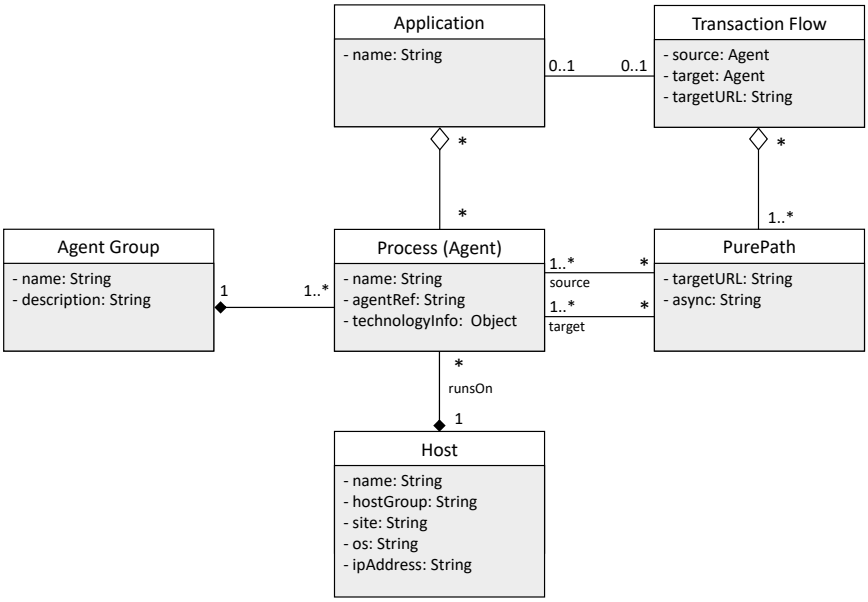


Figure 3.2.: AppMon Metamodel

As seen in Figure 3.2, AppMon's metamodel is kept rather simple. AppMon applications are logical groupings of an arbitrary amount of Agents whereas Agents and processes are used interchangeably in this context due to the Agent being injected into the process. Every Agent

[12] dynatrace.com/support/doc/appmon/appmon-reference/glossary/agent/

is part of exactly one Agent Group and runs on exactly one host whereby hosts are physical or virtual machines that belong to a host group and are assigned to a site. Hosts can also represent containers or run in the cloud. Additionally, Agents communicate with each other, which is captured by AppMon as PurePaths where they appear as source or target of the communication. An arbitrary amount of PurePaths is represented as Transaction flow that can be assigned to an AppMon application.

For the purposes of our solution approach, Transaction flows and PurePaths were identified as relevant data structures provided by AppMon and shall be presented hereafter in more detail.

- **Transaction flow**

  Transaction flows contain information on monitored Agents for a specific timeframe including information about the assigned hosts and calls between Agents. The information about communication between Agents is rather limited and does not include which interfaces were targeted. Furthermore, if communication took place asynchronously by means of a messaging queue, this message queue is abstracted from the Transaction flow, i.e. if Agent A communicated with Agent C over message queue B, the Transaction flow will show this as "A communicated with C" without mentioning the message queue. Consequently, all communication appears as synchronous (direct) communication in Transaction flows. In terms of data structures, Transaction flows are realized as arrays of JSON objects, where each entry represents one Agent with all its metadata.

- **PurePath**

  PurePaths are traces of requests as perceived by the monitored system whereas each PurePath represents one unique request and its way through the system. Multiple Agents can participate in a single PurePath. Unlike Transaction flows, PurePaths do capture targeted interfaces and include asynchronous communication over message queues. In terms of data structures, they are realized as data trees with arbitrary width and depth whereby each node represents an Agent and each edge represents a call between the connecting nodes (Agents).

3.3.3. Limitations & Workarounds

Unfortunately, AppMon exhibits a few limitations that also restrict the effectiveness of the proposed solution approach. These limitations and the applied workarounds will be presented and discussed hereinafter.

- **Lack of "useful" APIs**

  AppMon provides a variety of different REST APIs which are documented in the official AppMon documentation[13]. However, none of the openly available REST APIs for the AppMon version in use are useful in the context of our purposes.

[13] dynatrace.com/support/doc/appmon/appmon-reference/rest-interfaces/server-rest-interfaces/

AppMon provides its services in two ways: a web interface accessed through any common browser application and a dedicated JAVA client. To retrieve the requested information from AppMon through the web interface, the browser uses a REST API that is not openly available and therefore not officially documented. This REST API, however, does provide information considered useful for our purposes. In order to fully exploit these undocumented APIs, several useful URLs including their necessary parameters were identified by using Google Chrome to navigate the web interface and observe the network traffic between the browser and the AppMon server through Chrome's built-in developer tools.

This approach creates a new obstacle. Since these undocumented APIs are not designed to be freely used by developers, they require specific security credentials that need to be included in every HTTP request header in order for the AppMon server to reply with the expected response. Otherwise, it replies with a *HTTP 401 Unauthorized* response. In a trial and error approach, two parameters were identified as absolutely necessary to successfully retrieve the required information.

First, a so-called *WEBUIJSESSIONID* needs to be included in the cookie of each request. This session ID is sufficient to successfully perform HTTP-GET-Requests and retrieve Transaction flows.

Second, a CSRF-Token needs to be added as a *X-XSRF-Header* in each request. These tokens serve as security mechanism designed to prevent CSRF attacks. (Note: Explaining this type of attacks in further detail is not within the scope of this thesis.) This header, in addition to the session ID, is necessary to successfully perform HTTP-POST-Requests and retrieve PurePaths.

Note: While a HTTP-POST-Request is required to retrieve PurePaths, the data on the server side is not altered in any way. The HTTP-POST-Request merely serves the purpose of including a filter which is applied to the PurePaths to filter them by category. None of the available filters are useful for our purposes and therefore only an empty array ("categories":[]) as filter is included in our HTTP-POST-Requests.

- **Automation**

  One aspect of the proposed solution approach is to extract relevant information in an *automated* fashion without requiring further manual user input. As described in the previous section, there are no openly available APIs that can be used and the undocumented APIs, which serve as an alternative for our approach, require specific security credentials that can only be retrieved from the server after manually logging in.

  In order to circumvent this issue and automate the process of logging in, our approach makes use of Puppeteer[14], a JS library that provides an API which enables us to control a headless instance of Google Chrome. This way, our solution can automatically navigate to the URL where the deployed AppMon server instance resides and submit the login form with valid login credentials provided within the source code. The

[14] developers.google.com/web/tools/puppeteer

received response from the server is used to extract the necessary *WEBUIJSESSIONID* and *X-XSRF-Header* from the included cookies. Afterwards, Puppeteer is no longer required and can be closed. All further requests to the AppMon server can now be performed through the standard JS fetch API.

- **Lack of applicable filters**

  Exploiting the undocumented APIs as in the approach described above is further limited by the lack of applicable filters.

  Regarding Transaction flows, there are two options. One can either choose to request all Transaction flows of the entire monitored system for a specific timeframe (i.e. no filter) or filter them by AppMon application. Since filtered data can always be combined to match the unfiltered data, the latter option was chosen to request Transaction flows.

  PurePaths, on the other hand, can either be requested unfiltered or be filtered either by AppMon application or by Agent. While the last option provided the most thorough results, it also took a heavy toll on the discovery algorithm's runtime. To put a figure on it, it took a little bit over 24 hours to process the data for a timeframe of six hours, which is beyond feasible. Further test runs using AppMon applications as filter yielded an acceptable tradeoff between retrieved amount of data and overall runtime, hence this filter was chosen for the final run.

- **Naming convention**

  For the purposes of our solution approach, information on individual Agents is not regarded as relevant as information on the Agent Groups and therefore all applicable data is aggregated on Agent Group level.

  When looking at Transaction flows, Agents include a reference to their respective Agent Group. PurePaths, however, only contain references to the participating Agents. Since Transaction flows and PurePaths are requested separately by the discovery algorithm and information on individual Agents is only stored in an aggregated form on Agent Group level, it is not possible to relate received PurePaths to the according Agent Groups. AppMon does not provide a distinct naming convention to match Agents to Agent Groups.

  To solve this issue, every time an Agent is discovered through a Transaction flow, an entry containing the Agent and its corresponding Agent Group is added to a list which serves as dictionary when processing PurePaths to look up Agents and match them to their respective Agent Group. For the sake of simplicity, this dictionary is also added to our database even though it serves no further purpose in the grander scheme of our solution approach.

- **Timeframe restrictions**

  Timeframes serve as filter for the monitoring data and can either be defined relative to the current point in time or between two specific points in time. AppMon provides a set of predefined timeframes (e.g. *last 24 hours*), but also supports the functionality

to define custom timeframes in a very finegrained manner. Specific points in time are referenced through Unix timestamps in milliseconds. While custom timeframes are very helpful in the context of our solution approach, it was observed during the execution of the discovery algorithm that the further back in time data was requested, the longer AppMon needed to respond to the request. Additionally, PurePaths could only be retrieved for roughly the last ten days relative to the point in time the discovery algorithm was initiated. From that point on backwards, only Transaction flows could be retrieved.

- **Completeness of data**

  When requesting Transaction flows for a specific timeframe, AppMon provides a complete data set for that specific timeframe. When requesting PurePaths, however, the result is limited to the 100 most recent PurePaths. As a result, depending on how busy certain microservices are, requesting PurePaths for a timeframe of, e.g. one hour, might effectively retrieve data pertaining only to the last few minutes of that timeframe.

  This could be countered by minimizing the requested timeframe, but in turn, the overall runtime of the algorithm would increase. At some point, the duration of the algorithm exceeds the requested timeframe thereby rendering the solution approach practically infeasible. In various tests, setting the requested timeframe to six hours proved to provide an acceptable tradeoff between accuracy of individual timeframes and overall runtime of the discovery algorithm. Ideally, since there exists a finite set of possible PurePaths to be observed, running the discovery algorithm for a sufficiently long period of time would eventually lead to discovering each unique PurePath at least once. Further research is required to test this hypothesis and further optimize the span of the timeframes.

- **Parameters in requests**

  This limitation ties in with the last one and again only affects PurePaths. AppMon does not recognize which parts of observed URLs within communications between Agents are fixed and which parts are variable, hence, every PurePath is identified as unique even if multiple PurePaths represent the same requests where technically the same interfaces were called but with different parameters. This creates two issues.

  First, a lot of redundancy is introduced into the system. To give a specific example, when music is requested from a large database, each requested track is identified as unique request since each track has its own unique ID associated with it, while from a logical perspective all these requests represent the same type of request.

  Second, since parameters are not identified as such, they are also not masked. Consequently, possibly data protection sensitive information is shown in plaintext thereby infringing prevailing data protection law.

  In order to tackle both of these issues, a multitude of regular expressions were created for the context of the implementation environment (see section 3.2) to identify all sorts of redundancy and sensitive information and mask them with asterisks. Due to the

diverse nature of possible parameters, no guarantee can be made that all redundancy and especially all sensitive information could be found and masked accordingly. This, however, is primarily a fault on the side of the monitoring tool and beside the mentioned regular expressions no further measures will be taken to tackle these issues within the scope of this prototypical implementation.

- **No distinction regarding origin of requests**

  AppMon lacks the capability to distinguish the origin of requests. Therefore it is not possible to differentiate between requests that were sent to the system from, e.g. a web browser or a mobile application.

- **Overly strained AppMon instance**

  This is technically a pseudo limitation since it is not inherently a limitation of AppMon but shall still be mentioned for the sake of completeness. The deployed instance of AppMon in the implementation environment is reportedly running close to full capacity, therefore our industry partner asked us not to put too much workload on the server. For this reason, all requests directed at the server were performed in a sequential manner, which evidently increased the overall runtime of the discovery algorithm. As a positive side effect, requesting information only sequentially reduced the complexity of synchronizing access to the database when storing relevant information.

3.4. Data Model

This section will cover the data models as used in the solution approach and explain how information is mapped from the monitoring tool to our data model. In order to generalize the proposed solution approach and abstract the data model from specifics used in a particular monitoring tool, a generalized data model was used that allows extending the solution to support a multitude of different APM tools, each with their own unique data model in the future. To this effect, the ArchiMate notation is used as introduced in subsection 2.1.2.

3.4.1. Logical Data Model

The logical data model as seen in Figure 3.3 is used to describe the IT architecture elements. It is based on the ArchiMate framework described in 2.1.2. The individual entities of the data model will be explained in more detail hereinafter.

- **Business Layer**
  - **Domain**

    Domains are the elements placed highest in the hierarchy and can contain an arbitrary amount of subdomains. Subdomains, however, can only belong to one overlying domain. Technically, domains do not exist in the ArchiMate notation and were therefore adapted from business functions.
  - **Product**

    Products are nested within the domain they belong to and in turn can contain an arbitrary amount of subproducts. Subproducts, however, can only belong to one overlying product.
  - **Business Service**

    Business services belong to exactly one (sub)product and represent business applications that expose a certain business functionality. They use and depend on microservices to deliver their capabilities. In that regard, they are modeled as logical groupings of application components (microservices).
- **Application Layer**
  - **Application Component**

    Application components are the central unit of the application layer within this model. They represent microservices and belong to at most one business service.
  - **Application Service**

    Application services represent the functionality of application components (microservices) which is exposed through their interfaces. Communication takes place between application components by calling their respective application services with a well-defined URL. Each Application Service is assigned to exactly one

application component and an arbitrary amount of application services (but at least two) can be part of an application interaction.

  - **Application Interaction**

    Application interactions represent requests that are directed at the system and are processed by a set of application components represented as application collaborations. They may realize an arbitrary amount of application services.

  - **Application Collaboration**

    Application collaborations represent logical groupings of application components that jointly perform some collective task which is represented as application interaction. Therefore an arbitrary amount of application components (but at least two) can be part of an application collaboration which collectively fulfill an arbitrary amount of application interactions.

- **Technology Layer**
  - **Node**

    Nodes represent hosts on which the application components run. They are the technical infrastructure that is necessary to operate the application components and provide their functionality. An arbitrary amount of application components can be assigned to one node. Nodes can either be physical or virtual and can be situated on-premises or in the cloud. Containerization is also supported.

  - **Facility**

    Facilities represent the data centers in which nodes are situated. They form the physical and logical grouping of hosts whereby an arbitrary amount of nodes can be assigned to one facility. Facilities are placed lowest in the hierarchy of the data model and can be considered the technical foundation for the entire IT landscape.

Note: The logical data model also contains an entity called "Device". Originally, this was intended to be used for modeling client devices such as browsers or mobile devices that issue the requests to the monitored systems by calling the application services of application components. Since AppMon does not provide this kind of information, it is not used in the prototypical implementation and is therefore greyed out in the figure. It was chosen to be left as stub in the data model for possible future extensions supporting other APM tools that do provide this information.

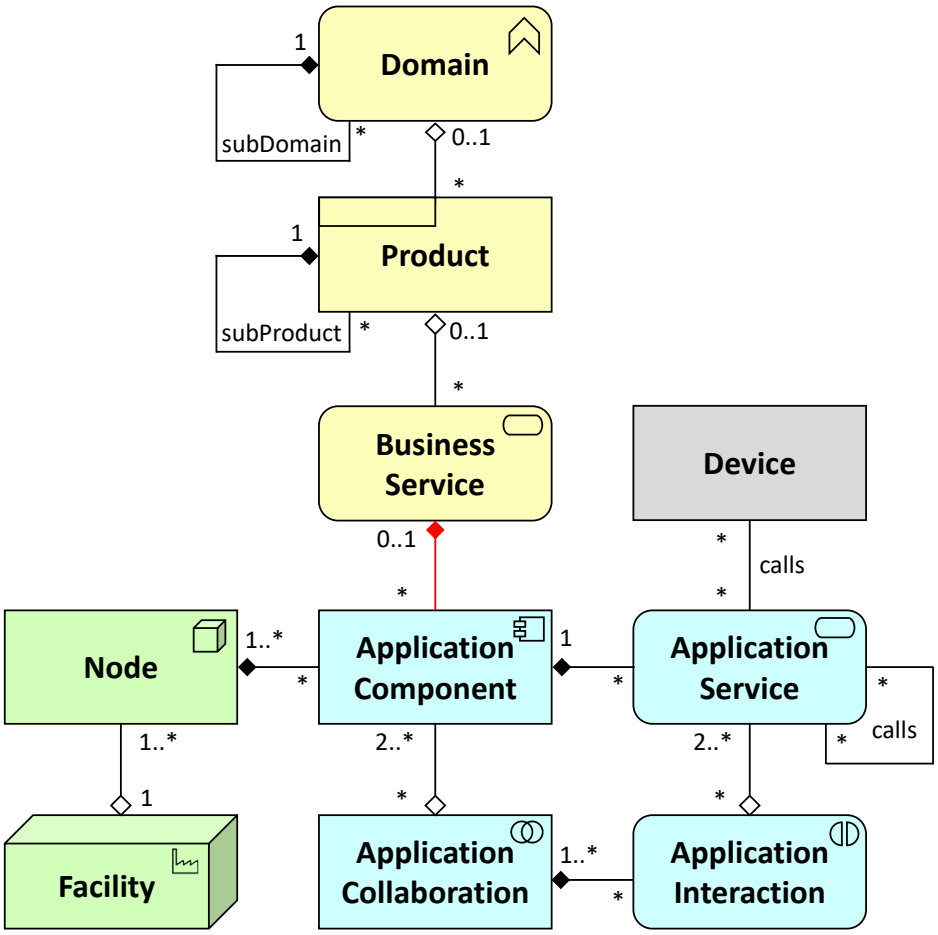


Figure 3.3.: Logical Data Model

3.4.2. Mapping

Figure 3.4 shows how runtime data from the APM tool, in this case AppMon, is mapped to the logical data model presented in the previous section.

Application Layer

The most important elements are the Agent Groups which represent our microservices. They are modeled as application components. This information is extracted from the Transaction flows where individual Agents are aggregated to Agent Groups and the total amount of Agents is stored as attribute of the respective application component.

Application services are not directly monitored as separate entities by AppMon, which is different from other APM tools that do keep track of individual application services. To keep the data model as general as possible and allow for conformance with other APM tools in the future, every time an Agent Group is discovered, two application services are created automatically. One application service represents an interface for incoming requests and the other for outgoing requests. They strictly follow a naming convention where "_IN" or "_OUT" is added to the ID of the respective application component to signify the incoming and outgoing interface. The actually targeted URLs are modeled as attributes of the incoming and outgoing communication calls.

Application collaborations represent the AppMon applications. These are logical groupings of Agent Groups as perceived by AppMon. Unfortunately, these AppMon applications do not necessarily always match an actual application as defined and referred to as by our industry partner. This data can be queried directly from the monitoring server and extracted from the associated response. Application collaborations are also used as filter when querying Transaction flows and PurePaths.

Application interactions directly represent PurePaths. Every unique root PurePath is mapped to an application interaction and assigned to its respective application collaboration. Since PurePaths are composed of an arbitrary amount of communication calls (requests) between application components, each individual request is stored separately as relationship between two application services whereby the targeted URL is assigned as attribute to the relationship, all while keeping track of the association between the individual requests and the root PurePath. This way, it is possible to identify and show individual communication between application components (through their application services), but also shift the focus to individual root requests and their path through the system, thereby enabling multiple perspectives on communication relationships within our model.

Technology Layer

AppMon hosts are represented as nodes. Associated information like operating system and IP address, among others, are stored as individual attributes of the node. This information is extracted from the Transaction flows, which also contain the hierarchical relationships between hosts and Agent Groups. Every time a new host is discovered, a separate query is sent to the server to request further information about the host to gain insight into whether

the host is a physical or a virtual machine and whether the host runs on-premises or in the cloud, among other things.

AppMon hosts are also grouped into host groups and assigned to sites within the data model of AppMon. Originally, the intention was to map either host groups or sites to facilities within our data model. AppMon, however, reported the site and host group of every discovered host to be either "local" or "default". Since neither of these pieces of information is particularly useful, the name of the hub to which all monitored applications and hosts belong was used as facility. This information was extracted from AppMon's system profile. Since access to the APM tool was limited to one hub during the conduction of this thesis, there is also only one facility that could be "discovered". It is not clear whether the trivial information ("local" and "default") are a configuration issue on the side of our industry partner or a discovery issue on the side of AppMon, therefore it is not listed as limitation of AppMon in subsection 3.3.3. In any case, this does not limit the proposed solution approach in general, since some information that can be used to describe and represent facilites could automatically be extracted.

Business Layer

The entities of our data model pertaining to the business layer are not directly monitored within the AppMon environment. This data would normally be extracted from the JSON configuration files. Since these could not be implemented as part of the proposed solution approach as explained in section 2.4, a workaround is used to gap the bridge between application and business layer and provide a complete picture of the IT landscape.

Every time an Agent Group (application component) is discovered, its ID is normalized (spaces, special characters, etc. are removed) and compared with the manual documentation of our industry partner. The manual documentation is a collection of various information from different sources (CMDB and other federated systems) and was provided by our industry partner in the form of a Microsoft Excel sheet. This sheet contains a column with microservice IDs, which are also normalized in the same way as the IDs of the discovered microservices. The normalized IDs are then compared and if a match is found, various information from the associated columns are mapped to the business layer entities within our data model including domains, (sub)products and business services and their corresponding associations within the business layer but also to the application layer. Additionally, relevant attributes (e.g. product owner) are added to their respective entities.

Note that business services in this context represent applications as defined by our industry partner. Ideally, these would match the application collaborations discovered by AppMon but this is not always the case as mentioned before.

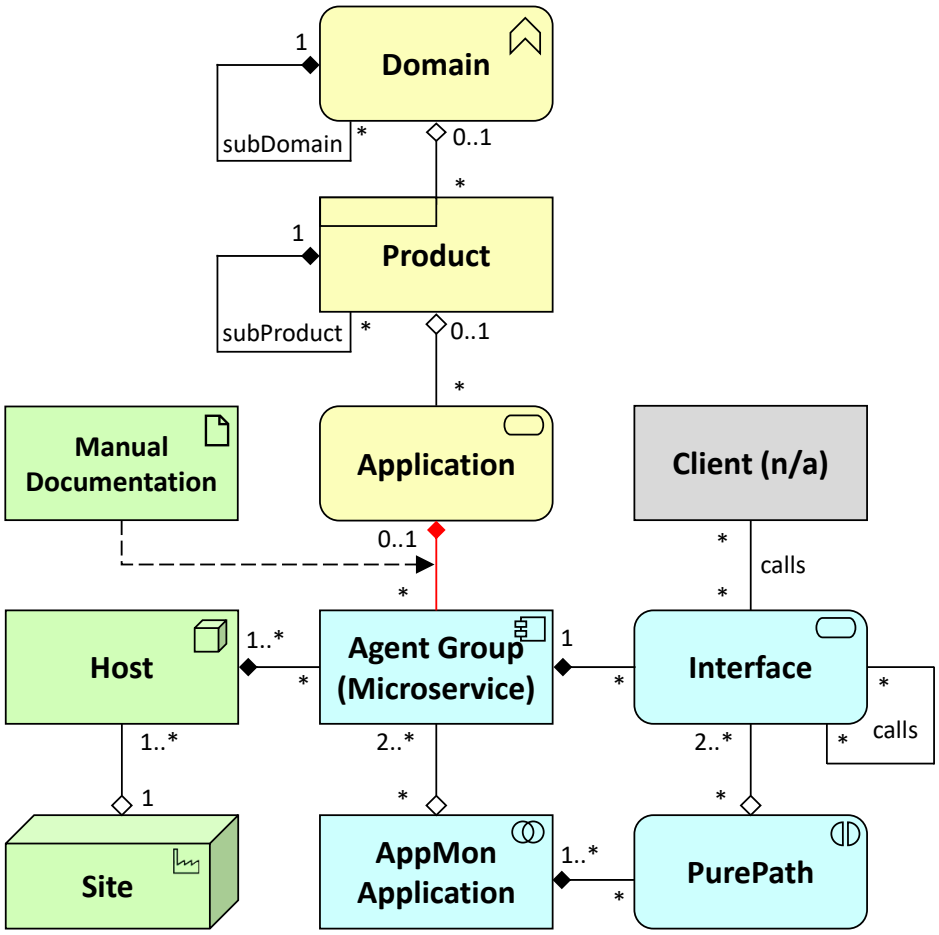


Figure 3.4.: Mapping of Data Model

3.4.3. Physical Data Model

The logical data model presented in subsection 3.4.1 is realized within the database as depicted in Figure 3.5.

Since the logical data model tends to change quite frequently during prototyping, which was also the case here, a simple yet flexible physical data model was chosen for the prototypical implementation. This way, changes in the logical data model did not necessarily always result in having to drop the database and start from scratch to accommodate for the change.

The physical data model consists of three tables, or in this case, collections, which shall be explained in further detail hereinafter.

- **ArchitectureModel**

  Every entity from the logical data model is stored as architecture model in the database whereby all models possess the same attributes and are only distinguished by their type. The type attribute is modeled as enumeration where all distinct types from the logical model are available. It follows that each object is assigned the same type from the corresponding logical entity with the exception that subdomains and subproducts are treated as domains and products respectively.

  In addition to the unique ObjectId ("_id") automatically assigned by MongoDB (see subsection 3.5.2), each object is also associated with its ID and name extracted from AppMon or the manual documentation.

  The attributes "validFrom" and "validTo" denote the timespan in which a particular architecture model is considered to be active and therefore relevant. The former is initialized with the timestamp of the element's first discovery and the latter with positive infinity, a special number that will always be bigger when compared to any valid timestamp. This is important for queries that request architecture models that were valid at a certain point in time t_q. These queries will receive all architecture models as response for which the following condition applies: "validFrom" $\leq t_q \leq$ "validTo".

  Additionally, the attributes "validFrom" and "validTo" are necessary for the historization of elements. Every time an architecture model changes, "validTo" is updated with the timestamp of that change and a new data object is inserted into the database that is assigned a new "_id" but otherwise carries over all attributes from the previous version whereby "validFrom" is set to the value of "validTo" of the previous version and "validTo" is initialized again with positive infinity.

  The attribute "lastSeen" simply denotes the point in time in which the particular element was last detected within the runtime data. The following condition needs to always hold true: "validFrom" $\leq$ "lastSeen" $\leq$ "validTo".

  An architecture model can have an arbitrary amount of relationships but needs to have at least one. This ensures that there is no element in the database that is not connected to at least one other element in some way.

- **Relationship**

  Every edge from the logical data model is stored as relationship in the database. Three different types of relationships are distinguished: hierarchy, grouping and communication. These represent the compositions, aggregations and associations of the logical data model respectively. The remaining attributes are the same as for the architecture models and are used for the same reasons and purposes.

  A relationship always establishes a link between exactly two architecture models which are referenced by ID through "source" and "target". The attribute "owner" additionally allows to group multiple relationships which logically belong together. This is used for application interactions (PurePaths) that contain multiple individual communication relationships. While these communication relationships connect two application services with each other, they are also part of the overlying root request and need to be designated as such.

- **Annotation**

  Annotations basically represent key-value pairs. They are also assigned a unique ObjectId from the database, but additionally possess a reference to the ObjectId of either an architecture model or a relationship, which serves as foreign key. This way, each version of an architecture model or relationship can have its own set of distinct annotations. An annotation is always associated with exactly one architecture model or relationship whereas architecture models and relationships can have an arbitrary amount of annotations.

  Annotations are used to store any type of attribute that the associated architecture model or relationship might have. Since it was not clear during the prototyping phase which attributes extracted from the runtime data are relevant and should therefore be stored in the database, using annotations as an universal means to represent any possible attribute, allowed the physical data model to remain flexible and allow for adjustments later on without having to constantly update the data model to add or remove fixed attributes.

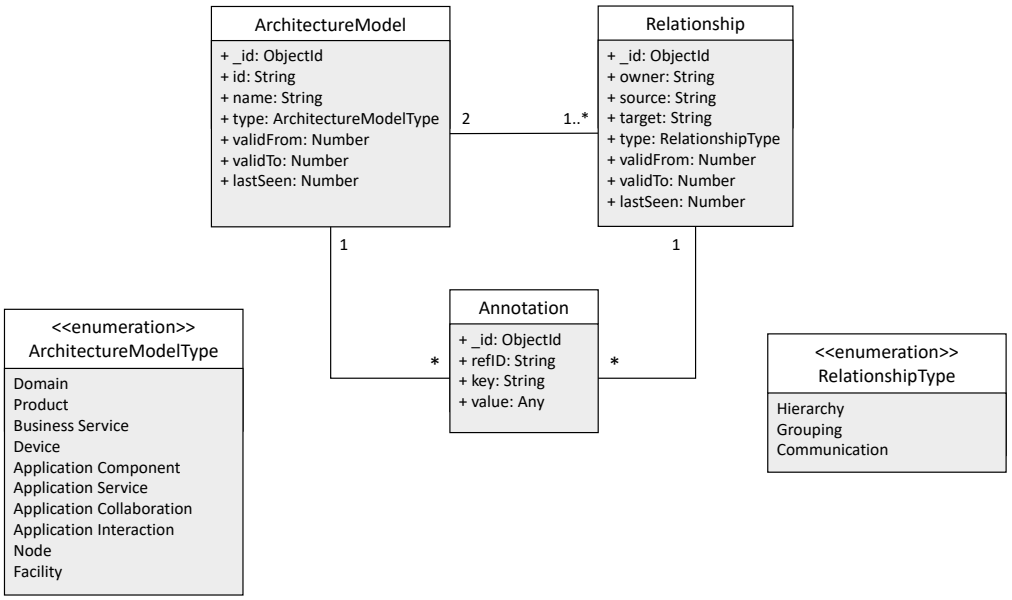


Figure 3.5.: Database Data Model

3.5. Backend

This section will cover the most important implementation decisions and details regarding the backend including a comprehensive explanation of the discovery algorithm that represents a crucial element of the proposed solution approach.

3.5.1. GraphQL

GraphQL is an open source data query language originally developed by Facebook and represents an alternative to the commonly used REST architecture.[15] It was chosen as query language for the proposed solution for the following reasons.

Advantages of GraphQL

- **No over- or under-fetching**

  REST-based interfaces typically suffer from over- and under-fetching. Over-fetching is when a request returns too much data because the addressed endpoint returned fixed data structures including data which is not needed. Under-fetching, on the other hand, is when an addressed endpoint returns not enough data requiring the client to send one or more additional requests to other endpoints to acquire the desired results.

  GraphQL solves this problem by exposing all of the data from a single endpoint, thereby enabling the client to request precisely the data that is required with a single query. This not only reduces the complexity of creating queries, but also minimizes the amount of data transferred.

- **Strong typing**

  Due to its strongly-typed type system, GraphQL is able to check queries for syntactic correctness and validity before execution, thereby allowing the server to make certain guarantees about what response to expect.

- **Documentation**

  With the aid of certain tooling, GraphQL is able to automatically generate a basic documentation derived from the defined schema, which reduces the need for manual documentation, and is therefore considered self-documenting.

The logical data model as presented in 3.4.1 was mapped directly one-to-one to a GraphQL schema.

For the purposes of the proposed solution, two GraphQL resolvers were implemented. Resolvers are functions that resolve values for corresponding queries. Incoming queries are parsed, validated against the schema and executed. The result is a JSON Object where all requested values have been resolved.

[15] graphql.org

In this regard, queries are used for data fetching and are therefore read-only, they do not change the underlying data set. To modify the data set, so-called mutations should be used, which are basically a special type of query.

- **Database Resolver**

  The Database Resolver serves as central endpoint for all queries aiming at fetching data from the database. It contains no mutations and therefore it is not possible to alter the data set in any way. The main purpose of this resolver is to provide access to the database so that the frontend is able to extract the information necessary for its various visualizations, which will be addressed in more detail in section 3.6.

- **AppMon Resolver**

  The AppMon Resolver represents the central endpoint for all AppMon-related matters. It provides real-time data fetching from an available AppMon server and contains the logic for the automated architecture discovery algorithm, which is implemented as GraphQL mutation since it modifies the data set. The latter part will be explained in more detail in subsection 3.5.3.

3.5.2. Database - MongoDB

MongoDB is a scalable and flexible document database which stores data in JSON-like documents and can be used free of charge.[16] Since data models tend to change a lot during prototypical implementations, a document-oriented database was chosen because of its flexibility to allow the developers to accommodate changes as they occur.

Note: While the built prototype does use a document-oriented database for the above reasons, this is not a requirement for the proposed solution to function correctly. The data model as presented in section 3.4 can alternatively be mapped to and used with a relational database.

[16] mongodb.com/what-is-mongodb

3.5.3. Automated Architecture Discovery Algorithm

The implemented Automated Architecture Discovery Algorithm (AADA) is based on the backwards and forwards discovery algorithms presented in subsection 2.2.3, which were adapted to the implementation environment.

The basic procedure of AADA consists of initiating the discovery at a specific point in time, iteratively going back in time by custom timeframes up until a specified point in the past, then continuing from the initial point in time moving forward again by custom timeframes up until a specified point and then terminating.

In order to understand the discovery algorithm, a few parameters need to be introduced and explained first.

- **currentTimestamp**

  The currentTimestamp denotes an arbitrary point in time in the form of a unix timestamp in milliseconds. It is used as iterator variable to keep track of the point in time which represents the current "present" for the algorithm.

- **pastTimestamp**

  The pastTimestamp denotes an arbitrary point in time in the past in the form of a unix timestamp in milliseconds.

  The following must always apply: pastTimestamp $\leq$ currentTimestamp.

- **futureTimestamp**

  The futureTimestamp denotes an arbitrary point in time in the future in the form of a unix timestamp in milliseconds.

  The following must always apply: futureTimestamp $\geq$ currentTimestamp.

- **startTimestamp & endTimestamp**

  The startTimestamp and endTimestamp both denote an arbitrary point in time in the form of a unix timestamp in milliseconds.

  The following must always apply: startTimestamp < endTimestamp.

- **timeInterval**

  The timeInterval denotes a timespan in milliseconds. It is used to set the timeframes by which the algorithm iterates through time.

Conceptually, AADA can be divided into two parts. One part serves as controller and handles the iterative function calling while the other handles the actual discovery of architecture elements.

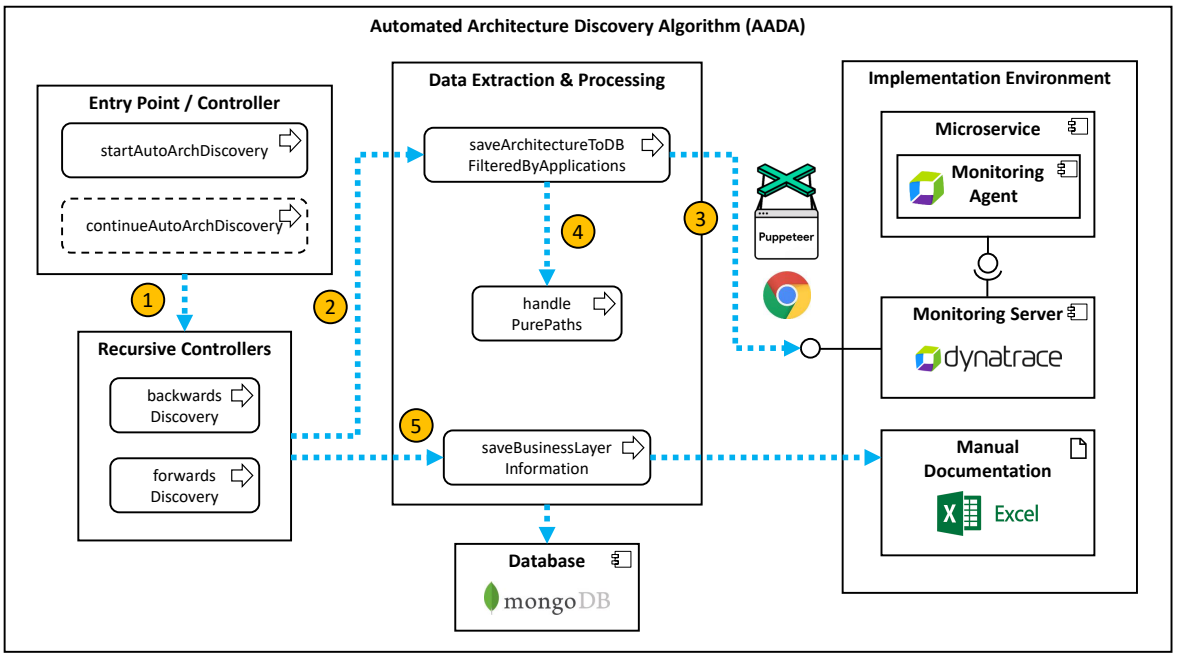


Figure 3.6.: Automated Architecture Discovery Algorithm

The actual implementation, however, follows the divide and conquer design pattern and is divided into six (seven) functions as seen in Figure 3.6[17][18][19][20][21].

- **startAutoArchDiscovery**(currentTimestamp, pastTimestamp, futureTimestamp, timeInterval)

  This function serves as entry point that can be triggered as mutation from the GraphQL backend and accepts up to four parameters to provide maximum flexibility. Since most APM solutions including AppMon offer custom timeframes for which monitoring data can be queried, AADA also allows to start the architecture discovery at an arbitrary point in time. The time interval is optional and defaults to six hours for reasons discussed earlier but can be overridden by passing the desired value to the function. The current and future timestamp are also optional and default to "now" should no other value be specified. In most cases, AADA will be configured to start at the "present time", thereby rendering the specification of a current and future timestamp unnecessary. However, additionally providing this option allows for more use cases, e.g. executing AADA at the same point in time with different time intervals to check which interval provides the best results. The only required parameter is the past timestamp so that AADA knows

[17] dynatrace.com
[18] developers.google.com/web/tools/puppeteer
[19] google.com
[20] mongodb.com
[21] products.office.com/excel

how far back in time (limited only by the capabilities of the APM solution in use) it should discover the architecture.

- **backwardsDiscovery**(currentTimestamp, pastTimestamp, timeInterval)

  This function is an implementation of the backward discovery algortihm formally described in subsection 2.2.3. It is responsible for controlling the discovery of architecture information into the past. To this effect, it takes the provided timestamps, calculates new timestamps that designate the next timeframe relevant for the discovery according to the provided timeInterval and recursively calls itself with parameters necessary to continue the traversal back through time.

- **forwardsDiscovery**(currentTimestamp, futureTimestamp, timeInterval)

  This function is an implementation of the forward discovery algortihm formally described in subsection 2.2.3. It is responsible for controlling the discovery of architecture information into the future. To this effect, it takes the provided timestamps, calculates new timestamps that designate the next timeframe relevant for the discovery according to the provided timeInterval and recursively calls itself with parameters necessary to continue the traversal forward through time.

- **saveArchitectureToDBFilteredByApplications**(startTimestamp, endTimestamp)

  This is the main function for the actual discovery of architecture elements and is provided with a start timestamp and an end timestamp from the calling controllers (forwardsDiscovery and backwardsDiscovery) which together define the timeframe that shall be queried. In its first phase it retrieves Transaction flows for all monitored applications which are processed to identify structural architecture elements like application components (web servers, databases, etc.) and their corresponding infrastructure (nodes, facilities, etc.). Once all relevant structural information has been processed for the given timeframe, the second phase starts and requests PurePaths for each application. These highly detailed traces of requests containing communication relationships between application components are handled by the function “handlePurePaths".

- **handlePurePaths**(path, appInteraction, appName, timestamp)

  This function is a recursive helper function that processes PurePaths to extract relevant communication relationships. It is provided with one PurePath at a time and since PurePaths are represented as data trees of arbitrary depth, it recursively calls itself once for every edge in the tree in order to traverse the entire PurePath. In every execution it checks whether the source (parent node) and target (child node) belong to different microservices (Agent Groups) to exclude internal communication of microservices and if so, extracts the targeted URL and whether the communication was synchronous or asynchronous. Should the child node also have child nodes, the function recursively calls itself with the previous child node as new parent and the previous child’s child node as new child node. This process is repeated for every PurePath and all its contained edges until all PurePaths have been processed in their entirety.

- **saveBusinessLayerInformation**()

  This function is called once the discovery of architecture models has been completed for a given timeframe and tries to extend the discovered elements with further static information pertaining to the business layer according to ArchiMate. This step is necessary because it was not possible to implement the JSON configuration files which would normally contain this kind of information. For this reason, a workaround was implemented consisting of looking up the relevant information within the manual documentation as explained in subsection 3.4.2.

- [QoL addition] **continueAutoArchDiscovery**(timeInterval)

  This function is technically not necessary to provide the full functionality of AADA and represents only a Quality of Life addition. Similar to *startAutoArchDiscovery*, it serves as wrapper and entry point for the underlying functionality and is intended to be used after having executed *startAutoArchDiscovery* at least once. It requires no parameters and continues the discovery from the most recent point in time where AADA has left off by querying the most recent timestamp from the database. The time interval can optionally be overridden should one wish to do so.

3.6. Frontend - Visualizations

Various vizualizations were designed and built to leverage the full potential of having EA-related information stored in a model-based approach to provide different views with a different focus for different stakeholders. They all take full advantage of the GraphQL backend and its database by issuing GraphQL queries to the corresponding resolvers thereby only requesting the information necessary for providing the respective view.

All views were built using yFiles for HTML, a JS library for diagramming and graph visualization provided by yWorks[22], which were embedded into a frontend written in React, a JS library for creating user interfaces originally developed and maintained by Facebook.[23] To generalize the displayed entities and abstract from the monitoring tool in use, the ArchiMate notation and colors were used as explained in section 2.1.2.

In the following, the different views will be presented and explained in more detail. They will later be evaluated in chapter 4.

Note: Throughout the remainder of this thesis, the terms visualization and view will be used interchangeably.

3.6.1. Business Landscape View

The Business Landscape view shows a hierarchical representation of the business layer. Application components are grouped according to their business services, which in turn are grouped by the products or subproducts they belong to. Products are nested within their respective domains. The nested groups can be closed and opened at will to show exactly what information is deemed relevant at any given time. Clicking on an element, opens a sidebar with multiple tabs that display additional information about the selected entity. This view is primarily intended to give an overview of business layer related elements directed at stakeholders who are not interested in the technical infrastructure. An example of this view can be seen in Figure 3.7.

3.6.2. Application Landscape View

The Application Landscape view shows a hierarchical representation of the application and infrastructure layer. To avoid congesting the view with too many elements, this view can be filtered by application collaborations. In this regard, this view is layouted as hierarchic tree with the selected application collaboration at the top, the associated application components in the second layer, the corresponding nodes they are running on in the third layer and the respective facilities at the bottom. Application components are grouped according to their technology and nodes according to their operating system. An example of this view can be seen in Figure 3.8.

[22] yworks.com/products/yfiles-for-html
[23] reactjs.org

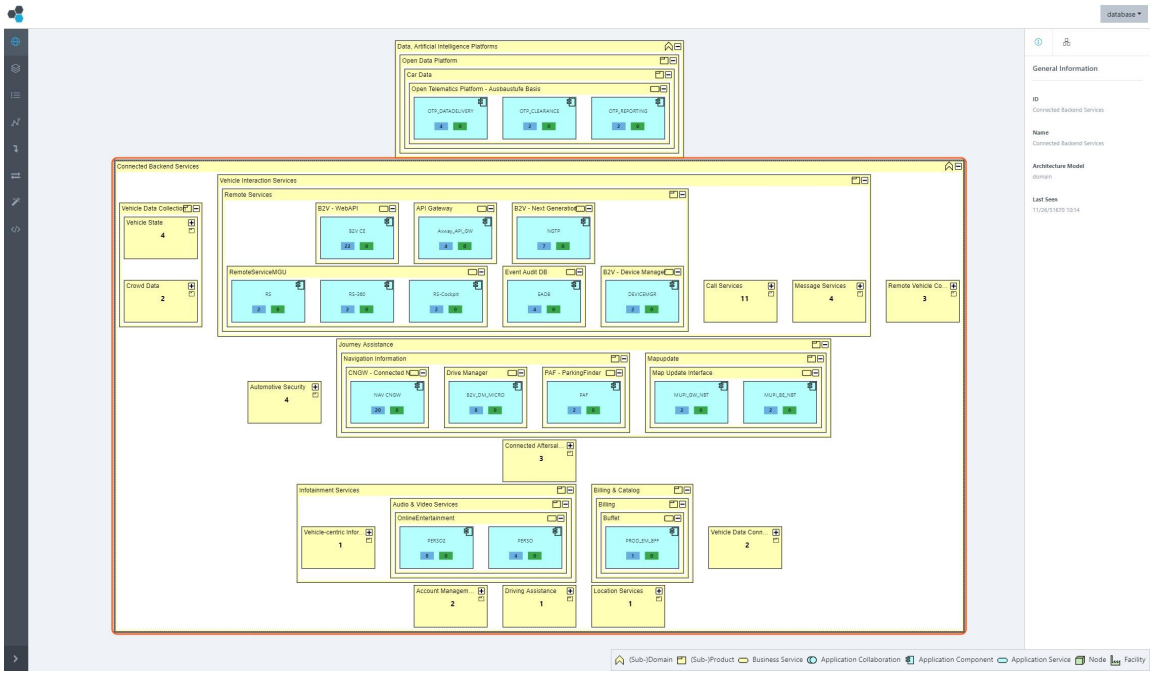


Figure 3.7.: Business Landscape View

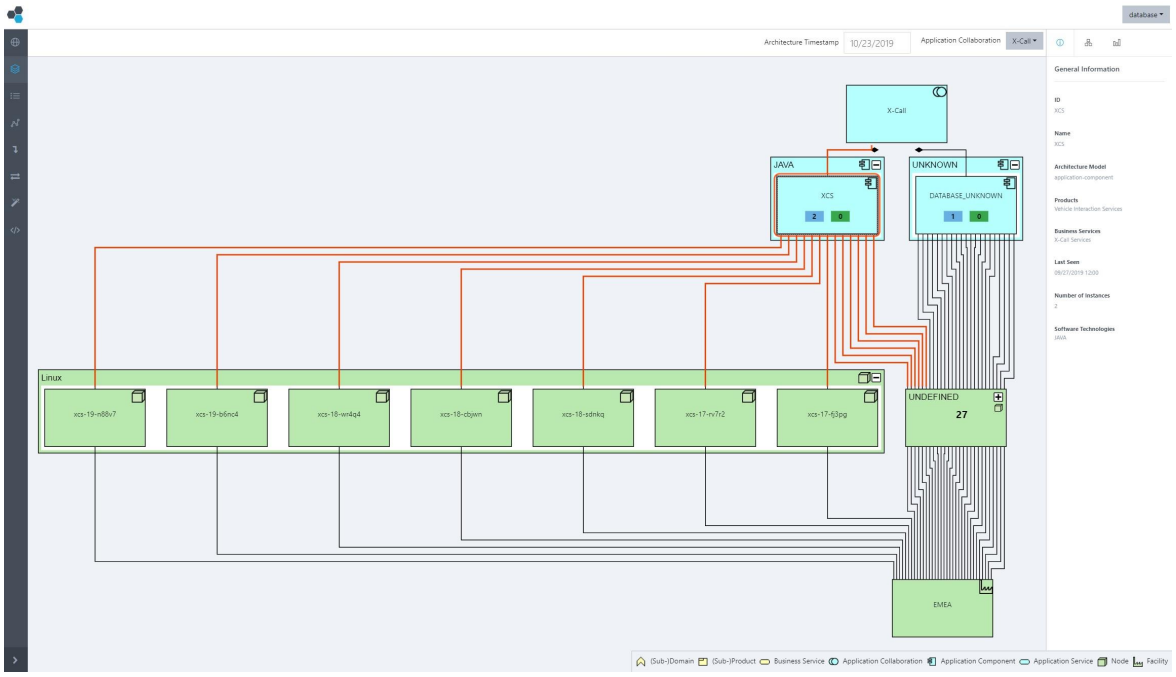


Figure 3.8.: Application Landscape View

3.6.3. Table View

The Table view provides a tabular representation of the underlying data set. Distinct tabs show elements according to their type and data can be sorted ascending or descending. Since all elements are somehow connected hierarchically with each other according to our data model as described in section 3.4, selecting one element will filter all data to show only the elements that are associated with the selected element (directly and transitively). This view uses pagination to limit the amount of elements per page. An example of this view can be seen in Figure 3.9.

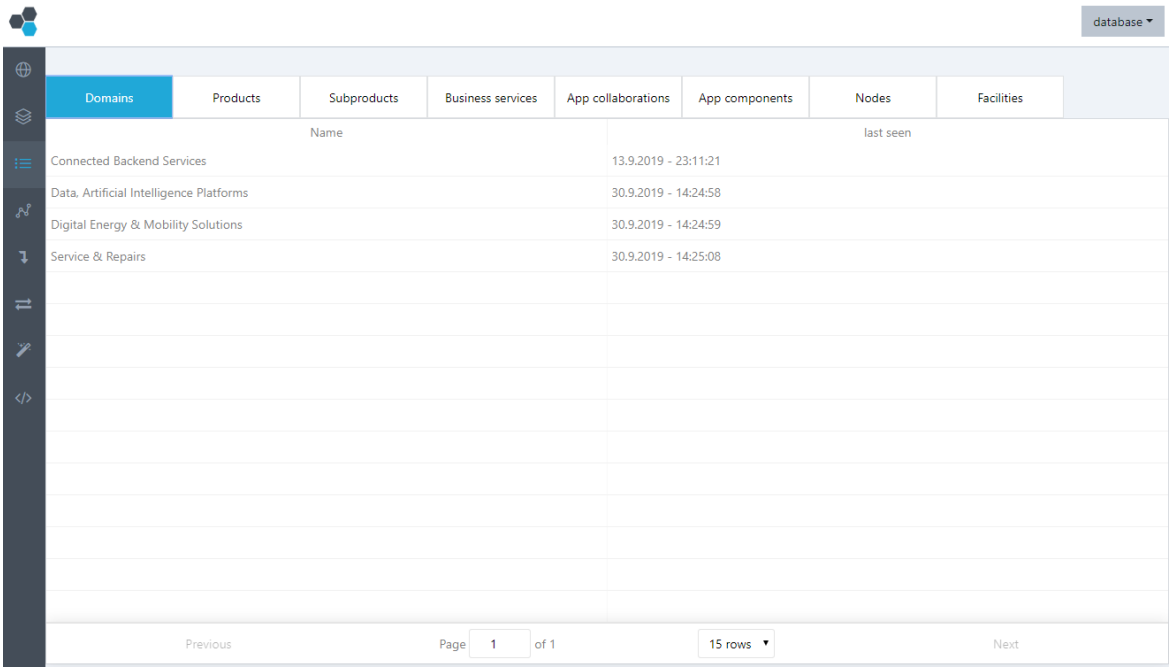


Figure 3.9.: Table View

3.6.4. Communications View

The Communications view shows all recorded communication between individual architecture elements. The communication relationships can be drilled up or down within the hierarchy to reflect the desired level of abstraction. (Note: The prototypical frontend only supports viewing communication on application component and business service level for now but the backend already supports drillups and drilldowns to any level in the hierarchy.) An example of this view can be seen in Figure 3.10 where communication between application components is shown.

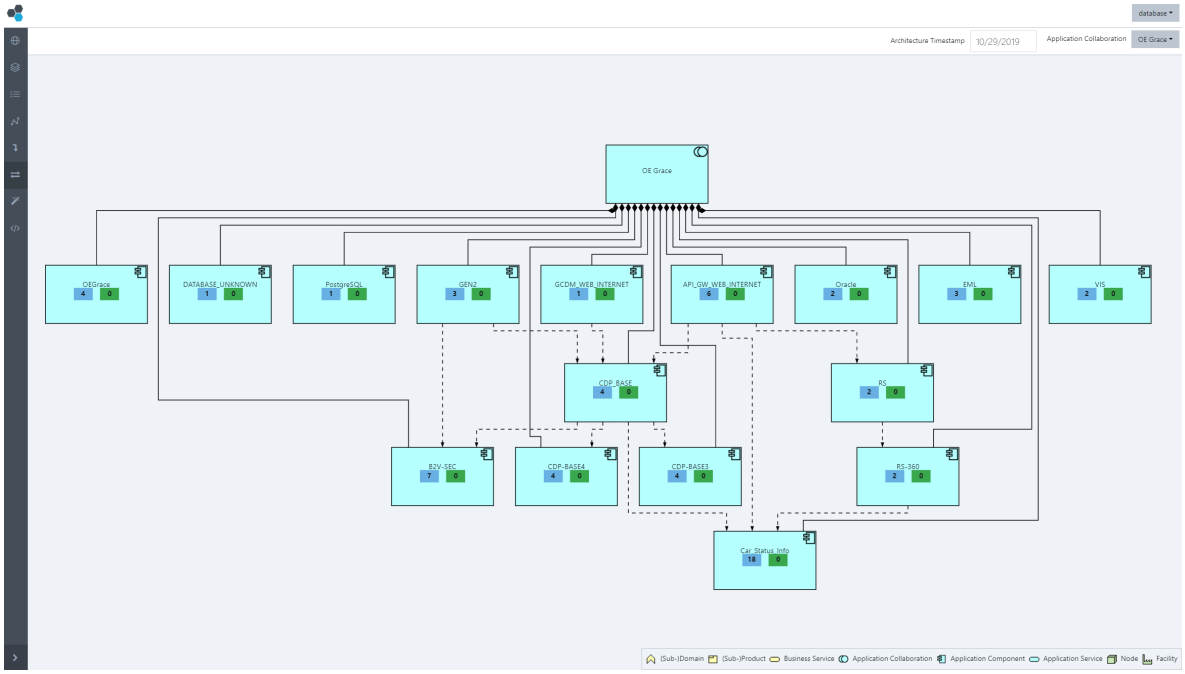


Figure 3.10.: Communications View

3.6.5. Application Interaction View

The Application Interaction view is similar to the Communications view and shows communication between application components. The difference is that it adds another layer of detail by allowing the selection of individual application interactions which represent requests directed at the monitored system. Selecting one of these application interactions highlights the path the request took through the system in the visualization and shows which interfaces were called and whether the communication occurred synchronously or asynchronously. An example of this view can be seen in Figure 3.11.

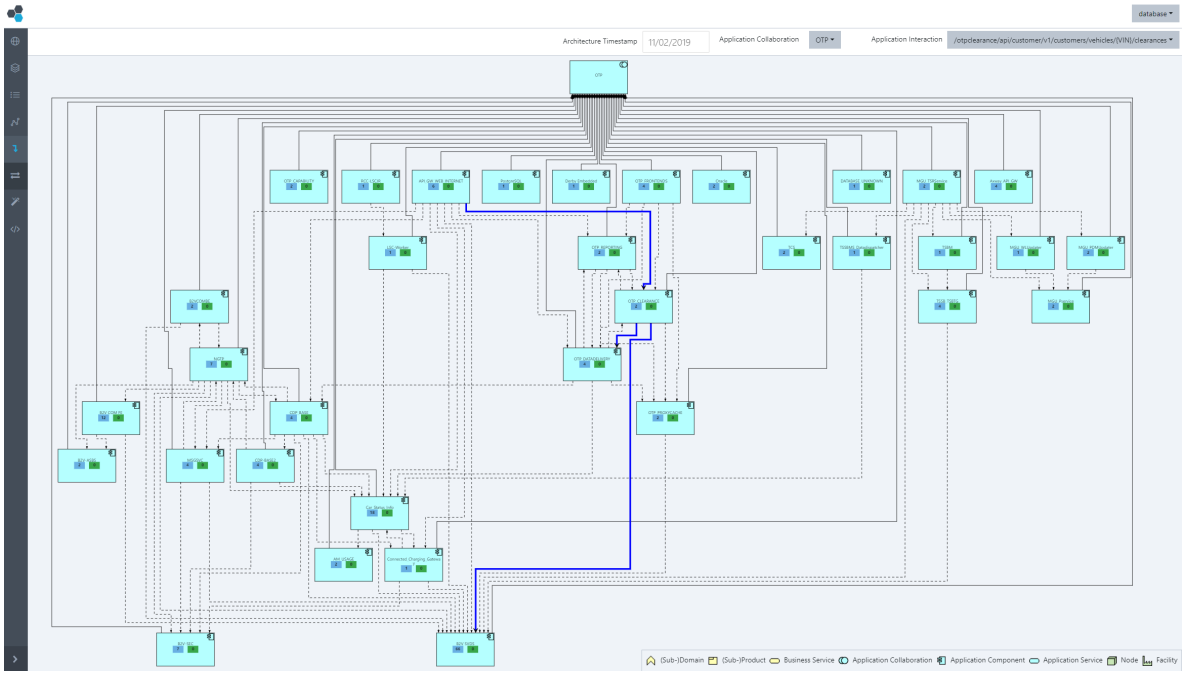


Figure 3.11.: Application Interaction View

3.6.6. Comparison View

The Comparison view shows two visualizations side by side at the same time. As the name suggests, this view is intended to be used when comparing two states. One can choose to compare the same IT artifact at different points in time or two different IT artifacts at the same point in time (or a mixture of both). Differences are highlighted in green when elements were added or red when elements were removed. Since the proposed solution approach is primarily designed to document the as-is landscape, the prototype does not offer the capability to compare the as-is landscape with planned (future) states, this could however be added in future work. (Note: The prototypical frontend only supports viewing comparisons on application component and business service level for now but the backend already supports drillups and drilldowns to any level in the hierarchy.) An example of this view can be seen in Figure 3.12.

Figure 3.12.: Comparison View

3.6.7. GraphQL View

While this view does not technically qualify as visualization, for the sake of completeness it shall still be mentioned. This view basically provides a direct link to the GraphQL backend. The left side serves as live editor where GraphQL queries can be written and issued to the corresponding resolvers. Since GraphQL is self-documenting, it is possible to display and explore the underlying GraphQL schema whereas the editor assists the user in writing queries by highlighting syntax errors and providing available valid options at any point inside the query. Query results are displayed on the right side. This view is primarily directed at developers to test their queries, e.g. when extending existing views or implementing new ones, and technology-affine stakeholders who are not interested in the visualizations and want to work with the underlying data set directly. An example of this view can be seen in Figure 3.13.

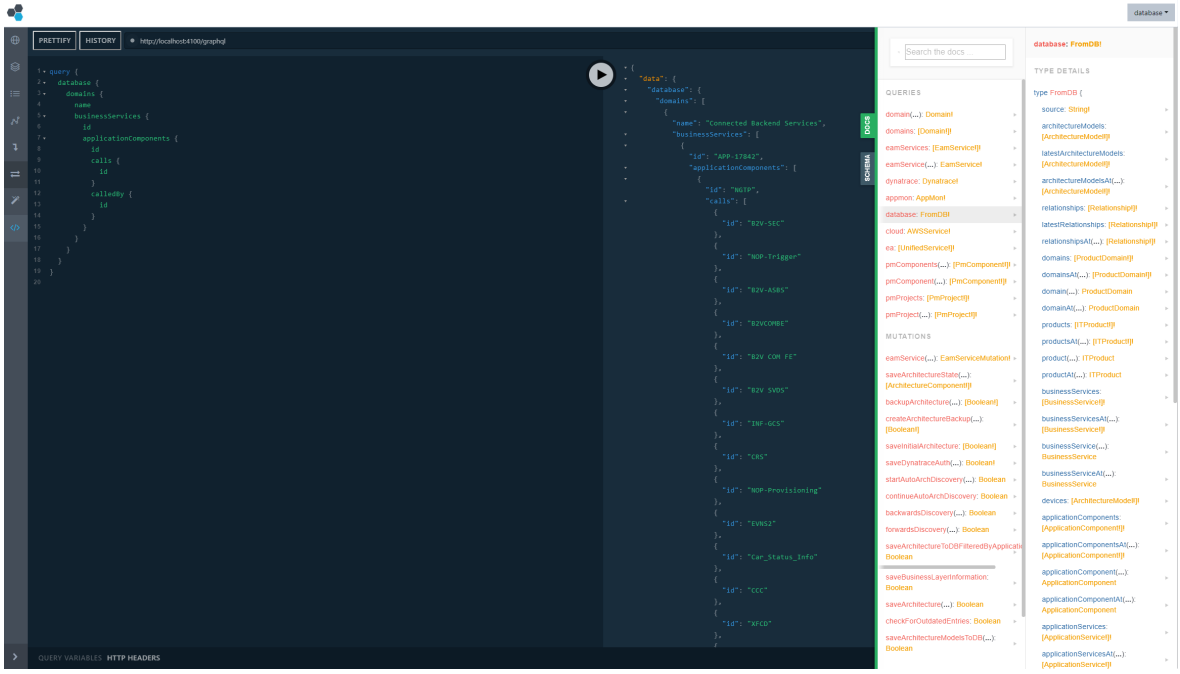


Figure 3.13.: GraphQL View

4. Evaluation

The previous chapter presented the proposed solution approach and explained the implementation details. This chapter will focus on the evaluation of the implemented solution approach.

The evaluation is separated into two sections, a quantitative and a qualitative analysis. The quantitative analysis will evaluate the implemented solution approach quantitatively on basis of statistical data that was gathered during the execution of the trial discovery run while the qualitative analysis will evaluate the solution approach on basis of multiple expert interviews that were conducted with EA practitioners from our industry partner.

4.1. Quantitative Analysis

In this section, the proposed solution approach will be evaluated on the basis of data gathered during the trial discovery run of the implemented Automated Architecture Discovery Algorithm (AADA). Every time AADA completed an iteration, a function was called that queried the database for relevant metadata including how many new elements were discovered during the particular iteration, which was stored in a local log file for further analysis later on.

4.1.1. Discovery Run

After multiple test runs that were used to test and debug AADA, a final discovery run was started on September 12th, 2019, which collected all the data used for the evaluation.

The discovery run was performed on a physical machine with moderate hardware and the connection to the monitoring server, which is only available within the internal network of our industry partner, was established through a VPN connection.

A total of 174 iterations were completed successfully starting September 12th and ending September 27th, thereby covering a time window from August 15th (00:00) up until September 27th (12:00). The algorithm did not run continuously but was stopped and resumed various times due to network failures and the fact that the machine running the algorithm was needed for other purposes as well.

The individual timeframes that were used to iterate through time were set to 6 hours, simply for the reason that they provided an acceptable tradeoff between runtime of the algorithm and data coverage from the monitoring server, which was established during the initial test runs. Further research is required to optimize the timeframe setting.

One iteration took roughly one to two hours to complete. Keep in mind though, that since our industry partner asked not to put too much load on the monitoring server, all requests were executed sequentially. Executing some requests in parallel would have most

likely reduced the duration of the discovery algorithm per iteration, but it is impossible to conclusively say by how much.

Metadata Discovery Run - Summary:

- Start: September 12th
- End: September 27th
- Timeframe Size: 6 hours
- Completed Iterations: 174
- Iterations backwards: 112 (28 days)
- Iterations forwards: 62 (15.5 days)
- Duration per iteration: ~60 - 120 minutes

4.1.2. Findings

Observations about the data extraction process

- **High robustness and reliability**

  AADA proved to be very robust and reliable during its execution. Encountered issues such as not receiving a response from the server for a certain query or longer response times from the server were handled accordingly and did not result in the termination of the discovery process. At one point, AADA ran for almost 24 hours straight without any issue but had to be terminated due to a forced shutdown of the VPN connection initiated by security protocols from our industry partner. Still, the discovery process could unproblematically be resumed from where it had left off.

- **Incomplete data after a while**

  Once AADA had traversed roughly 10 days into the past, the monitoring server stopped responding with PurePaths. This is most likely because of storage limitations on the side of the AppMon instance deployed by our industry partner. Transaction flows, however, could still be requested even more than a month back into the past. By the time AADA completed the backwards discovery, data was still available that could be requested but due to time restrictions regarding the conduction of this thesis and the limitation described next, it was decided to conclude the backwards discovery after 112 successful backwards iterations.

- **Increasing response times further back in time**

  The further back in time data was requested, the longer the monitoring server needed to respond to the request. In addition to the increasingly higher response times, the responses became unreliable meaning that for some requests the server timed out and

did not respond at all or returned an empty data set. Again, this is most likely due to storage limitations from AppMon and an increased performance effort to accumulate data further back into the past.

- **Feasibility**

  Thanks to the observed high robustness and reliability of AADA, the acceptable tradeoff between algorithm runtime and timeframe coverage (up to 2 hours runtime to cover a timeframe of 6 hours), the ability to trigger the discovery algorithm automatically without further manual input once it is implemented and the fact that the discovery process can be executed on moderately equipped hardware, it can be claimed that extracting information from runtime data is a feasible approach for EA-related purposes.

Analysis of discovered elements

Table 4.1 shows an overview of all discovered elements during the discovery run distinguished by the types defined for the data models. The most important elements are the application components which represent our microservices, the nodes on which they run and the application interactions and their corresponding communication relationships.

Architecture Models	**Amount**
Application Components	221
Application Services	442
Application Collaborations	73
Application Interactions	1141
Nodes	5805
Facilities	1
Domains	4
(Sub)Products	46
Business Services	79
Relationships	**Amount**
Hierarchy	12544
Grouping	3086
Communication	2250
Annotations	15432

Table 4.1.: Discovered Architecture Elements

- **Application Layer**

  A total of 221 application components were discovered over the course of the entire discovery run covering 43.5 days. The interesting part is that 196 of these 221 components (~88.7%) were identified during the first iteration covering merely a timeframe of 6 hours. Within the first day of discovery (4 iterations), the total rised to 211 (~95%). The remaining 10 application components were discovered dispersed over the remainder of the discovery run duration.

  The discovery of these components is visualized in Figure 4.1 where the amount of discovered elements was plotted against the iterations of the algorithm. To interpret the graph correctly, one needs to understand that the numbers on the x-axis represent iterations where the numbers themselves are continuously rising and the algebraic sign denotes whether it was an iteration during the backwards discovery (negative sign) or the forwards discovery (positive sign). Because of this, it appears like the amount of discovered elements declined and then increased again which does not make sense in this context. The graph was designed like this to better visualize how the amount of discovered elements is distributed over the backwards and forwards discovery. To this effect, one should read the data from right to left starting at iteration -1 up to iteration -112 (backwards discovery) and then continue from iteration 113 going from left to right up to iteration 174 (forwards discovery).

  Since application services were added automatically to each application component as explained in subsection 3.4.2, they are directly dependent on the discovery of application components and therefore trivial for the analysis.

  Application collaborations represent AppMon applications. The total of identified application collaborations amounts to 73, which was observed during the first iteration and did not change at all over the course of the discovery run. Since some of these collaborations do not include any further architecture elements and hence, only represent empty containers that might at some point have encompassed multiple Agent Groups and their interrelationships, there is reason to believe that these elements maintained by AppMon are never updated by the APM tool.

  Additionally, these application collaborations were of no particular interest to the advisors from our industry partner because they do not directly represent applications as perceived by and maintained within the federated information systems of our industry partner. Therefore the amount of identified application collaborations bears no expressiveness.

  Given an application landscape in which AppMon applications are configured to directly match an IT application as perceived by the organization, these might become more relevant and should probably entail a change of the underlying data model where they replace or are merged into the business services.

  Application Interactions represent PurePaths, i.e. requests directed at the monitored systems originating from external sources such as browsers or mobile devices. Over

the course of the discovery run, a total of 1141 unique application interactions were identified.

The observed application interactions were plotted in Figure 4.3 in conjunction with their corresponding communication relationships. (The graph is interpreted the same way as explained for the graph about application components.)

Considering that a few days into the past, no further PurePaths could be retrieved and the majority of the discovered ones were identified during the beginning phase of the backwards discovery and during the later ensuing forwards discovery, which was still detecting new PurePaths even in its final iteration, it can be suspected that there are still a lot of application interactions that were not observed.

In addition to the lack of received PurePaths further back in time, the amount of identified PurePaths is heavily limited by the AppMon constraint to only respond with the 100 most recent PurePaths for the given timeframe of the request. Depending on how busy the monitored system is, the requested timeframe of 6 hours may effectively only cover the last few minutes (or even less).

This is even further exacerbated by AppMon's design choice to not mask parameters (i.e. the variable part) within request URLs. In the worst case, this can result in requesting PurePaths and receiving 100 entries that essentially represent the same request just with different parameters. This was the case for the initial test runs, which resulted in a lot of redundant entries in the database. This was solved through a multitude of regular expressions that identified the variable part of request URLs and reduced the storage of essentially duplicates within the database significantly. Despite that, some redundancy still remains as the regular expressions already in place are not sufficient to detect all occurring parameters within request URLs caused by the inherent diverse nature of requests. To improve the detection rate, machine learning could be utilized but this is beyond the scope of the prototypical implementation presented in this thesis.

Ideally, observing the runtime data over a sufficiently long enough period of time, the received application interactions should be eventually consistent. Because of the presented limitations, as another possibility to improve the coverage of unique requests, one could implement different timeframes for requesting Transaction flows and PurePaths, e.g. one request with a 6 hour timeframe to receive a complete Transaction flow and six requests with a 1 hour timeframe to (hopefully) receive a more complete set of PurePaths. Further research is needed to tweak the approach and optimize the received results.

- **Technology Layer**

  Nodes represent hosts on which the application components run. Surprisingly, quite a lot of nodes were identified during the discovery process (a total of 5805), which appears to be somewhat disproportionate considering only 221 application components were discovered during the same time span, especially since there are multiple spikes arising suddenly during the backwards discovery.

This can be observed in Figure 4.2. (The graph needs to be interpreted the same way as explained for the graph about application components.) These observed spikes do not match the discovery behavior of the application components when comparing the two graphs.

In order to comprehend why so many nodes were discovered in sudden bursts, EA practitioners were asked about this during the conduction of the interviews for the qualitative evaluation. According to them, this phenomenon is accounted for by the usage of pods.

A pod is a concept of Kubernetes.[1] It represents a deployable unit that is used to run a single instance of an application by encapsulating and managing its container. Our industry partner uses OpenShift technology, which leverages the concept of pods.[2]

The sudden rise of nodes as observed by the discovery algorithm can be explained through the lifecycle of pods. Since pods can not be modified while running, they are terminated and recreated with adjusted configuration to reflect the changes. The pods do not maintain state during this process, which is most likely why AppMon is unable to recognize that essentially the same node (host) was redeployed.

Therefore, AppMon identifies the host as new and treats it as such. Subsequently, since AADA directly depends on the information provided by AppMon which presents the essentially same node with a new ID, it is stored as new node within our database without removing the old one.

From AADA's perspective an application component appears to be running on a new node whereas the old node is not observed during that timeframe. Since not observing the old node in the runtime data is insufficient to conclude that the node does not exist any longer as defined by the overall solution approach, the change goes unnoticed.

Eventually, the "lastSeen" value of the old node, which would no longer get updated, would indicate that the old node is most likely no longer available, but until a reasonable amount of time has passed to justify such assumptions, the nodes could have been redeployed multiple times, which in the long run poses a problem because the database gets flooded with outdated data that is not recognized as such. This issue definitely needs to be addressed in future research.

Since there is only one facility available for discovery due to reasons explained in subsection 3.4.2, a further analysis regarding facilities does not provide any additional insight and is therefore omitted.

[1] kubernetes.io/docs/concepts/workloads/pods/pod-overview/
[2] docs.openshift.com/enterprise/3.0/architecture/core_concepts/pods_and_services.html

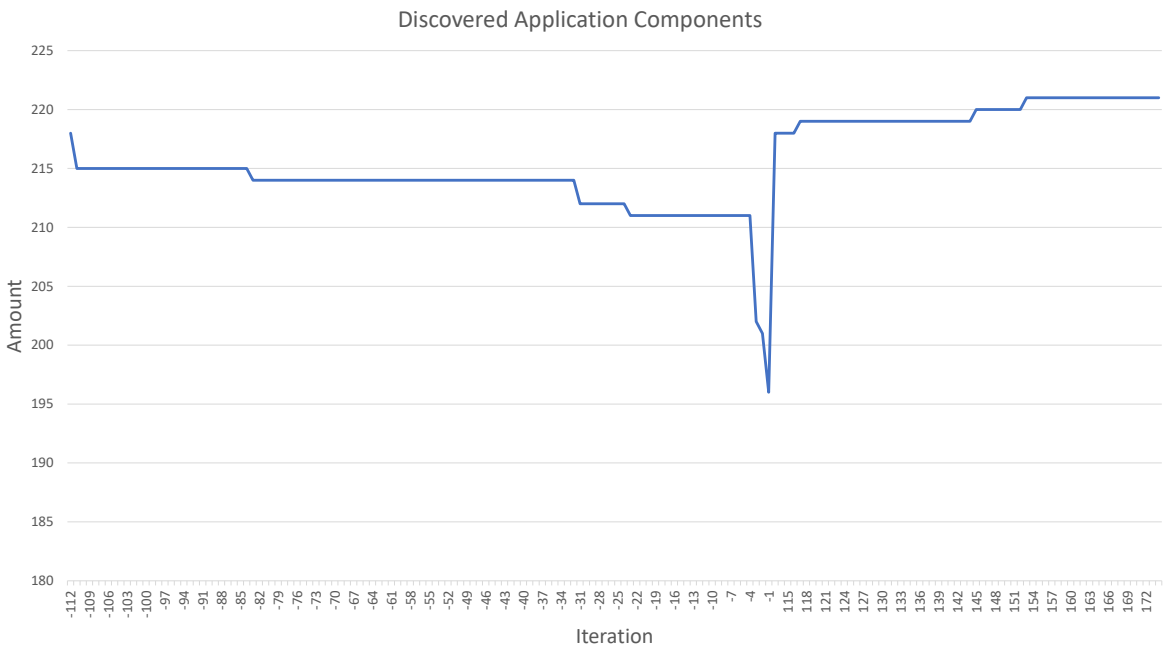


Figure 4.1.: Discovered Application Components

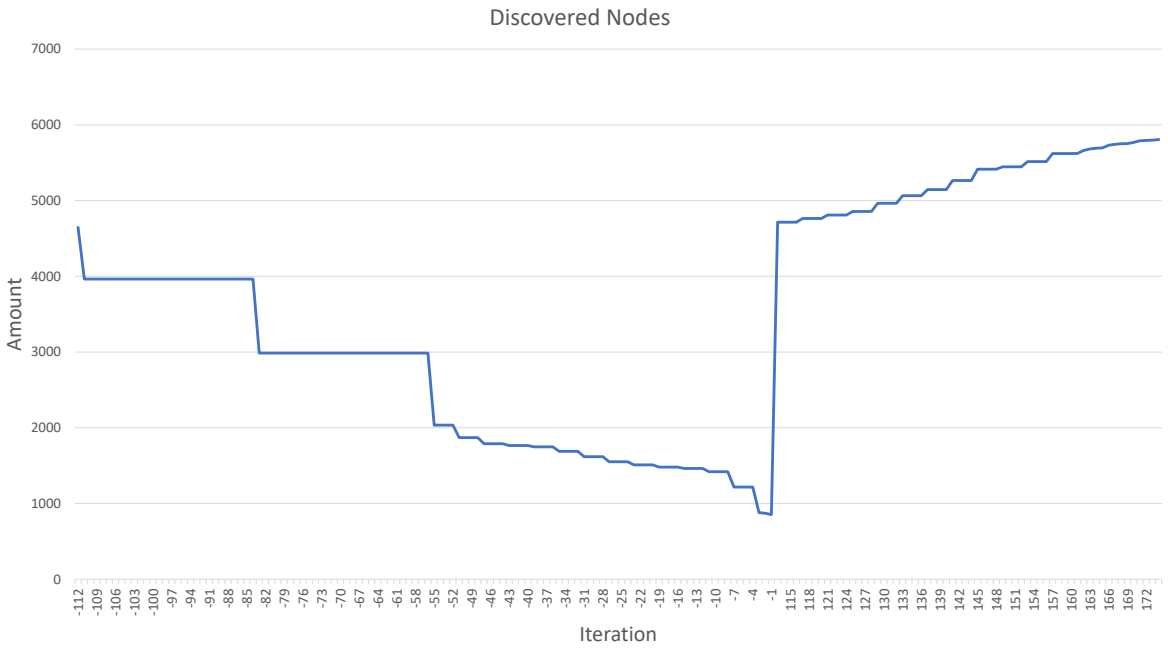


Figure 4.2.: Discovered Nodes

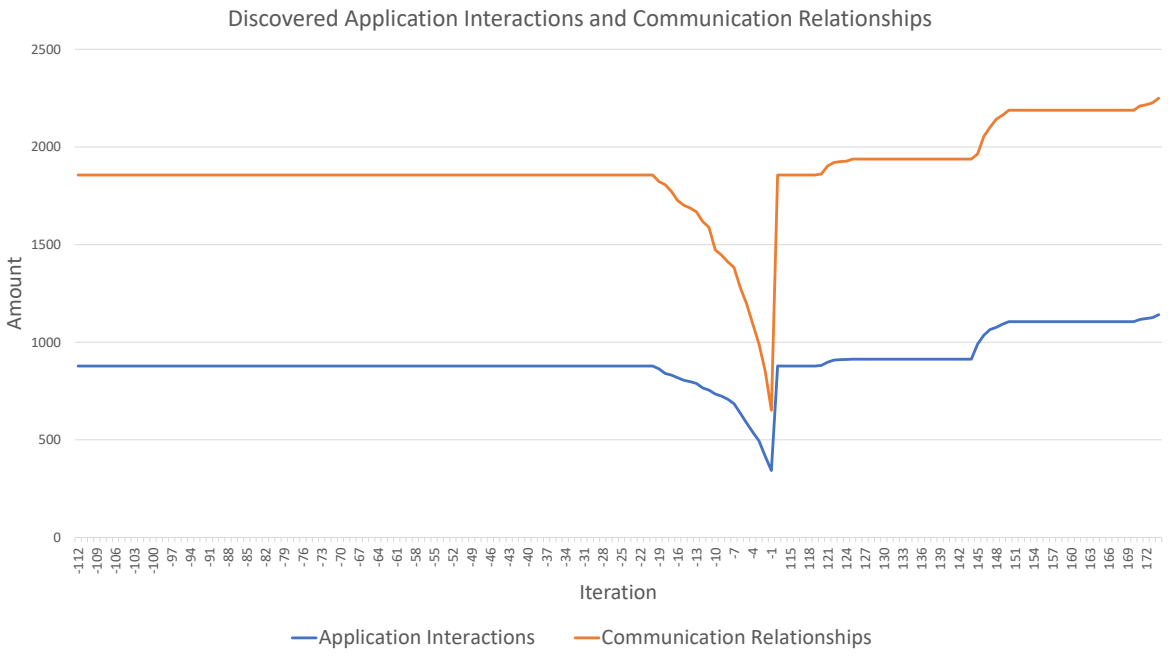


Figure 4.3.: Discovered Application Interactions and Communication Relationships

- **Business Layer**

  As explained before, none of the business layer elements are directly monitored by the APM tool and therefore, technically speaking, one can not claim to have "discovered" such elements.

  The elements were extracted from the provided manual documentation by looking up the microservice ID each time a microservice was actually discovered within the runtime data and associating the hereby found information with said microservice.

  This resulted in the "discovery" of 4 domains, 46 products and subproducts and 79 business services, which represent the IT applications as defined by our industry partner.

Coverage and Accuracy

To determine how well the discovery algorithm performed in reconstructing the IT landscape, it is important to compare the automatically extracted information against the manual documentation.

Since our industry partner considers microservices too small of a unit and too large in numbers to justify documenting them within the centralized, federated information systems, a comparison against such a single source of truth is not possible.

As an alternative, the manual documentation which is a manual collection of information from different sources in the form of a Microsoft Excel sheet, the same manual documentation that was used to look up and extract information from the business layer, will be used to compare the discovered data against.

Since this manual documentation contains a lot of information including data from different hubs that are not part of the monitored landscape which embodied the discovery environment for AADA, it would obviously not make for a fair comparison to compare the discovered data against data that was technically impossible to discover.

Thus, to allow for a reasonable comparison the author proceeded as follows:

1. All actually discovered application components were looked up manually and marked within the manual documentation. The difficulty hereby was that there is no uniform naming convention in place to unambiguously match the AppMon IDs with the manually documented IDs. The IDs were therefore normalized as described in subsection 3.4.2 and matched to the author's best knowledge.

2. All remaining entries in the manual documentation not marked during the previous step were examined in order to identify whether they were supposed to be monitored within the application landscape that AADA run on by looking for an entry in the respective AppMon agent field. If there was no agent specified, the entry was removed from the documentation.

Note: Some of the remaining entries were declared as "retired". These entries were not removed, since part of the proposed solution approach is to request runtime data from the past and therefore, these elements could still have been discovered even if they are not considered relevant anymore.

The remaining entries in the documentation were used for the comparison and the results are presented hereinafter.

- **Application components**

  After adjusting the manual documentation as described above, a total of 167 microservices remained that are considered discoverable. AADA discovered 159 of these 167 microservices (~95%). One might even argue that 159 of 164 (~97%) were automatically discovered because three of the manually documented microservices are reported as "retired".

  In any case, while both these numbers represent a high coverage and exceedingly satisfying accuracy, the even more interesting observation is that AADA discovered 221 application components, which is a lot more than the 159 that could be matched.

  Further analysis revealed that 8 of the 221 discovered application components represent databases which are treated as Agent Groups by AppMon but not as microservices by our industry partner. The remaining 44 surplus application components could not be matched to any of the manually documented microservices.

  Some application components share common naming patterns with already marked microservices from the manual documentation which could indicate that they are one of the cases where multiple Agent Groups jointly form one microservice. Should this be indeed the case, it would have no effect on the achieved coverage since the manually documented microservice was already discovered, only the amount of surplus application components would decrease.

  Another possibility is that the manual documentation is outdated or incomplete or simply erroneous (which ironically would verify the problem description stated in section 1.1). Indication that this might actually be the case, is the fact that a lot of the microservices that could be matched are reported as "in planning" and/or do not have an associated AppMon agent specified in the manual documentation, which is obviously false.

- **Business services**

  AADA "discovered" 79 of 92 (~86%) possibly discoverable business services that represent IT applications as defined by our industry partner.

  While this can be regarded as a decent to good coverage, manually comparing the individual elements revealed that the manual effort of matching the discovered application components with the manually documented microservices undertaken by the author as described above resulted in more matches than AADA achieved automatically. This can be explained by the lack of a uniform naming convention. The normalization of IDs that is utilized to programmatically match the microservice IDs is not sufficient to achieve the same coverage as when matched manually.

  Assuming a standardized naming convention was in place, AADA would have automatically detected 88 of 92 (~96%) business services or 88 of 90 (~98%) when the retired microservices are neglected.

- **Products and Subproducts**

  A total of 17 products were "discovered" by AADA after automatically matching the application components to microservices. Even without accounting for the inaccuracy caused by the error-prone matching explained above, this results in a coverage of 100% since the manual documentation contains exactly 17 products that are considered discoverable.

  When it comes to subproducts, 29 of 31 (~94%) possible ones were detected. When accounting for the matching inaccuracy, all 31 of these subproducts would have been detected, which again results in a coverage of 100%.

- **Domains**

  AADA was able to identify 4 out of 4 possible domains resulting in a coverage of 100%.

- **Remaining Elements**

  The remaining elements of the presented data model can not be compared to the manual documentation because it does not contain the necessary information. To that effect, all provided information discovered by AADA regarding these elements can be considered added value.

4.1.3. Conclusion of the quantitative analysis

Taking into consideration all the findings of the quantitative analysis, it can be claimed that the proposed solution approach shows more than promising results.

From a technical standpoint it is simple to execute and requires no further manual input once implemented. Furthermore, its execution does not require many computational resources but still exhibits a high robustness.

Additionally, the achieved coverage and accuracy are exceedingly satisfying and even if the results might be slightly skewed due to the manual adjustments to the manual documentation to allow for a reasonable comparison, which might not have been perfectly accurate, the automatically discovered elements, which came at no further cost since the monitoring tool was already in use before, still provide additional value. The discovered application interactions alone, for example, even if incomplete (but eventually consistent) contain information which is not documented at all so far.

All in all, it can be concluded that the implemented solution approach provides valuable information at no additional cost and therefore no argument can be presented against having it run in the background gathering data that can assist the documentation process, among other applicable use cases.

4.2. Qualitative Analysis

In this section, the proposed solution approach will be evaluated on the basis of multiple interviews which were conducted with EA practitioners from our industry partner.

All interviews were conducted face-to-face on the basis of a predefined questionnaire and took roughly an hour to one and a half hour depending on how familiar the respective interviewee already was with the solution approach and how much he had to say. The interviews were recorded (with the explicit permission of the interviewees) and later transcribed. The used interview questionnaire and part of the transcribed interviews can be found in the appendix under appendix A.1 and A.2 respectively.

The interviews were structured as follows:

1. The proposed solution approach was presented and explained on the basis of a digital presentation.
2. The first set of questions related to the concept of the solution approach were asked.
3. The prototypical implementation was demonstrated with the data set gathered during the trial discovery run.
4. The second set of questions related to the implementation of the solution approach were asked.

A total of seven interviews were conducted and build the basis for the following evaluation. It will be structured as follows:

1. **Question:** The question will be presented. (Denoted as "Q".)
2. **Feedback:** Feedback from the interviewees will be summarized. (Denoted as "F".)
3. **Assessment:** The author will critically assess the presented feedback. (Denoted as "A".)

4.2.1. Relevance of problem description

Q: To what extent do you accept the stated problem description? Do you differ in opinion?

F: **Agreement:**

- fully support the problem description
- time and budget restrictions apply
- documentation is not a priority
- data outdated and incomplete indeed true

Disagreement:

- lack of clear responsibilities for documentation not true
- uncertainty whether problem is caused by lack of appropriate tools, lack of interest or lack of responsibility
- doubt that manual documentation is actually that time-consuming
- compared to other activities manual documentation does not take that much time relatively seen

A: This question aims at scrutinizing the problem statement which was primarily derived from research literature (see section 1.1). Since it is used as motivation behind conducting this thesis, it is important to put it to the test and verify if practitioners from the field will confirm or disprove it.

In this case, all interviewees generally agreed with the stated problem description. It was mentioned that documentation is often not a priority and that there is neither an extrinsic motivation to do so nor any sort of penalty for not doing so. One person described this as "lack of carrot and stick". Due to time and budget restrictions the teams focus mostly on getting the work done first and only then take care of the documentation, if at all. Therefore, technical debt is amassed in the organization which gets increasingly more difficult to cut down on over time. While all interviewees agreed that the documented data is indeed outdated and incomplete, there was a disagreement whether the stated reasons actually hold true. A lack of clear responsibilities regarding the documentation was reported as false which is in direct contradiction to [10]. Additionally, there was uncertainty whether the documentation problem is caused by a lack of appropriate documentation tooling or something else like lack of interest. Also, one person questioned the statement that manual documentation is actually that time-consuming. Relatively seen, it only takes up a negligible amount of time considering that the majority of the work is the actual implementation. While the causes can't be confirmed conclusively, the stated problem description does hold true in general and is therefore a relevant foundation for research in this field.

4.2.2. Solution Approach

Q: How do you rate the approach of extracting architecture information from runtime data in order to assist the IT landscape documentation?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

F: **Benefits/Affirmations:**

- usage of runtime data extremely important
- medium to long-term no other way
- depiction of reality
- automatable
- collection of data en passant

Limitations/Concerns:

- certain inaccuracy always present
- important but insufficient
- lack of explanation
- understanding of architecture not possible
- connection to source code missing
- validation necessary
- focus on as-is landscape
- too technical and finegrained
- (only partial view if not everything is monitored)

Mean Grade ($\varnothing Grade[n = 7]$): 1.86

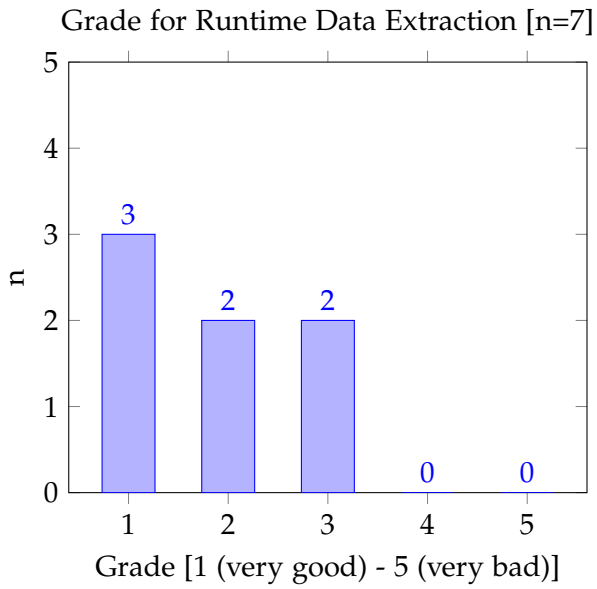


Figure 4.4.: Grade for Runtime Data Extraction

A: This question aims at evaluating the usage of runtime data to identify relevant information independent of the JSON configuration file extension.

In general, extracting architecture information from runtime data was perceived as an useful and extremely important approach. One interviewee went as far as calling it the only feasible approach in the long run, especially because it can be automated and relevant data can be gathered en passant without requiring a lot of resources as verified by the prototypical implementation of this thesis.

Also, automating the approach is viewed as absolutely necessary due to microservices being deployed and utilized in such large numbers that documenting them manually is regarded as infeasible and not scalable for the future.

Another key advantage identified by the practitioners is the fact that runtime data represents the truth. However, while runtime data is a genuine depiction of reality, it was argued that there is also always a certain inaccuracy present. The reasoning behind this is that runtime data will always show the presence and therefore existence of certain elements because they were observed within a specific timeframe but can never conclusively prove the absence of elements because not having observed an element within the timeframe can always be attributed to lack of activity or failure in observation.

Furthermore, while runtime data can show that a communication took place between two elements it can not explain why these elements communicated with each other. Therefore, it was argued that an actual understanding of the underlying architecture is not possible.

Additionally, the longer the discovery process is executed, the more elements will be discovered over time but no element will ever be removed automatically due to the aforementioned reasons. Thus, at some point it is no longer possible to unambiguously assert that the extracted information is still relevant.

To that extent, some means of validation is necessary. One suggestion was to include more information at runtime like the source code itself to validate the observed runtime data and be able to, e.g. identify which elements can (and should) be discovered since they are defined in the source code and subsequently be able to analyze why something was not discovered. This linkage of additional information as extension to the runtime data extraction approach is covered in the next question regarding the JSON configuration files.

Another issue that was mentioned is that runtime data can only ever provide information on the as-is landscape, never on planned (future) states. It was claimed however, that for many EA stakeholders it is very important to be able to see how the IT landscape will look like after the next deployment (n+1 analysis). Consequently, it is only possible to react and not act proactively. This is indeed an issue that can not be solved by the usage of runtime data and represents a limitation of the proposed solution approach.

Also, runtime data is naturally very technical and finegrained and therefore not suitable to be used as a basis when communicating with stakeholders outside the technical domain (e.g. management). It needs to be abstracted and translated into terms and KPIs appropriate for the stakeholders that the information is directed at.

A final concern was that the approach only works as intended when all relevant systems are monitored, which is not the case for our industry partner, e.g. some legacy systems simply do not support that functionality. This results in receiving only a partial view of the IT landscape. While this is indeed true and a problem, it also conflicts with the prerequisites described in subsection 2.2.2 that clearly state that all relevant microservices need to be monitored to receive a full picture.

All in all, it can be concluded that while the runtime data extraction approach is perceived as valuable and promising (mean grade 1.86), it is also deemed insufficient for reasons explained above.

Q: How do you rate the approach of maintaining further relationship information within JSON configuration files?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

F: **Benefits/Affirmations:**

- necessary
- good approach in relation to other approaches
- JSON preferable to other formats due to simplicity and validation ability

Limitations/Concerns:

- every bit of contained information needs to add value
- should not contain unnecessary information to minimize maintenance effort
- handling of unclear or unknown information possibly an issue
- needs to be enforced
- anti-pattern
- decentralized approach might not be ideal
- technology affinity required
- standardized structure required

Mean Grade ($\varnothing Grade[n = 7]$): 2.14

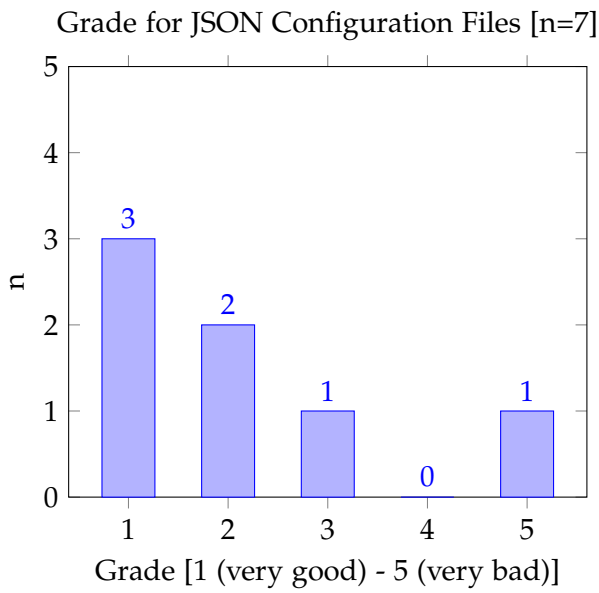


Figure 4.5.: Grade for JSON Configuration Files

A: This question aims at evaluating the usage of JSON configuration files to enrich the dynamic information from runtime data with static information from federated information systems thereby bridging the gap between the application layer and the business layer.

The approach of adding JSON configuration files to the microservices was considered a good approach in relation to other approaches and also deemed necessary.

Some argued that it does not necessarily need to be the JSON format and that other formats would suffice as well, but in general JSON was considered preferable to other formats because of its simplicity and the ability to automatically validate it.

While the JSON configuration files were received mostly positively by the practitioners, a concern was raised that the amount of contained information should be kept to a minimum of only the most important references. In order to minimize the manual maintenance effort, no unnecessary information should be added and therefore every piece of contained information needs to represent added value.

Another issue that was mentioned is that despite its simplicity, some not technology affine people might still have problems with JSON, this should however not be a big issue since the responsibility to create and maintain the configuration file is assigned to a whole team so even if some people might still have issues, others would be there to assist them. Alternatively, there is always the possibility of providing workshops to teach the correct handling of these files, therefore this is not considered an issue by the author.

To prevent the potential issue of different teams using different notation for the configuration files, a standardized structure needs to be defined by the responsible authority which constitutes a necessary effort but is only required once when integrating the approach into the organization and therefore should not pose much of a challenge.

Additionally, some practitioners expressed uncertainty about what should be done if necessary information for the configuration files is unclear or unknown at the time of creation. Regarding this issue, one interviewee suggested centralizing the approach, i.e. instead of maintaining one configuration file for each microservice, there should be one configuration file for all microservices where each team is responsible for their part and unknown information could be filled out by other teams. This would however most likely create the issue that no one feels responsible anymore since so many people could potentially fill in the required fields now. Also, validating the file which is covered by the next question is made more complex by this. Since the JSON configuration files could not be implemented during the conduction of this thesis, it is difficult to say how this issue should be handled.

In any case, the one concern raised by most of the interviewees was that such an approach requires people to do something and no one does anything that requires extra effort if it is not considered beneficial. One person went as far as calling the approach of manually maintaining configuration files an anti-pattern because no one would adhere to it. Other practitioners stated that the creation and maintenance of the configuration

files need to be enforced by the governance team, ideally by also convincing the teams responsible for the files of the achieved benefits of the approach.

In general, the JSON configuration files approach was deemed valuable and necessary even though it raised a few concerns about its feasibility due to the manual effort required. However, no better alternative could be identified by the practitioners (mean grade 2.14).

Q: How do you rate the approach of ensuring the maintenance of the configuration files through JSON Schema validation?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

F: **Benefits/Affirmations:**

- indispensable
- no alternative

Limitations/Concerns:

- contingency plan required

Mean Grade ($\varnothing Grade[n = 7]$): 1.57

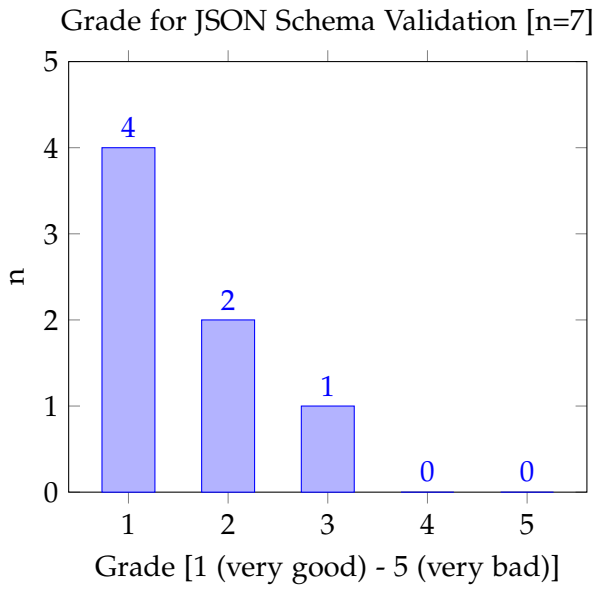


Figure 4.6.: Grade for JSON Schema Validation

A: This question aims at evaluating the usage of JSON schema to validate the JSON configuration files.

The interviewees unanimously agreed with the usage of JSON schema. JSON is perceived as a state-of-the-art format and should there ever be a better format, one can consider switching to the new format, but until then validating the configuration files with a JSON schema was perceived as indispensable and an absolute necessity. The validation was also seen as a necessary step to enforce the adherence to the manual maintenance of the configuration files.

However, a need for a contingency plan was proposed. In cases where certain key attributes are unknown or unclear and therefore omitted, ultimately resulting in failing the validation within the pipeline, some measure needs to be in place so that the validation constraint can be bypassed temporarily in order to ensure deployment in critical situations. This, of course, opens up the possibility to bypass the validation step more often than actually necessary, but can be kept in check by establishing very strict conditions for bypassing the validation.

Overall, the JSON schema validation was received very positively. The practitioners were convinced that there is no alternative and therefore it is a must for the proposed solution approach (mean grade 1.57).

Q: To what extent does the solution approach contribute in general to improving the EA documentation?

F:
- depiction of reality
- revelation of differences and problems
- validation of manual documentation
- support for root-cause-analysis, triage analysis
- every bit of information that can be extracted automatically represents an improvement

A: This question aims at identifying important contributions of the solution approach regarding the EA documentation.

The perceived contributions of the proposed solution approach are manyfold. Many of these were already mentioned in previous questions, which is why the interviewees did not have much to add when answering this question.

The most important aspects are that since the approach processes monitoring data which provides a representation of reality as explained earlier, it is able to reveal differences and problems regarding the manual documentation. In this regard, the automatically extracted data can be used to validate the manual documentation and thereby assist the documentation process in general.

But it is not limited to mere validation. It also supports different types of analyses such as root-cause-analysis and triage analysis. Some argued that these analyses are even more important than the documentation itself.

In general, it was stated that every piece of information that can be extracted automatically is considered an improvement compared to the manual processes prevailing at the moment. An example was mentioned where in order to assess if certain elements or attributes are still relevant, people will go around the departments asking other teams if they use these elements in any way, which was reported to be an extremely time-consuming and cost-intensive process. This information could be identified automatically by the implementation of the solution approach.

4.2.3. Technical Integration

Q: What barriers do you recognize regarding the technical integration of the approach, especially with respect to the technical requirements that need to be met?

F:
- mostly no or only minor technical barriers
- trivial
- usage of different monitoring tools
- monitoring tool limitations
- cloud migration
- needs to be seamless
- UX considerations
- security / privacy
- regulations forbidding data to leave the site
- real-time performance considerations

A: This question tries to gauge the feasibility of the solution approach with respect to the efforts that need to be undertaken on the technical side in order to implement it.

All interviewees regarded the technical integration as technically feasible without any major obstacles, some even going as far as calling it trivial. JSON is considered a standard and therefore not an issue. The usage of monitoring tools is already widespread within the organization and does therefore not require additional effort to set up. The biggest concern that was raised though, is the integration of multiple APM tools. This is indeed an issue, since as of now combining multiple APM solutions is not supported by the proposed solution approach. There was a disagreement however, whether this represents a technical issue that needs to be solved by extending the solution approach or an issue within the organization that should be solved by uniformly adapting one APM tool throughout the organization.

Additionally, in environments where microservices are partially deployed on-premises and partially migrated to the cloud, uncertainty was expressed as to how well the monitoring approach can handle this distribution. Since most APM solutions including AppMon provide support for both on-premises and cloud deployments this should not constitute an issue.

Furthermore, it was mentioned that the technical integration needs to be seamless and involve as little effort as possible. People using the implemented solution should not be required to have profound knowledge in how the tool works but simply be able to use it. To that effect, it needs to be intuitive and provide different levels of abstraction so that different stakeholders can access the information at a level of detail appropriate for them. This, of course, makes the implementation effort more complex as more views and QoL features need to be implemented, but this would be taken care of by the team

integrating the solution into the organization and should not pose an insurmountable technical barrier.

Another concern that was raised is that of security and privacy. Since the solution approach processes and stores sensible information, appropriate measures need to be taken to ensure compliance with prevailing laws and regulations. While this does not have to be a barrier per se, it does require additional implementation effort. One example was that some countries do not allow sensible information to leave the country and be stored elsewhere. Considering that many organizations operate data centers in different parts of the world, this is indeed an issue that needs to be addressed.

One possible solution would be to store the collected data locally within the allowed boundaries and aggregate them on demand thereby querying them from their respective storage locations. This does in turn affect the complexity of the solution approach regarding real-time performance of queries since data needs to be accumulated from different sources. Also, when the network is unreliable or a data source is not available it is no longer possible to achieve a complete view of the underlying IT landscape. Further research is required to determine how these issues can be addresses as they go beyond the scope of the prototypical implementation presented in this thesis.

Overall, all practitioners agreed that the technical integration is manageable as no identified barrier is perceived as insuperable and therefore should not prevent the proposed solution approach from gaining acceptance and being approved from a technical point of view.

Q: To what extent do you perceive the integration of the approach into a CI/CD pipeline as useful?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

F: **Benefits/Affirmations:**

- best approach to force people to do something
- useful
- no alternative
- validation
- automation
- Infrastructure as Code (IaC)

Limitations/Concerns:

- reliability an absolute requirement
- possibility to arouse hatred
- additional effort
- enforcement of adherence

Mean Grade ($\varnothing Grade[n = 7]$): 1.57

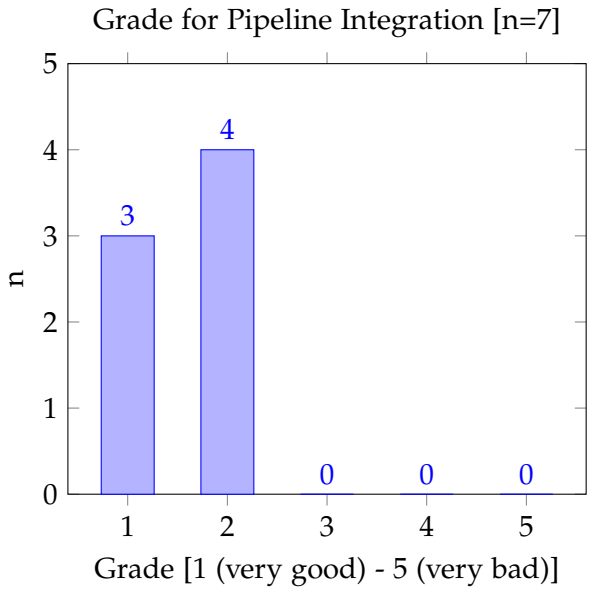


Figure 4.7.: Grade for Pipeline Integration

A: This question aims at evaluating the integration of the solution approach into a CI/CD pipeline by assessing advantages and possible limitations.

Pipeline integration is seen as very useful and reportedly the best way to get people to comply with the overall approach. Infrastructure as Code was mentioned in this regard and it was argued that pipeline integration enables automation and adds the ability for further validation. Not just the validation of the JSON configuration file that was covered in a previous question but also validation of the source code in general or compliance with governance regulations.

Therefore, the usage of a pipeline is perceived as necessary and that there is no alternative, but reliability of the approach needs to be ensured at all times to avoid arousing hatred from developers who for example need to roll out a hotfix in the middle of the night and can not because of the validation or the pipeline in general malfunctioning. This, however, is only hypothetically seen as an issue and should usually not be a problem due to the simplicity of JSON schema validation and vast experience with CI/CD pipelines in today's commonly applied development environments.

Another mentioned issue was that pipeline integration requires additional effort which can lead to some teams trying to circumvent this. To that effect, adherence to the approach needs to be enforced so that all teams stick to it and do not implement their own solutions outside of a pipeline or possibly within a pipeline without the necessary validation step. Since organizations have departments in place however that are responsible for such matters, this should not be regarded as a critical issue.

All in all, it can be concluded that pipeline integration is deemed a necessary requirement for the proposed solution to function as intended for which no alternative could be identified by the practitioners (mean grade: 1.57).

4.2.4. Organizational Integration

Q: How do you rate the approach of shifting the documentation responsibility towards developer teams?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

F: **Benefits/Affirmations:**

- absolute necessity
- already a reality within the organization
- assign responsibility closest to the source

Limitations/Concerns:

- decentralized approach possibly inadequate
- motivation necessary

Mean Grade ($\varnothing Grade[n = 6]$): 2.0

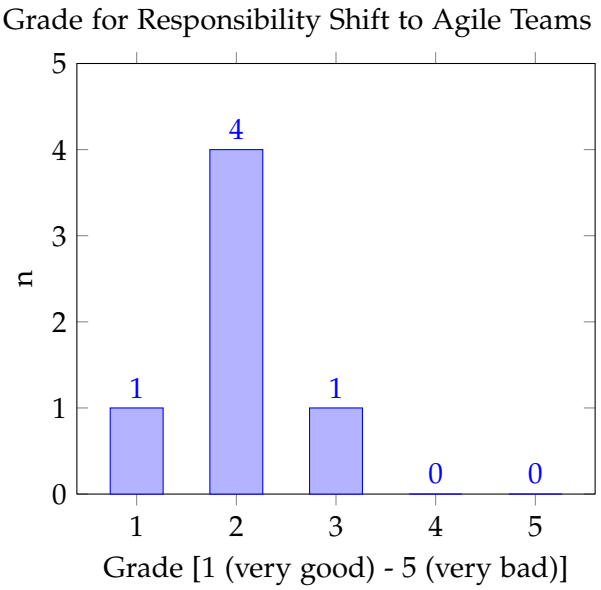


Figure 4.8.: Grade for Responsibility Shift to Agile Teams

A: This question tries to identify if shifting the responsibility of documentation to the agile teams is considered the right approach or if this should be handled differently.

The interviewees mostly agreed with this shift of responsibility as it was mentioned that this is already a reality within the organization of our industry partner. The documentation should be done by those who are closest to the source and therefore

the truth. They possess the necessary information to comprehensively document the implemented IT artifacts, which is reportedly not necessarily the case for EA architects that were not involved in the development process.

Some argued that documentation which is performed in a decentralized way by the responsible teams might be more complicated to keep consistent to each other since not every team might use the same notation and structure. Also, the teams need motivation to actually fulfill their responsibility for documentation, otherwise they might neglect it. This should however not be an issue as there are measures in place to enforce a standardized structure and adherence to it which were covered in previous questions.

Overall, shifting the documentation responsibility to the agile teams was perceived as a necessary measure to improve overall documentation quality and is already a reality in the context of our industry partner (mean grade: 2.0).

Q: Which persons do you need to involve in order to integrate the approach into the existing organization?

Which role do these persons hold?

Which persons do you see as drivers/supporters and which as blockers?

F: **Drivers:**

- architecture department
- top management
- senior management
- main department manager
- possibly everyone who might benefit from the solution

Neutral:

- agile team
- product owner
- operations management

Blockers:

- data protection team
- IT security
- possibly everyone who needs to adapt to the changes

A: This question aims at identifying which persons or roles need to be taken into consideration when integrating the solution approach into an organization so that the respective persons can be addressed appropriately depending on whether they are more likely to be in favor of the solution or more likely to be in conflict with it.

A variety of people and roles were mentioned that need to be involved. Architecture departments and management positions were identified as drivers since they either

profit most from the integration of the solution approach or do not necessarily need to take much action in actually implementing the solution besides approving and overseeing it.

Possible blockers would be persons and departments responsible for data protection and security related concerns. As the proposed solution approach accesses and processes possibly sensible information and on top of that grants insight into the data to different stakeholders, a variety of measures such as access management need to be put in place in order to ensure compliance with applicable laws and regulations. This represents additional effort and might limit the acceptance of the solution approach by the responsible people.

Tying in to the last statement, it was argued in general that the integration of the solution approach represents additional effort and therefore the most important aspect for determining whether a certain person would act as a driver or a blocker are the benefits achieved by implementing and using the solution and how well these benefits can be communicated to the respective people. This is also the reason why the agile teams and product owners but also the operations management team who were identified as the people responsible for the integration and therefore the bulk of the necessary implementation work, were described as neutral to the approach. Depending on the perceived benefits in relation to the necessary effort, they could either be in favor or against the solution approach.

All in all, it can be concluded that in general everybody who would potentially benefit from the solution would act as a driver and everybody that would foremost perceive additional effort for adapting to the changes would act as a blocker. This is why the solution approach needs to be communicated to the respective people in a way that convinces them of the benefits they would achieve when using the solution in order to gain broad approval for its integration.

4.2.5. Usability

Q: How do you rate the cost-benefit ratio of the solution approach in general?

F: **Benefits/Affirmations:**

- estimated costs relatively low
- estimated benefits tremendous

Limitations/Concerns:

- big bang integration difficult
- focus on mission critical elements
- benefits difficult to quantify
- costs differ depending on organization

A: This question aims at assessing the value proposition of the solution approach by contrasting the estimated costs with the expected benefits.

All interviewees estimate that the costs are relatively low. APM tools are already in place and therefore would not create additional cost. Creating and maintaining a JSON configuration file is also not perceived as a huge effort even for external contractors. Still, it is advised to keep the content of the configuration files as minimal as possible. No information should be added just for the purpose of adding additional information, every piece of information needs to add value.

Furthermore, it was argued that the integration needs to be conducted incrementally and not all at once (big bang). While this minimizes risks in general, it also requires multiple procedures for documentation to be used in parallel while integrating all microservices into the proposed solution approach, which could negatively impact its adoption.

One person argued that only mission critical microservices should be incorporated into the solution approach in order to save costs and because they are the sole elements that profit the organization most. While this brings the most important elements of the IT landscape into focus, it also eliminates the ability to provide a complete view of the enterprise architecture and therefore conflicts with one of the goals of the proposed solution, which is to provide a holistic approach.

While some interviewees considered the benefits to be difficult to quantify, the consensus was that the benefits are tremendous. One person put a figure on it by estimating that for a root-cause-analysis alone the value can reach six to seven figures.

In general, it was argued that costs and benefits strongly depend on the organization in question. In case of our industry partner, where APM tools are already in widespread usage and therefore do not represent additional costs, the perceived benefits clearly outweighed the overall cost estimations.

But even if monitoring tools would need to be integrated from scratch, it was stated that the automatically extracted runtime information in combination with the static information from the JSON configuration files represents a tremendous benefit which the organization can capitalize on when used correctly.

Overall, it was asserted that the solution approach represents an improvement regarding all the mentioned dimensions in comparison to the processes and solutions currently in use.

Q: How would you rate the improvement of the IT landscape documentation regarding the dimensions (data) completeness, actuality, consistency and reliability?

F: **Benefits/Affirmations:**

- improvements along all dimensions
- huge improvements regarding actuality and consistency

Limitations/Concerns:

- completeness not necessarily improved
- reliability not necessarily improved
- uncertainty regarding benefits of using proposed solution approach instead of APM tool

A: This questions aims at identifying to what extent various dimensions are improved after integrating the proposed solution approach.

The practitioners were in general agreement that all stated dimensions seem to improve through the usage of the solution approach.

Actuality and consistency are considered to profit most since data is collected automatically from a source that observes reality. Also, it was claimed that manual documentation, in the worst case, becomes outdated the moment it is created.

When it comes to actuality, however, there was also uncertainty whether the solution approach provides any benefit compared to directly using the APM tool in use, since it represents the primary source for the information and is therefore even more up to date. While the APM tool contains the most recent information, it does not associate that information with the information from the JSON configuration file and therefore the author argues that there is some benefit for using the proposed solution approach.

But in general, the goal of the solution approach is not to replace monitoring tools. There will still be use cases for which directly consulting the APM tool will be beneficial and preferable, but for others, especially regarding documentation purposes, the proposed solution approach should provide additional benefits.

On another note, some concerns were raised regarding completeness and reliability and mentioned that they do not necessarily experience improvements. Since it can not be

guaranteed that the solution discovers all relevant elements during a specific timeframe, no claim can be made that the extracted data is actually complete. (Another reason that was mentioned in this regard was that data from microservices and legacy systems that are not monitored is not included in the result and therefore it is incomplete but this again contrasts with the prerequisites described in subsection 2.2.2, which is why it shall only be mentioned for the sake of completeness here.)

In addition to not necessarily being complete, there is also some uncertainty as to whether the observed data is still relevant or if it is already outdated, which is why it was argued that the data in question is not necessarily reliable, one person even called it dangerous for that reason.

If the responsible teams adhere to the maintenance of the configuration files and the extraction and interpretation of runtime data is implemented correctly, then the provided information should be very reliable, and for critical decisions it is still advised to double check the information, but further research is required to assess how complete and reliable the collected data actually is in the long run and how the situation can be improved.

4.2.6. Visualizations

Business Landscape View

Q: What is your feedback for the Business Landscape view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- valuable content
- good overview, but no added value because of missing dependencies
- useful for reorganizations, otherwise limited usefulness
- uncertainty whether hierarchic nested grouping is ideal representation, suggested alternative: hierarchic tree

Mean Grade ($\varnothing Grade[n = 7]$): 2.86

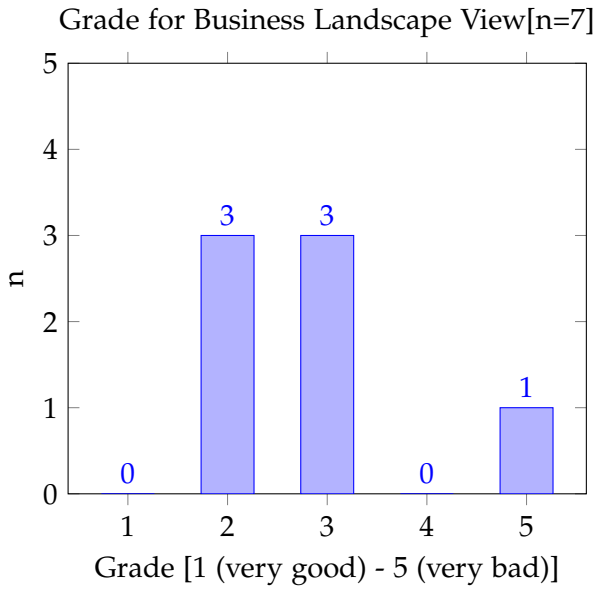


Figure 4.9.: Grade for Business Landscape View

A: This question aims at assessing the perceived usefulness of the Business Landscape view presented in Figure 3.7 and identifying suggestions for improvement.

The Business Landscape view received mostly mediocre feedback and ratings. While it contains valuable information, it was not perceived as very useful as a visualization. Its usefulness is reportedly limited to specific use cases such as providing an overview when a company goes through a reorganization, but even then does it not add any value to already existing visualizations. Since this type of data (mostly pertaining to the business layer) does not change that often, it does also not suffer from the

actuality problem as other more frequently changing data and therefore the manual documentation of such overviews is usually sufficient.

Furthermore, it was argued that the hierarchic nested grouping approach for this visualization does not provide any additional benefit in comparison to a hierarchic tree.

Some even preferred a purely tabular view of this kind of information and considered a visualization completely useless.

Overall, this view did not make a good impression on the EA practitioners, but at the same time it does no harm having it available for those specific use cases when it does come in handy (mean grade: 2.86). Still, the Business Landscape view should probably not be prioritized when further developing and extending the visualizations.

Application Landscape View

Q: What is your feedback for the Application Landscape view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- valuable for technical users
- too much information
- too many edges
- grouping of technologies not useful
- focus on individual systems necessary

Mean Grade ($\varnothing Grade[n = 7]$): 2.71

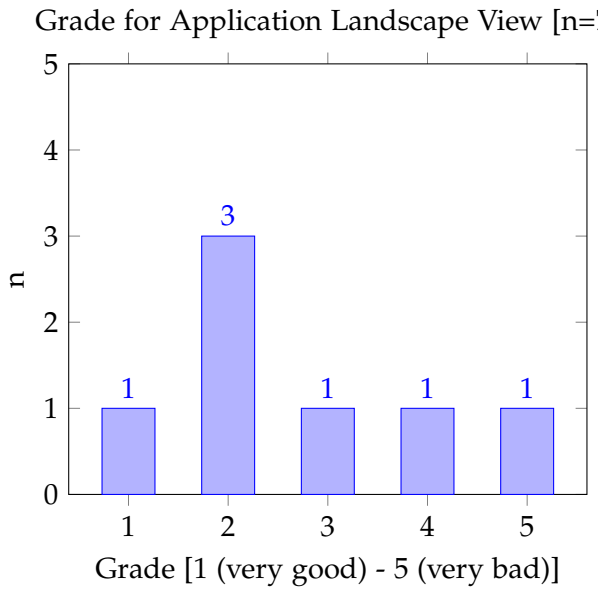


Figure 4.10.: Grade for Application Landscape View

A: This question aims at assessing the perceived usefulness of the Application Landscape view presented in Figure 3.8 and identifying suggestions for improvement.

The Application Landscape view was mostly criticized for its level of complexity. Since it is filtered by AppMon applications, depending on how many microservices are associated with the application, the resulting view can become quite large with too many edges and too much information in general. Also, grouping microservices by their technology and nodes by their operating systems was perceived as confusing and unnecessary by some practitioners.

The view was deemed useful for technical users who are responsible for deployments and more interested in the underlying IT infrastructure.

Otherwise, to provide additional value, it was argued that the view needs the ability to focus individual systems.

Overall, the view received mostly mediocre feedback and ratings (mean grade: 2.71) and needs to be improved during future development of the solution. More levels of abstraction need to be added and the displayed information needs to be further reduced in order to provide valuable information.

Table View

Q: What is your feedback for the Table view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- important information
- no added value
- more analysis functionality required

Mean Grade ($\varnothing Grade[n = 7]$): 2.57

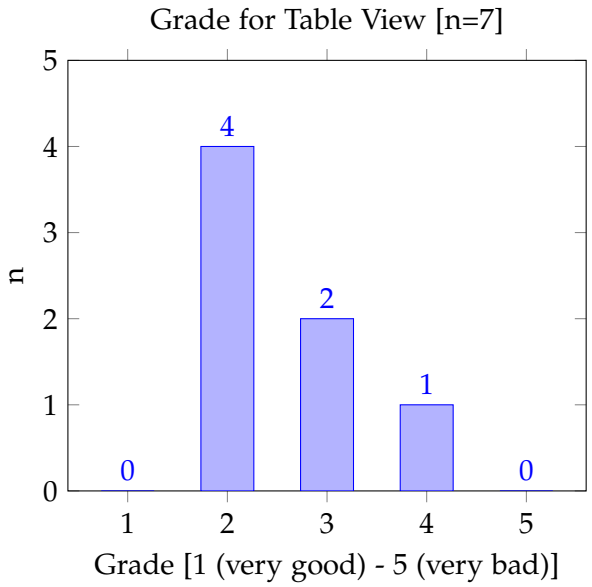


Figure 4.11.: Grade for Table View

A: This question aims at assessing the perceived usefulness of the Table view presented in Figure 3.9 and identifying suggestions for improvement.

The Table view was considered an important view for quickly accessing information when actual visualizations are not required but at the same time it does not provide any additional value compared to an ordinary tabular file format.

It was argued that more analysis functionality is required in order to be actually useful. One person mentioned extending the view to represent a service catalog including available interfaces, descriptions and responsible persons of contact in order to provide additional value and be eligible for a rating of 1 (very good), which is what should be taken into consideration when extending the view in the feature.

Otherwise, it received mostly mediocre ratings (mean grade: 2.57).

Communications View

Q: What is your feedback for the Communications view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- added value
- information about complexity
- interrelationships visible
- more metrics required
- highlighting of hotspots
- different levels of abstraction necessary

Mean Grade ($\varnothing Grade[n = 7]$): 1.71

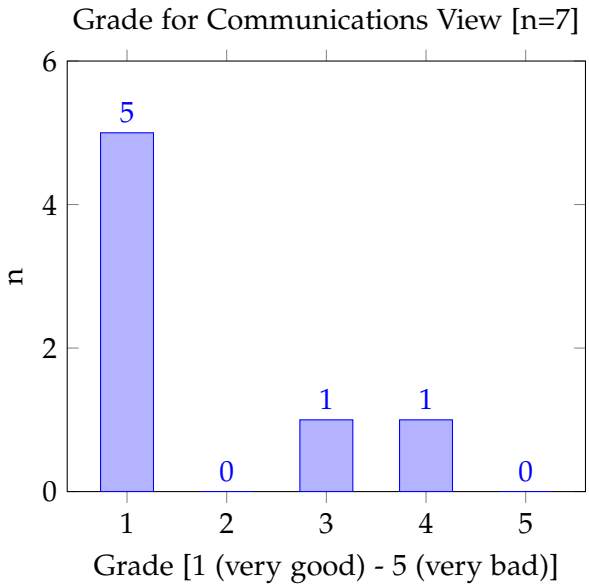


Figure 4.12.: Grade for Communications View

A: This question aims at assessing the perceived usefulness of the Communications view presented in Figure 3.10 and identifying suggestions for improvement.

The Communications view made mostly a good impression on the practitioners. It provides an overview of the IT landscape including interrelationships between individual elements which allows the respective users to gauge its complexity .

While this is perceived as added value, it was argued that more metrics should be included and hotspots should be highlighted, i.e. elements with many incoming and outgoing edges should be placed and visualized more prominently. Also, additional functionality is deemed necessary to drill the information up and down to the desired

level of aggregation. This is already supported by the backend, but was not implemented in the frontend due to time restrictions.

Overall, the Communications view received mostly good to very good ratings and was considered one of the most valuable visualizations (mean grade: 1.71).

Application Interaction View

Q: What is your feedback for the Application Interaction view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- exceptional additional value
- targeted interfaces visible
- removal of application collaborations suggested

Mean Grade ($\varnothing Grade[n = 6]$): 1.0

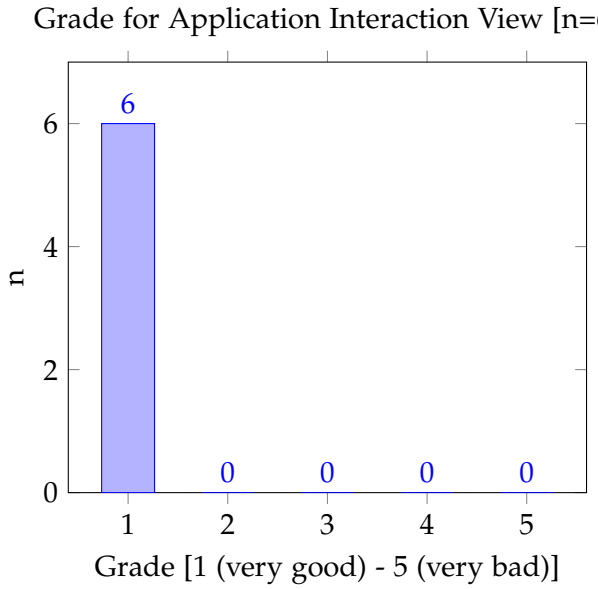


Figure 4.13.: Grade for Application Interaction View

A: This question aims at assessing the perceived usefulness of the Application Interaction view presented in Figure 3.11 and identifying suggestions for improvement.

The Application Interaction view made an exceptional impression on the EA practitioners. It was praised for providing information which reportedly no other tool provided in this manner by not only visualizing the interrelationships between elements but also exposing the targeted interfaces.

The possibility to select an individual request (PurePath) and view its way through the system including the called interfaces was perceived as one of the most important added values by the tool in general.

In order to further improve the view, it was suggested to remove the application collaboration from the visualization since it does not provide any additional information and therefore only unnecessarily complicates the view.

All in all, the Application Interaction view received a perfect rating with everybody giving it a 1 (mean grade: 1.0).

Huge further potential was identified by associating the extracted requests with business use cases and scenarios to create a link between these elements and be able to comprehend what happens on a technical level when a certain use case or scenario is triggered. This could be accomplished by adding this information to the JSON configuration file and should be addressed when extending the solution in the future.

Comparison View

Q: What is your feedback for the Comparison view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- useful functionality
- filter by deployment, instead of date
- future missing
- false positives

Mean Grade ($\varnothing Grade[n = 7]$): 1.86

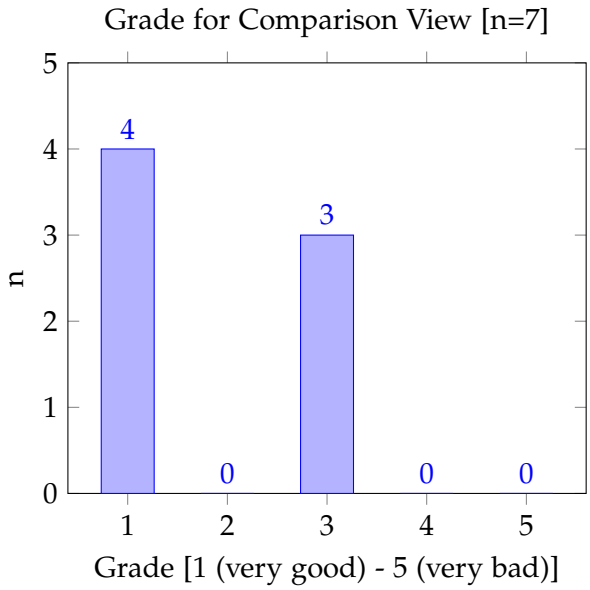


Figure 4.14.: Grade for Comparison View

A: This question aims at assessing the perceived usefulness of the Comparison view presented in Figure 3.12 and identifying suggestions for improvement.

Comparing different states within the Comparison view was generally considered useful. The implemented highlighting of added and removed elements was identified as a necessary feature. It was mentioned, however, that filtering states by date should be replaced or supplemented by the ability to filter by deployment since those are the triggers responsible for changes.

Furthermore, it was argued that future (planned) states need to be included in the comparison but this is difficult to implement since the primary focus of the tool is to extract and visualize runtime data which can obviously only contain information on the as-is landscape. This could be added, however, by extending the data model in order to

support planned states of elements which could be manually stored in the database and then retrieved to be compared against automatically generated as-is graphs. It would require a manual effort to model the planned states though.

Another concern that was raised was that of reliability. Since runtime data does not necessarily provide a complete data set as discussed in previous questions, the comparison view can contain false positives where differences are detected and visualized but in reality they only came to existence because the data was incomplete. It was argued that this leads to reduced reliability and can potentially even be dangerous. Also, it requires manual effort to double check if a difference is actually true or not which to some extent tarnishes the proposed value of this view in general.

Overall, the Comparison view received mostly good feedback and ratings (mean grade: 1.86), but multiple concerns were identified that need to be addressed in order to achieve the view's full potential.

GraphQL View

Q: What is your feedback for the GraphQL view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

F:
- useful for developers
- suggestion for direct visualization functionality

A: This question aims at assessing the perceived usefulness of the GraphQL view presented in Figure 3.13 and identifying suggestions for improvement.

Most interviewees had nothing to say about the GraphQL view besides that it is useful for developers to create their own queries. It was added to the evaluation for the sake of completeness, but the author refrained from asking for a rating in the end.

It was suggested, however, to add the ability to directly visualize the results from individual queries, which is probably easier said than done since queries can become quite complex, but the suggestion represents interesting food for thought for possible future extensions.

4.2.7. General Remarks & Feedback regarding the visualizations

Q: Do you have any general remarks or feedback regarding the visualizations?

F:
- different color schemes
- search / filter functionality
- export functionality
- more analysis features / KPIs
- more context required

A: This questions aims at collecting additional feedback for the visualizations in general which applies to multiple views and not just a particular one.

From the feedback to this question, multiple features could be identified that were requested time and time again. It was mentioned that the color scheme could be more pleasant to the eye and also more colors should be used in general to visualize the information in more useful ways. As explained before, in order to provide a generalized view of architecture models, the color scheme in use was based on ArchiMate which provides guidelines for color coding various elements. During the conduction of the interviews, however, it turned out that most interviewees were not familiar with ArchiMate and therefore, it might be advisable to discontinue the usage of its color scheme and switch to a different one or maybe even design a custom one.

Additionally, many interviewees asked for more search and filter capabilities to better limit the views to only contain the information that is deemed valuable and necessary for any given time. Furthermore, the provided information should not only be visualized but there should also be various export features available to extract the relevant information and import it in other tools, especially the Microsoft Office product family was mentioned in that regard.

Finally, the tool should offer more analysis features included in the visualizations which highlight KPIs and generate reports.

As a last note, it was mentioned that the visualizations do not provide enough context to be understood intuitively. Most visualizations only make sense to people who created or have worked with them, or people who are familiar with the underlying data set, but otherwise they do not contain enough meta information to be understood by ordinary users.

4.2.8. Use Cases

Q: Do you recognize further use cases besides EA documentation, which can be assisted by the usage of the proposed solution approach?

F:
- guideline compliance
- anomaly detection
- failure / fault analysis
- incident management
- cloud migration analysis
- comparison of states

A: This questions tries to identify further use cases for which the proposed solution approach can be utilized.

The mentioned use cases can be found above and do not require any further assessment. It shall be noted, however, that it was argued that all the identified use cases can be

assisted by a good documentation in general, which is why documentation is perceived as the most valuable use case the solution approach contributes to.

4.2.9. General Remarks & Feedback regarding the overall solution approach

Q: Do you have any general remarks or feedback regarding the overall solution approach?

F:
- integration of monoliths and legacy systems
- differentiation from APM tools

A: This question is the last one and aims at gathering final feedback for the overall solution approach that did not fit anywhere else.

Multiple practitioners expressed uncertainty as to how the presented solution approach can be integrated into organizations that already possess a variety of monoliths and legacy systems next to the increasing amount of microservices. Since the presented solution approach is primarily scoped for the usage of microservices, this is indeed an issue and further research is required to determine if and how this integration could be accomplished.

On another note, one practitioner suggested that the proposed solution approach needs to stronger differentiate itself from common APM tools and better work out its unique value propositions. While the presented solution approach does heavily rely on an APM tool to provide its functionality, it offers multiple features that common APM tools do not. It provides a standardized query language to retrieve data, it enables the historization of extracted data, and it assists the documentation process in an automated manner, to name a few. Another thing that might not have been as apparent caused by the prototypical implementation, is that in the long run it is planned to support a multitude of different APM tools and provide unified access to their collective data which is mapped to one standardized data model.

4.2.10. Conclusion of the qualitative analysis

Taking into consideration all the findings of the qualitative analysis, one can assert that the proposed solution approach seems promising.

Many concerns were raised and discussed that need to be addressed in future research and when extending the IT artifact. Other than that, the overall solution approach was perceived as useful and from a technical as well as from an organizational perspective feasible in its integration. The estimated costs are reportedly outweighed by the expected benefits and therefore, it can be concluded that all in all, the proposed solution approach is relevant and should definitely be further developed.

4.3. Before-and-After Analysis

In this section, the state of the implementation environment prior to the implementation of the proposed solution approach will be compared with its state after the implementation. Differences will be presented and discussed.

4.3.1. Requirements Analysis - Revisited

Table 3.1 listed requirements identified prior to the implementation that should be fulfilled by the solution approach. In the following, these requirements are checked whether they were indeed fulfilled or not and the results are presented in Table 4.2.

Rank	Requirement	Fulfilled
1	Data flow and dependencies between applications	✓
2	Interfaces / APIs	✓
3	Mapping and associations within application layer	✓
4	Application Components (logical unit)	✓
5	Communication technology (protocols)	✓
6	Business Processes	✗
7	Mapping and associations within technology layer	✓
8	Physical IT resources	✓
9	Mapping and associations within business layer	(✓)
10	Use Cases / Scenarios	✗

Table 4.2.: Requirements Analysis - Fulfillment

As can be seen in the table, 7 of 10 requirements could be fulfilled. All of these fulfilled requirements correspond to the application and technology layer. It comes as no surprise that the three not fulfilled requirements pertain to the business layer and could therefore not be fulfilled because runtime data does not cover elements from this layer. Note that the ninth requirement shows a checkmark in parentheses since it was technically fulfilled but the information was matched through looking up the information in the manual documentation and was not identified automatically, which was actually the purpose of fulfilling these requirements.

The not fulfilled requirements of the business layer could semi-automatically be fulfilled by the implementation of the JSON configuration files. Despite that, the requirements analysis comparison still reveals satisfying results since the top three requirements, which were assigned the same score and were considered the most important, could be fulfilled to full extent.

4.3.2. Documentation Process - Comparison

Before moving on to the conclusion of this thesis, it is important to assess the possible improvements regarding the EA documentation which can actually be achieved by implementing the solution approach within the environment of our industry partner.

Figure 4.15 depicts how the elements from our data model are documented before and after the implementation of the proposed solution approach. The red outlines indicate that the documentation process is conducted manually or semi-automatically whereas the green lines indicate that the elements are documented automatically. Grey elements are not documented at all.

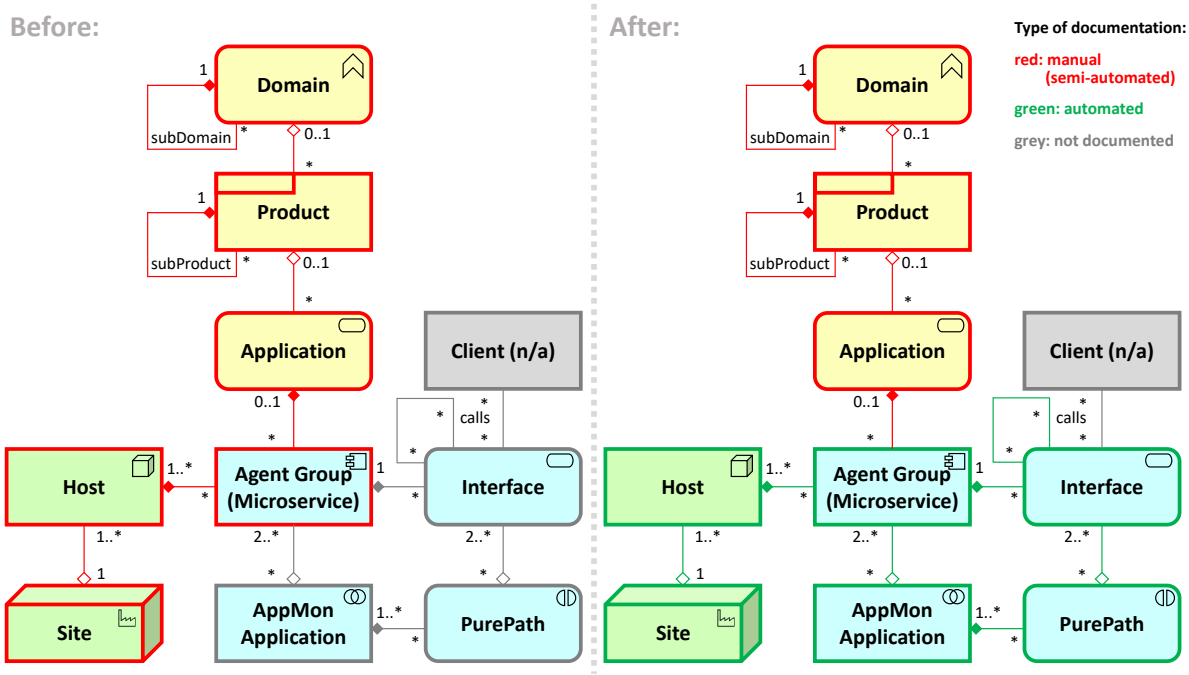


Figure 4.15.: Before-and-After Documentation

Before the implementation (left) the entire hierarchy from domains to sites (facilities) was documented completely manually whereas elements like interfaces and requests (PurePaths) were not documented at all. Furthermore, microservices are considered too small of a unit to justify documenting them in a centralized repository, therefore the manually documented information is spread across multiple repositories, which makes it difficult to locate and access the required data.

The implementation of the proposed solution approach (right) shows that all elements from the application and the technology layer can be documented automatically and especially the requests and interfaces which were not documented at all before provide crucial added value. Additionally, all information is stored centrally in one repository where it can easily

be retrieved from. The elements of the business layer still have to be documented manually, but this could be semi-automated by implementing the JSON configuration files.

Note that clients could also be documented automatically by using a different APM tool that offers the capability to monitor such elements, but since this is not the case for AppMon this element remains greyed out in the figure and could alternatively be documented manually (or added to the JSON configuration files).

5. Conclusion

This chapter will summarize the findings of the previous chapters, conclusively answer the stated research questions and give an outlook on possible future research.

5.1. Summary

In order to facilitate the practice of Enterprise Architecture Documentation (EAD), which faces various challenges such as widespread usage of manual documentation that is costly and time-consuming and results in low quality data to base decisions on, this thesis built upon a novel approach called *Microlyze* that is being developed at the SEBIS chair of the TUM and that combines dynamic information from runtime data extracted from an Application Performance Monitoring (APM) tool with static information contained in JSON configuration files to reconstruct the EA landscape.

The solution approach was implemented and tested in a real world environment situated in the automotive industry. A thorough quantitative and qualitative analysis were conducted, comprising multiple expert interviews with EA practitioners from our industry partner. The implemented IT artifact was thereby evaluated and its benefits and limitations critically assessed. The findings described in full detail during the previous chapters will be used to answer the stated research questions hereinafter.

5.2. Findings

5.2.1. RQ1: Runtime Data Discovery

Part of this thesis was the identification of IT artifacts and their communication relationships that can be discovered and extracted from runtime data. The proposed solution approach managed to extract fully automated all relevant IT artifacts pertaining to the application layer and the technology layer as defined by ArchiMate. This includes communication and dependency relationships as well as finegrained information such as called interfaces (APIs). Business layer elements, however, could not be extracted automatically due to not being directly monitored. This information needs to be included from other sources such as JSON configuration files or manual documentations.

5.2.2. RQ2: Requirements

A variety of requirements have been identified during the implementation and evaluation of the solution approach that need to be fulfilled in order for the proposed solution approach to work as intended.

- **Accessible APIs**

  One of the most important requirements that needs to be fulfilled in order to automatically extract relevant information is the availability of one or more accessible APIs that provide the necessary information. The implementation of the proposed solution approach almost failed due to the APM tool in use within the implementation environment not exposing an useful interface from which to query and retrieve data. This also heavily impacts the performance of the implemented solution. Since data needs to be queried and processed accordingly, the whole process becomes much easier when the data is already retrieved in an immediately usable format. It can be concluded that an available API is all the better, the more finegrained it allows queries to be so that data can be queried in exactly the amount necessary and does not need to be heavily post-processed to extract relevant information.

- **Naming Convention (Mapping)**

  In order to correctly identify architecture elements and their interrelationships, it is of utmost importance to be able to map the information accordingly. Different APM tools use different data models and notations which do not necessarily coincide with the notation used within an organization. Sometimes the naming of elements does not coincide even within the APM tool's own data model as seen with AppMon where different names exist to denote the same agent. This poses a huge challenge that needs to be overcome in order to associate information of the respective elements correctly to each other. Ideally, elements are named unambiguously and consistently through all stages of the data flow. The terminology used within the APM tool (source) should distinctly match the terminology used within the organization (target) and therefore also the one used in the proposed solution approach (middleware). At the very least, there should be clear rules how to map elements to each other should they not follow a strict naming convention.

- **Domain Knowledge**

  One issue that was encountered while implementing the proposed solution approach was that no one could fully explain all the data and information that could be seen within the APM tool. As a consequence, it was difficult to identify which information needed to be extracted in the first place and how this information could be mapped correctly to logical entities. Therefore it is claimed that it is absolutely necessary to know beforehand which information is relevant and how it can be retrieved and mapped before implementing the solution approach in order for it to actually work as intended afterwards.

- **Holistic Approach**

  In order to maximize the perceived value of extracting EA information from runtime data, all relevant microservices need to be included in the monitoring environment, otherwise the achieved data coverage is incomplete by design.

- **Adherence to Regulations**

  Taking security and data protection regulations into consideration is crucial to the successful execution of the proposed solution approach. Since failing to adhere to important regulations might prevent the solution from being approved for productive environments. This constitutes more of an organizational requirement as a technical one.

5.2.3. RQ3: Benefits & Limitations

Part of this thesis was to identify and assess benefits and limitations of the implemented solution approach. This was done meticulously throughout the thesis but the most important findings shall be summarized once again hereinafter.

Benefits

- **Depiction of Reality**

  One of the most important perceived benefits is that runtime data represents a genuine depiction of reality. The information shows exactly what happened and is therefore considered a source of truth. Since the data can be queried in real-time the information is current and consistent.

- **Automation**

  The proposed solution approach is able to automatically gather relevant information while not requiring many resources for its execution or any manual input once implemented. The data can be collected en passant, which realizes an enormous time-saving potential as it eliminates any manual effort.

- **High Coverage & Accuracy**

  As seen in the quantitative analysis, the solution approach showed promising results when it comes to coverage and accuracy of identified elements. While not being perfect, it was still considered to be better than all manual documentation processes that are currently being used.

- **Model-based Approach**

  Since the proposed solution approach represents a model-based approach, i.e. data is stored in an universal data format that can be processed to match any requirements, it provides a high degree of flexibility to be used for many applicable use cases.

- **Historization**

  Storing extracted information centrally in a database enables the solution approach to provide information from far into the past as it is not limited by the storage restrictions of common APM tools.

- **Unique Value Proposition**

  As reported by EA practitioners, the solution approach offers unique additional value that no other tool can provide with its ability to display individual requests and their way through the monitored system.

Limitations

- **Application Performance Monitoring Tool**

  Referring back to all the limitations described in subsection 3.3.3 that are specific to the APM tool in use at our industry partner, which served as a basis for the prototypical implementation of this thesis, it comes as no surprise that the APM tool in use is a crucial component for the successful operation of the proposed solution approach and can be considered its single biggest limitation in general. The reasoning behind this is that since the proposed solution approach strongly depends on extracting data provided by the APM tool in use, it is directly affected by the capabilities and limitations of said APM tool.

- **Lack of Explanation**

  While the proposed solution approach shows promising results in reconstructing the as-is landscape including dependency relationships from runtime data, it is not capable of providing an explanation as to why a dependency relationship exists, it can only show that the relationship exists. A complete understanding of the underlying Enterprise Architecture is therefore not possible and the approach of reconstructing the IT landscape from runtime data can be considered insufficient. Extending the proposed solution approach with the JSON configuration files explained in section 2.2 could resolve this, but the configuration files could not be implemented during the conduction of this thesis in the implementation environment. This approach, however, would require the people responsible for the creation and maintenance of said configuration files to fully know all the details of their IT artifacts including the reasons why these artifacts communicate with certain other artifacts, which is not necessarily always the case in a heterogeneous business environment where agile teams independently develop and integrate their microservices and applications into an existing IT landscape. Even if that was the case, there would still be no explanation for communication between microservices that is happening out of the ordinary. The APM tool would detect it, but even the JSON configuration file would not be able to explain why it happened, since the communication occurred unexpectedly (most likely due to some error) and it is impossible to provide an explanation beforehand to something that is not anticipated to happen.

- **Focus on As-Is Landscape**

  Enterprise Architecture Management does not concern itself solely with the as-is landscape but also takes planned (future) states into consideration in order to derive plans for realizing the transformations necessary to get there. The proposed solution approach in its current form does not support planned states since runtime data can only provide information on what is being observed and obviously it is impossible for APM tools to observe the future. Planned states could be integrated into the JSON configuration files but validating them within the CI/CD pipeline appears to be a complex undertaking for which further research is required to evaluate its feasibility.

- **Reliability**

  Since it was shown that runtime data does not necessarily provide complete information on the underlying EA landscape it can also not be considered reliable without a doubt. This can potentially be even dangerous when unreliable data is used to base decisions upon which is why further measures need to be taken to validate the extracted information and ensure its reliability.

5.2.4. Final Assessment

All in all, combining the findings from the quantitative analysis, the qualitative analysis and the before-and-after analysis presented in chapter 4, it can be concluded that the proposed solution approach is highly valuable as it shows promising results in reducing the manual effort necessary for documentation. While some concerns could be identified, the achieved benefits still outweigh the estimated costs as reported by the EA practitioners. Therefore, further research should be conducted in order to address these concerns and further optimize the proposed solution approach so that one day it might achieve its full potential and possibly be used in productive environments.

5.3. Outlook

While this thesis presented a successful implementation of the proposed solution approach, it also revealed multiple relevant concerns that need to be addressed in future research.

- **Extension of the proposed solution approach**

  Despite the successful implementation of the prototype, there are still many features and improvements in general that can be added to the existing tool which can build the basis for future research endeavors where the solution approach is extended and evaluated anew, possibly in different sectors of the industry including new requirements.

  Also, the JSON configuration files, which are part of the overall solution apporach could not be implemented and were only simulated for evaluation purposes. This could be substance of another thesis where distributed tracing and JSON configuration files are implemented and evaluated collectively.

- **Integration into existing (heterogeneous) EA landscapes**

  One issue that demands further attention is the question of how to integrate the presented solution approach into organizations that already utilize a multitude of EAM tools and repositories and that consist of more than just microservices. Monoliths and legacy systems are not covered by the solution approach but are still widespread across environments of today's industry.

  Further research is required as to how these systems can be incorporated into the approach or how they can be operated effectively in parallel.

- **Integration of multiple APM tools**

  The implementation of the solution approach at our industry partner revealed that different departments use different APM tools to monitor their microservices. Further research is needed on how these APM tools can be utilized to extract data and acquire information that transcends the borders of the individual tools and provides a holistic view of the entire landscape.

  One possible solution could be the implementation of OpenTelemetry[1], an open standard for distributed tracing that could facilitate the extraction of runtime data across different tools. Dynatrace already announced its support of the standard in the future.[2]

[1] https://opentelemetry.io/

[2] dynatrace.com/news/blog/dynatrace-joins-the-opentelemetry-project/

A. Appendix

A.1. Interview Questionnaire

The following questionnaire was used to evaluate the proposed solution approach.

Q1: How do you rate the approach of extracting architecture information from runtime data in order to assist the IT landscape documentation?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

Q2: How do you rate the approach of extracting architecture information from runtime data in order to assist the IT landscape documentation?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

Q3: How do you rate the approach of maintaining further relationship information within JSON configuration files?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

Q4: How do you rate the approach of ensuring the maintenance of the configuration files through JSON Schema validation?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

Q5: To what extent does the solution approach contribute in general to improving the EA documentation?

Q6: What barriers do you recognize regarding the technical integration of the approach, especially with respect to the technical requirements that need to be met?

Q7: To what extent do you perceive the integration of the approach into a CI/CD pipeline as useful?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

Q8: How do you rate the approach of shifting the documentation responsibility towards developer teams?

What advantages and disadvantages do you see?

How do you rate the approach on a scale from 1 (very good) to 5 (very bad)?

Q9: Which persons do you need to involve in order to integrate the approach into the existing organization?

Which role do these persons hold?

Which persons do you see as drivers/supporters and which as blockers?

Q10: How do you rate the cost-benefit ratio of the solution approach in general?

Q11: How would you rate the improvement of the IT landscape documentation regarding the dimensions (data) completeness, actuality, consistency and reliability?

Q12: What is your feedback for the Business Landscape view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q13: What is your feedback for the Application Landscape view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q14: What is your feedback for the Table view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q15: What is your feedback for the Communications view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q16: What is your feedback for the Application Interaction view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q17: What is your feedback for the Comparison view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q18: What is your feedback for the GraphQL view and and how do you rate the visualization on a scale from 1 (very good) to 5 (very bad)?

Q19: Do you have any general remarks or feedback regarding the visualizations?

Q20: Do you recognize further use cases besides EA documentation, which can be assisted by the usage of the proposed solution approach?

Q21: Do you have any general remarks or feedback regarding the overall solution approach?

A.2. Transcripts

Please note: The interviews were conducted in German. To avoid falsification of the actual statements, all interviews were transcribed in their original language.

A.2.1. Transcript 1

Q: Wie lange arbeiten Sie im EA-Bereich und was gehört zu Ihren täglichen Aufgaben?

A: In dem Architekturbereich bin ich jetzt seit 5 Jahren und mein aktuelles Aufgabenfeld ist dort im Wesentlichen das Thema Produktportfoliomanagement, wo wir bei uns im Unternehmen ja den Produktkatalog oder alles nach IT-Produkten strukturieren, was dann aber auch die Basis für sagen wir mal auch die technische Architektur ist, weil nach diesem Produktkatalog strukturieren wir alles. Das ist so momentan meine Haupttätigkeit.

Q: Inwiefern akzeptieren Sie die genannte Problembeschreibung? Sind Sie anderer Meinung?

A: Die kann ich so voll unterstützen. Die ist definitiv so, weil – vielleicht um ein bisschen auszuweiten – im Zweifelsfall ist ja immer alles unter Zeit- und auch Budgetanspannungen und dann wird so etwas wie Dokumentation, im Zweifelsfall fällt das dann immer hinten runter, weil der Code halt wichtiger ist, dass er fertig wird, bevor noch groß dokumentiert wird und es ist schon etwas wo wir sehr damit kämpfen.

Q: Haben Sie Microservice-basierte Architekturen im Einsatz?

A: Ja.

Q: Haben Sie bereits einen Ansatz zur automatisierten EA-Dokumentation entwickelt?

A: Wir nutzen eine Jenkins-Pipeline, womit wir dann auch Informationen automatisiert auslesen können. Wir haben da auch so Programmier-Guidelines erstellt, die wir eigentlich auch unseren Dienstleistern immer mitgeben, was sie zu dokumentieren haben, damit wir das eben automatisiert auch auslesen können, was aber eben aus den genannten Gründen auch oftmals leider nicht getan wird, aber eigentlich geben wir es vor, was ja dann die Basis dafür ist, aber wie gesagt passiert dann leider oftmals dann doch nicht.

Q: Wie bewerten Sie den Ansatz, Architekturinformationen aus Laufzeitdaten zu extrahieren, um somit die IT-Landschaftsdokumentation zu unterstützen? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A: Ich würde dem Ansatz eine 1 geben, weil es aus meiner Sicht eigentlich der einzig sinnvolle Weg ist, weil manuell gepflegte Informationen, haben wir in der Vergangenheit gesehen, das funktioniert nicht aus den Gründen, was wir vorhin schon hatten und man muss dazu hinkommen, zum einen, Entwickler dazu zu bewegen, diese Informationen zu pflegen und wenn es eben nur über den Zwang ist, dass sie dann nur durch die Pipeline kommen und dann möglichst viele Informationen automatisiert auslesbar zu bekommen. Ich sehe eigentlich in der Zukunft keinen Weg daran vorbei. Es ist am Anfang erstmal eine Umstellung, aber die muss halt gemacht werden, das ist immer so bei einer Veränderung, aber sonst fallen mir aktuell keine Nachteile ein.

Q: Wie bewerten Sie den Ansatz, weitere Beziehungsinformationen in Konfigurationsdateien zu verwalten? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A: Immer wenn wir diese Meta-Informationen erweitern, sie müssen halt definitiv sauber begründet sein, weil man sollte aus meiner Sicht auch den Aufwand so gering wie möglich halten, weil wenn ich diese Meta-Informationen jetzt unnötig aufbläh, fördert das natürlich wieder das Interesse das Ganze zu umgehen und an der Pipeline vorbeizugehen, um das zu vermeiden. Es muss, alles was wir dort pflegen wollen, sauber begründet sein unn auch wirklich einen Mehrwert stiften, aber ich würde versuchen, das so gering wie möglich zu halten. Da fällt mir doch noch ein Nachteil ein: Wenn wir eben diesen Zwang einführen dieser Dokumentation, wir müssen halt dafür sorgen, dass es keine Umwege um diese Pipeline gibt, sodass einzelne Produkte wieder auf die Idee kommen eine eigene Pipeline irgendwie sich zu bauen, um diesem Zwang zu entgehen, aber das ist eine Governance-Aufgabe das wirklich so konsequent einzufordern. Für die JSON-Configfile würde ich auch eine 1 geben, weil ohne das funktioniert es halt nicht.

Q: Wie bewerten Sie den Ansatz, die Pflege der Konfigurationsdatei mit Hilfe von JSON Schema sicherzustellen? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A: Was wäre die Alterative, ich sehe grad keine Alternative. Das ist ja nur das Format. In irgendeiner Form muss ich es ja vorgeben, also ich glaube JSON-Schema ist da jetzt momentan state-of-the-art. Wenn es dann irgendwann mal in der Zukunft ein neues Format gibt, was besser geeignet ist, ja, dann sollte man vielleicht das nutzen, aber momentan sehe ich das JSON-Schema da definitiv angebracht.

Q: Inwiefern trägt der Lösungsansatz im Allgemeinen zur Verbesserung der EA Dokumentation bei?

A: Trägt definitiv dazu bei, weil alles, was ich automatisiert auslesen kann, ist deutlich besser als das, was wir heute haben.

Q: Welche Barrieren erkennen Sie bei der technischen Integration des Ansatzes? Insbesondere in Anbetracht der technischen Voraussetzungen, die erfüllt werden müssen.

A: Das ist für mich halt eben, dass heute schon bei uns und selbst in kleineren Bereichen unterschiedliche Monitoring-Tools einfach heute im Einsatz sind, vielleicht auch berechtigt unterschiedliche Monitoring-Tools im Einsatz sind, weil sie unterschiedliche Anforderungen besser bedienen können. Da ist halt die Frage, wenn ich dann Bereiche zwinge, auf ein definiertes Monitoring-Tool zu gehen, ob ich dort halt Widerstände habe oder dadurch Kosten verursache. Dann ist halt bei uns, wir sind momentan in einer Transformation zwischen einer On-Premise-Welt und einer Cloud-Welt, also wie das dann zusammen spielt. In der Cloud habe ich ja auch wieder andere Möglichkeiten, auch Monitoring-Tool-technisch, wie das dann zusammen funktionieren kann oder ob

es dann wirklich ein Ansatz ist, der dann erst zu 100% trägt, wenn wir zu 100% auch in der Cloud sind, das weiß ich heute einfach noch nicht. Ansonten JSON ist ja ein gesetzter Standard und sollte eigentlich kein Hindernis sein. Und zum letzten Punkt „Delegation Richtung agiler Teams“, das hat jetzt damit eigentlich nichts zu tun, das ist grundsätzlich so, dass die agilen Teams jetzt mehr Verantwortung bekommen und diese auch nehmen müssen, aber das ist ja im Zuge der Umstellung auf ein agiles Arbeiten. Das wäre jetzt sagen wir mal eine Grundvoraussetzung.

Q: Inwiefern halten Sie die Integration des Ansatzes in eine CI/CD Pipeline für sinnvoll? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A: Ich kriege immer wieder die Diskussionen: Kann es eine Pipeline für alle geben? Da gibt es halt auch immer wieder Argumente dagegen, aber wir kommen an der Pipeline aus meiner Sicht auch nicht vorbei. Das ist für mich auch dann zwingende Voraussetzung. Pipeline-Integration würde ich auch eine 1 geben.

Q: Wie bewerten Sie den Ansatz, die Dokumentationspflicht in Richtung Entwicklerteams zu verlagern? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A: Die Verantwortung muss dorthin.

Q: Welche Personen müssen Sie involvieren, um den Ansatz in die bestehende Organisation zu integrieren? Welche Rolle spielen die Personen? Welche Personen sehen Sie als Treiber/Unterstützer und welche als Blockierer?

A: Als Treiber würde ich definitiv, also von der Organisation her gesehen, wir haben jetzt eine Gruppe, die sich um Operations Management kümmert, die würde ich dort definitiv als Treiber sehen, weil die für mich solche Standards eben definieren müssen, auch bezüglich welches Monitoring-Tool wir einsetzen wollen. Dann kommt jetzt bald eine Architekturgruppe, die dort auch massiv solche Standards dann definieren und auch treiben müssen. Widerstände wird es wahrscheinlich immer irgendwo geben, auch auf Produktebene dann, weil es eben auch eine gewisse Veränderung einfach bedeutet oder auch Aufwände verursacht und wird es auch auf Produktebene erstmal Widerstände geben, weil dadurch generiere ich nicht eine neue Business-Funktionalität, sondern das ist halt zum Teil wahrscheinlich erstmal technische Schulden aufarbeiten und da wird es auch erstmal Widerstände geben.

Q: Wie bewerten Sie im Allgemeinen das Kosten-Nutzen-Verhältnis des Ansatzes?

A: Ich würde halt schauen, dass in den Configfiles wirklich nur das Nötigste dokumentiert werden muss, um den Aufwand so gering wie möglich für die Entwicklerteams oder den Product Owner zu halten, dass das nicht immer eine ewige Dokumentation ist nur des Dokumentationswillens. Da muss halt dann definitiv auch eben der Nutzen überwiegen. Deswegen muss für mich alles was dort dokumentiert werden muss,

muss transparent aufgezeigt werden, welchen Nutzen haben wir davon. In diesem Fall überwiegt der Nutzen definitiv, weil wir einfach eine bessere Dokumentation haben, Abhängigkeiten zwischen Produkten, Wirkketten, alles das wirklich sauber und aktuell, also v.a. aktuell darstellen können.

Q: Wie würden Sie die Verbesserung der IT-Landschaftsdokumentation bewerten innerhalb der Dimensionen (Daten-) Vollständigkeit, Aktualität, Konsistenz und Zuverlässigkeit?

A: Da sehe ich eine Verbesserung in allen genannten Dimensionen. Die manuelle Dokumentation ist im Zweifelsfall kurz nachdem sie erstellt wurde veraltet und wenn wir hier deutlich aktuellere Daten haben, sehe ich da definitiv einen Nutzen.

Q: Wie bewerten Sie die einzelnen Darstellungen auf einer Skala von 1-5?

A: Business Landscape View:

Das ist für mich auch primär eine Visualisierung der Architektur, um gewisse Abhängigkeiten herauszufinden, aber im Fehlerfall muss ich eben dann in das Monitoring-Tool schauen. Ich würde speziell dieser Visualisierung eine 2 geben, sie gibt schon mal einen sehr guten Überblick aber ich sehe den wirklichen Mehrwert erst in weiterführenden Visualisierungen, wo ich Abhängigkeiten habe, wo ich eben nicht nur sehe, dass eine entsprechende Applikation aufgerufen wird, sondern eben z.B. in einer Wirkkette, wer wen aufruft und ob da irgendwelche Zirkelbezüge oder Mehrfachaufrufe existieren, das ist dann wirklich der architekturelle Mehrwert, wo ich dann auf irgendwelche Missstände aufmerksam werde, um auch die Architektur zu optimieren.

Application Landscape View:

Für mich persönlich ist diese Darstellung nicht so relevant, weil ich mich eher mit der fachlichen Ebene befasse, aber für die Kollegen in der technischen Architektur und im Operations Management ist die glaube ich sehr hilfreich, da würde ich eine 1 geben. Wobei ich unter dem Begriff Application Landscape hätte ich das nicht erwartet. Im EAM haben wir die Business-Architektur, die Applikationsarchitektur, die Informations- und die technische Architektur. Das wäre jetzt für mich die technische Architektur. Unter Applikationsarchitektur hätte ich dann eher Abhängigkeiten unter, jetzt in dem Fall, Business-Applikationen erwartet. Aber das ist nur eine Naming-Angelegenheit, sonst passt.

Table View:

Die rein tabellarische Darstellung, die habe ich im Repository auch, dort haben wir den Master für den Produktkatalog, dort haben wir eigentlich auch alles dokumentiert, bis auf die Microservices. Würde ich jetzt mal eine 3 geben, es ist vielleicht gut zu haben, aber da sehe ich jetzt nicht so den massiven Mehrwert.

Communications – Business Services:

Das ist wirklich die interessante Information, wie ist die Informationsbebauung geschnitten, habe ich irgendwelche Zirkelbezüge oder habe ich irgendwelche Ketten, dass die

unterschiedlich aufrufen, aber Daten wieder weiterverarbeiten und damit eigentlich nicht die Originaldaten mehr haben und solche Informationen, ja. Kann man das hoch aggregieren, z.B. auf Produkte, sodass ich Abhängigkeiten zwischen Produkten, also dass ich sehe, folgendes Produkt wird von x Produkten aufgerufen und dann kann ich reindrillen welche Applikationen oder Microservices das im Einzelnen sind? Das fände ich z.B. noch hilfreich, dass man das wirklich auch auf Domäne oder Produkte hoch aggregieren kann.

Communications – Application Components:

Die ist glaub ich sehr hilfreich, die ist dann halt schon wirklich im Detail, aber konkret im Bsp. „Web-API“: Wenn ich jetzt sagen will, ich will die Web-API abschalten, dann möchte ich ja z.B. herausfinden, welche Schnittstellen wurden erkannt, mit wem muss ich jetzt alles reden oder wer wäre betroffen, wenn ich die Web-API abschalten will. Und dann wäre es halt hilfreich, wenn ich sehe, ich habe drei Produkte, die auf die Web-API zugreifen, dann kann ich mit den Product Owners darüber sprechen und dann ins Detail gehen. Kommunikationsansicht würde ich in beiden Fällen eine 1 geben.

Application Interactions View:

Diese Ansicht halte ich für besonders interessant. Da wäre noch interessant, das hoch zu aggregieren, einerseits auf unsere IT-Produkte, aber auch auf Endkundenprodukte, wie z.B. ein RTTI, dass man eben sieht welche Aufrufe müssen im gesamten Backend bei uns erfolgen, um eben diesen Dienst RTTI für Endkunden erbringen zu können und dann die Wirkkette darzustellen, um dann eben im Fehlerfall schneller suchen zu können. Das sehe ich auch als sehr hilfreich, dem würde ich auch eine 1 geben. Ist glaub ich auch einer der größten Mehrwerte, die wir dadurch erzielen.

Comparison View:

Ist definitiv auch interessant, würde ich auch eine 1 geben. Ich würde vielleicht nicht nach Datum gehen, sondern nach Deployment, weil bei einem Deployment ändert sich ja etwas. So müsste ich halt wissen, wann sich etwas geändert hat. Im Zweifelsfall ist so eine Ansicht ja relevant, wenn ich z.B. im Feld auf einen Fehler stoße oder wenn Kunden Fehler melden, da würde mich vielleicht interessieren, wer hat in diesem Zeitraum neu deployed und was hat sich dann verändert. Dann kann man es natürlich über Datum machen oder ob man es dann nach Deployment macht, aber dann müsste man ja natürlich wissen, wer was deployed hat wann. Dass man eben auch über die Pipeline abfrägt, wer hat dann in diesem Zeitraum vielleicht deployed, welches Produkt. Also es ist auf jeden Fall auch sehr hilfreich, dass man nachschauen kann, was hat sich denn geändert, wenn irgendwelche Fehler auftreten.

GraphQL View:

-

Q: Würden Sie das Tool einsetzen?

A: Ja, definitiv.

Q: Haben Sie allgemeine Verbesserungsvorschläge?

A: Es sollte natürlich möglichst intuitiv sein, für jeden anwendbar. Man müsste halt schauen, wer alles Zugriff darauf bekommt, also es muss dann halt Rollen-Rechte-Konzepte berücksichtigt sein. Ansonsten gefällt es mir eigentlich sehr gut.

Q: Welche weiteren Anwendungsfälle erkennen Sie neben der IT-Landschaftsdokumentation?

A: Ja, nicht nur Dokumentation, sondern wirklich auch für Analyse im Fehlerfall und grundsätzlich eben Aufzeigen von Schwächen in der Architektur, irgendwelche Zirkelbezüge, sind irgendwelche Daten unnötigerweise durch drei Microservices geschleust bevor sie zum Empfänger kommen wenn er sich es auch direkt von der Quelle holen könnte, solche Themen und für mich, weil ich viel im Produktkatalog unterwegs bin, Dartstellung von Abhängigkeiten von Produkten, dass ich vielleicht auch Produkte auf Basis dieser Informationen anders schneiden würde, weil eben Themen aufgrund ihrer Abhängigkeiten doch näher zusammengehören als wir sie heute so im Produktkatalog dargestellt haben. Vielleicht könnte man auch visualisieren, z.B. farbig darstellt, was schon alles in der Cloud ist und was ist noch on-premise. Weil die Frage kommt auch immer wieder. Um einfach mal was an die Wand werfen zu können, um ein Gefühl zu geben, ok , wir sind 50% on-premise und 50% in der Cloud. Neben der Visualisierung sollte alles auch tabellarisch auswertbar sein, z.B. wie viele Microservices haben wir im Einsatz, wie viele sind in der Cloud, wie viele on-premise, damit man das auch mit Zahlen belegen kann.

Q: Haben Sie weitere Anmerkungen oder Anregungen?

A: Grundsätzlich ist das für mich auch bzgl. Forschung die Zukunft der Dokumentation von Architektur. Wir müssen von dem manuellen Prozess wegkommen, weil er nicht funktioniert oder im Zweifelsfall eben veraltet ist. Von dem her finde ich das auch super, dass ihr euch da forschungstechnisch damit beschäftigt. Da ist es natürlich auch interessant, wie machen das andere Unternehmen oder was haben die noch für gute Ideen, dass man da noch einen übergreifenden Blick darauf hat.

A.2.2. Transcripts 2 & 3

Note: The following transcript represents an interview that was conducted with two interviewees at the same time. Since some of the answers build upon each other and can only be fully understood within the context of the full discussion, it was decided to transcribe the interview as a whole, the answers, however, were taken into consideration separately regarding the evaluation.

Q: Wie lange arbeiten Sie im EA-Bereich und was gehört zu Ihren täglichen Aufgaben?

A1: Ich arbeite 7 Jahre für das Unternehmen, in den letzten Jahren waren das vor allem Architekturtätigkeiten. Wir hatten eine Struktur mit Abteilungsarchitekten, d.h. wir

sind noch in Projekte involviert gewesen und haben die Kollegen beraten beim Einsatz von Technologien, beim Design von Architekturen und auch Applikationen, wenn wir gefragt wurden und hatten dann eben einen abteilungsübergreifenden Austausch aber die Architekten gehörten immer noch zu den eigentlichen Abteilungen. Dieses Jahr hatten wir eine Reorganisation, jetzt gehöre ich nicht mehr offiziell zur Offboard-Architektur, zur Gesamt-ConnectedCompany-Architektur.

A2: Ich bin 2,5 Jahre hier, als Abteilungsarchitekt tätig und wirke in der Beratung von Projekten mit. Architekturentscheidungen sind eines der wichtigsten Themen, aber für mich ist eines der wichtigsten Themen auch die Dokumentation der Backend-Landschaft. Von daher ist das für mich natürlich besonders relevant und ich arbeite ja im Prinzip an einer ähnlichen Idee, auch schon seit fast 2 Jahren, wo wir auch Dynatrace hergenommen haben und auch schon mal visualisiert haben, von daher ist es vom Konzept her relativ ähnlich zu dem was ich auch schon gemacht habe bisher, was natürlich nicht schlecht ist, weil wenn mehrere auf die gleiche Idee kommen, dann scheint die Idee wohl nicht so schlecht zu sein.

Q: Inwiefern akzeptieren Sie die genannte Problembeschreibung? Sind Sie anderer Meinung?

A2: Ja, ich sehe das Problem ähnlich. Wenn man so Tools nimmt aus dem Enterprise Architecture Management, wo die Idee ist, dass man die Lanschaft rein dokumentiert, sieht man, es funktioniert nicht. Jetzt ist die Frage: Woran liegt das? Was ist die Ursache? Ist das einfach ein Tool-Problem, ist es ein Problem, weil es keinen interessiert oder einfach keiner reinschaut, weil sich keiner verantwortlich fühlt? Das ist mir nicht so 100% klar an der Stelle. Deswegen ist die Frage, ist das einfach nur, weil es manuell ist, ist das das Problem, ist das wirklich so time-consuming, weil aus meiner Sicht, im Endeffekt, wenn man die Landschaft beschreibt, was macht man da, man beschreibt Microservices, Schnittstellen und Beziehungen zwischen den Schnittstellen, das ist eigentlich nicht besonders viel Arbeit, wenn man es sich genau anschaut. Eigentlich könnte man es von der Arbeit her machen, deswegen würde ich diesen „time-consuming task“ ehrlich gesagt ein bisschen in Frage stellen, finde ich nicht, dass es time-consuming ist, wenn man es im Vergleich mit den ganzen anderen Dingen, die man macht, z.B. eine Schnittstelle anzubinden an ein anderes System, was da an Arbeit anfällt, die Dokumentation darüber zu schreiben ist jetzt von der Menge her eigentlich relativ wenig. Dass die ganzen Modelle outdated und incomplete sind, ja, da gebe ich auf jeden Fall Recht, das stimmt, ist bei uns aktuell auch so und deswegen, dass Entscheidungen auf schlechter Qualität basiert sind, stimmt auch. Dass es keine klare Verantwortlichkeit für die Dokumentation gibt, das glaube ich nicht, eigentlich gibt es eine klare Verantwortlichkeiten. (A1: Sehr klare Verantwortlichkeit, ist klar definiert.) Das würde ich sagen, ist bei uns nicht der Fall. Manche Teams machen es halt für sich lokal, sodass die Dokumentation lokal ist. Da ist eher das Problem, dass die Dokumentation verstreut ist. Die einzelnen Systeme sind dokumentiert, teilweise sehr gut dokumentiert, teilweise vielleicht weniger gut, aber Dokumentation ist im Prinzip da, nur sie ist nicht in sich

zentral in einem Modell, sondern sie ist verteilt in Dokumenten wie PowerPoint-Slides oder Visio-Diagrammen, die man schlecht miteinander auch verknüpfen kann. Dass die IT-Landschaft sich zu schnell ändert, zu schnell für manuelle Dokumentation, würde ich auch nicht sagen, weil wenn ich einen neuen Microservice einführe oder eine neue Schnittstelle, brauche ich da für die Implementierung schon mehrere Wochen, die Dokumentation dazu ist eigentlich fast vernachlässigbar. Deswegen würde ich den Punkt auch nicht so unterschreiben.

A1: Ich habe eine etwas abweichende Sicht. Zu der Zeit, wenn man die JSON-File anlegt, die dann die Fachlichkeit beschreibt und die Verknüpfung zu dem AppMon anlegt und ein AppMon einzurichten dauert auch, kommt auf ungefähr das Gleiche raus. Was mir fehlt als Punkt 5 ist: Es fehlen Carrot Stick, und/oder. Es gibt keine Motivation für die Leute, es passiert nichts im Sinne von einer Bestrafung oder im Sinne von etwas Positivem. Wenn du das machst, das zentral dokumentierst, dann passiert irgendwas Gutes für dich. Das ist heute einfach nicht so. Das große Problem, was ich sehe, ist die Legacy. Unsere Platform dürfte in Amerika Alkohol trinken, ist 21 Jahre alt. Natürlich keine der Applikationen, die damals live war, ist heute noch live, aber es war eine Evolution, ging immer weiter und jedes Mal hat man irgendwelche Dependencies wieder aufs Alte mitgeschleppt und niemand weiß das heute mehr. Den Ist-Stand wirklich zu dokumentieren geht doch gar nicht.

A2: Das ist auf jeden Fall ein guter Punkt, ja.

Q: Wie bewerten Sie den Ansatz, Architekturinformationen aus Laufzeitdaten zu extrahieren, um somit die IT-Landschaftsdokumentation zu unterstützen? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A1: Sagen wir mal 2+ mit Potenzial. Was auf jeden Fall noch reinmuss ist die Verknüpfung mit dem Code. In der Betrachtung bei dir in der Präsentation hat man ja auch gesehen, dass je länger du das System betrachtet hast, desto mehr hast du dazu gelernt. Und dieses Delta kann man natürlich durch eine lange Betrachtung klein halten oder wenn man in den Code reinguckt und sieht, da sind so und so viele REST-Endpunkte definiert, selbst wenn ich die in meiner Laufzeitanalyse noch nicht gesehen hab, irgendwann werde ich sie wahrscheinlich sehen, um auch solche Deltas aufzeigen zu können, dass zwar irgendein Endpunkt gepflegt wird, weiterentwickelt wird und er ist im Code drin aber man könnte ihn eigentlich schon längst killen. Das wäre dann der Schritt zur 1.

A2: Also ich sehe da eine 3, aus mehreren Punkten. Ich beschäftige mich ja schon lange mit dem Thema und habe da mittlerweile auch eine relativ klare Meinung dazu gebildet. Ich glaube, Laufzeitinformationen zu verwenden für die Architektur ist auf jeden Fall extrem wichtig, dass man es macht, aber ich glaube, es ist einfach nicht ausreichend. Das hat mehrere Gründe. Das Schöne ist, du hast die Realität, du hast eine Abbildung der Realität, d.h. du weißt wirklich, was passiert zu einem bestimmten Zeitpunkt. Du hast aber das Problem, z.B. du gehst drei Monate zurück, dann weißt du nicht, was du vor drei Monaten gesehen hast, ob das jetzt noch relevant ist. Es kann sein,

dass das System zwischenzeitlich abgebaut wurde. Das heißt, du arbeitest wieder mit Information, die vielleicht schon wieder veraltet ist. Das Fenster muss aber eine gewisse Größe haben, sonst siehst du vielleicht bestimmte Interaktionen nicht, weil sie vielleicht selten passieren. Das heißt, du hast immer eine gewisse Ungenauigkeit. Deswegen glaube ich, dass es nicht komplett ausreicht. Das ist der eine Punkt. Der zweite Punkt ist, du siehst zwar, dass was passiert, aber du siehst nicht, warum es passiert, du kannst es nicht erklären. Du hast zwar ein Bild mit vielen Abhängigkeiten, das hilft natürlich schon mal unglaublich viel, das ist sehr hilfreich, aber du kannst nicht sagen, warum System A jetzt System B aufruft, du hast keine Begründung dahinter. Es reicht nicht aus als eine Beschreibung, um wirklich die Architektur zu verstehen.

A1: Dafür gibt es ja die JSONs, um die anzureichern.

A2: Da kriegst du eine gewisse Verknüpfung, ja. Deswegen glaube ich, es ist sehr hilfreich, aber nicht koplett ausreichend. Dann gibt es noch einen dritten Punkt, der für mich auch entscheidend ist. Wir haben in unserer Landschaft nicht alle Systeme in Dynatrace drin. Wenn wir wirklich alles in Dynatrace drin hätten, dann hätte ich auch ein komplettes Bild, aber so ist immer Dynatrace auch nur ein Ausschnitt von unserer Landschaft. Wir haben Legacy-Systeme, die kein Dynatrace drin haben, wir haben Fahrzeuge, die kein Dynatrace drin haben, wir haben verschiedene andere Komponenten, die kein Dynatrace drin haben, d.h. auch Dynatrace gibt auch nur einen Ausschnitt aus unserer Landschaft wieder, deswegen ist es auch aus dem Grund nicht ausreichend. Dynatrace ist sehr wichtig, um zu sehen, was wirklich passiert, aber ich glaube, es ist nicht komplett ausreichend, um wirklich eine vollständige Architekturbeschreibung zu haben.

Q: Wie bewerten Sie den Ansatz, weitere Beziehungsinformationen in Konfigurationsdateien zu verwalten? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A2: Das braucht man im Prinzip auch. Ob man das mit einer JSON-Datei macht oder ein Excel-Sheet.

A1: Die JSON kannst du direkt ins Git reinpacken und prüfen lassen, ob die Pflichtattribute befüllt sind, sonst kann er nicht mehr bauen, dann haben wir wieder einen Stick, die Excel-Datei hat jetzt wieder keinen Stick.

A2: Klar, wenn es Mal so gemacht werden kann, ist es natürlich ideal, aber auch da ist so ein bisschen das Problem, du musst alle wieder dazu bringen, das zu machen. Wenn wir alle dazu bringen, das zu machen, passt.

A1: Du brauchst ja nur einmal Top-Level Approval, dass die Datei eine Pflichtdatei wird im Build-Prozess und fertig.

A2: Dann müssten wir auch alle erstmal in die gleiche Pipeline reinkriegen, das ist auch schon nicht der Fall. Aber ja, grundsätzlich finde ich das auch eine gute Idee. Da haben

wir auch schon darüber nachgedacht. Das ist vom Ansatz her glaube ich schon auch der richtige Ansatz.

A1: Ich würde sagen eine 3, ist neutral, ist halt notwendig, es ist aber auch keine revolutionäre Idee.

A2: Wenn man es mit anderen Ansätzen vergleicht, ist es schon ein guter Ansatz, also eine 2 sage ich Mal. Mir fällt jetzt auch nichts Besseres ein.

Q: Wie bewerten Sie den Ansatz, die Pflege der Konfigurationsdatei mit Hilfe von JSON Schema sicherzustellen? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A2: Finde ich Pflicht, muss man machen.

A2: Das muss man machen, damit keine Pflichtattribute fehlen.

Q: Inwiefern trägt der Lösungsansatz im Allgemeinen zur Verbesserung der EA-Dokumentation bei?

A1: Zeigt definitiv große Unterschiede auf und Probleme. Man wird keine EA-Dokumentation approven, die gar nicht mit dem hier übereinstimmt. Es macht einfach keinen Sinn, v.a. wenn Teile aus dem hier in der EA-Doku fehlen. Es kann wiederum andersrum sein, dass die EA-Doku mehr abdeckt als das hier, weil nicht alles im Monitoring-System drin ist, z.B. unsere Fahrzeuge. Wenn man das Enterprise Architecture weglässt, als operative Dokumentation, Root-Cause-Analyse, Triage, etc. Oder auch kurzfristige Entscheidungen zu treffen, ob man überhaupt einen Client-Change-Request annimmt oder welche Clients muss ich denn betrachten, das geht zwar jetzt in die Richtung von Enterprise Architecture, da ist viel hilfreicher glaube ich noch als Enterprise Architecture.

Q: Welche Barrieren erkennen Sie bei der technischen Integration des Ansatzes? Insbesondere in Anbetracht der technischen Voraussetzungen, die erfüllt werden müssen.

A1: Genehmigung von der Stelle im Unternehmen, die für Privacy zuständig ist. Weil in den erfassten PurePaths Kundendaten drin sind und auch bei anderen Firmen sein könnten und dementsprechend nicht ohne Weiteres gespeichert werden dürfen, aber sonst, ein technisches Problem sehe ich keins.

A2: Wenn man sich unsere Dynatrace-Installation anschaut, die hat halt keine schöne API, die man gut abfragen kann, da haben wir im Prinzip ja so einen Workaround gebaut, den du ja im Wesentlichen gemacht hast.

A1: In Zukunft wird sich die Situation da aber auch verbessern. Sowohl AppDynamics als auch DataDog und Dynatrace wollen Open Telemetry unterstützen, d.h. die Architektur wie wir sie heute haben mit Agenten oder einer Applikation, die selber Daten sendet, einem Kollektor und einem Backend, das dann die Daten empfängt und aufbereitet, Machine Learning macht und was weiß ich nicht, wird standardisiert an den Schnittstellen

und das, was du dann in Zukunft nur noch implementieren musst, ist die Schnittstelle hinten Richtung Backend, sodass du die Daten empfangen kannst und fertig. Dann tust du so als wärst du die Applikation, dann kannst du alles sammeln live.

A2: Ansonsten sehe ich jetzt auch kein technisches Problem.

Q: Inwiefern halten Sie die Integration des Ansatzes in eine CI/CD Pipeline für sinnvoll? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A1: Das muss ein Stop-Kriterium sein. Wenn das nicht da ist, die JSON-Datei, darf nicht gebaut werden.

A2: Das ist der beste Ansatz normalerweise um Leute zu zwingen etwas zu tun. Von daher glaube ich auch, dass es sinnvoll ist.

A1: Es würde natürlich Hass erzeugen, aber es geht nicht anders.

A2: Es muss zuverlässig sein, sagen wir mal so. Es darf kein Mechanismus in die Pipeline eingebaut werden, der irgendwann Mal aus irgendeinem Grund vielleicht nicht funktioniert, weil dann hast du ein Problem, wenn du nachst um 2 einen Hotfix ausspielen willst und dann geht die Pipeline nicht aber ich glaube es ist sehr stabil. Sollte passen.

A1: JSON-Prüfung kriegen wir hin, ja. Note 2.

A2: 2, ja.

Q: Wie bewerten Sie den Ansatz, die Dokumentationspflicht in Richtung Entwicklerteams zu verlagern? Welche Vor- und Nachteile sehen Sie? Wie bewerten Sie den Ansatz auf einer Skala von 1-5?

A1: Easy.

A2: Genau so muss es sein.

A1: Es ist Status Quo für uns.

A2: Es ist auch bei uns so, dass die Teams relevant sind, um um die Dokumentation zu pflegen, es ist nicht die Verantwortung von irgendeinem Enterprise-Architekten.

Q: Welche Personen müssen Sie involvieren, um den Ansatz in die bestehende Organisation zu integrieren? Welche Rolle spielen die Personen? Welche Personen sehen Sie als Treiber/Unterstützer und welche als Blockierer?

A2: Wenn ich in der übergreifenden Architekturgruppe bin, kann die Gruppe das definieren wahrscheinlich.

A1: Top-Management oder Senior Management oder der Hauptabteilungsleiter muss halt einmal sagen: Ja, ihr dürft das als hartes Kriterium in die Pipeline einbauen und ab jetzt wird so dokumentiert.

A2: Und dass es dann verpflichtend wird für alle.

Q: Wie bewerten Sie im Allgemeinen das Kosten-Nutzen-Verhältnis des Ansatzes?

A1: Die Kosten schätze ich, ehrlich gesagt, relativ gering ein, das dürften selbst für die lernresistenten Dienstleister weniger als zwei Tage sein und der Nutzen ist auch gerade explizit in der Root-Cause-Analyse sechs-, siebenstellig.

A2: Vom Nutzen her würde man schauen, wie man das Tool möglichst sinnvoll einsetzen kann, das hängt bisschen auch dann vom Tool ab. Die Daten sind ja da, aber ob man mit den Daten dann auch wirklich arbeiten kann, ist so ein bisschen eine andere Frage. Von den Kosten her sehe ich es genauso. Da sind keine großen Kosten dahinter. Wenn man das einmal eingebaut hat in die Pipeline jedes Team, ist eine Datei, ist relativ klein, Dynatrace ist sowieso schon da bei uns, von daher sind das auch keine Extrakosten. Was der Nutzen letztendlich ist, ist glaube ich schwer zu beziffern.

A1: Was man jetzt halt noch bräuchte, wäre ein Application Insights-Konnektor und ein AppDynamics-Konnektor, weil dann hätten wir Stand heute, sagen wir Mal, 90% unserer Architektur erfasst, unserer Ist-Landschaft.

A2: Für mich die Frage auch: Funktioniert das auch mit mehreren Tools übergreifend? Da ist man normalerweise eigentlich blind. Außer dass man Produkte findet, aber die Aufrufe zwischen den Produkten sieht man dann nicht mehr.

A1: Man müsste einen Link herstellen zwischen verschiedenen Monitoring-Tools.

A2: Ich glaube eher, dass der richtige Weg ist, das bei uns zu fixen und ein einheitliches Tool zu benutzen.

A1: Nach der Reorg ist vor der Reorg. Ich habe inzwischen gelernt dieses Jahr, es gibt sogar Leute, die nutzen NewRelic bei uns. Kommt die nächste Reorg, zack, hast du das an der Backe, selbst wenn du hier vereinheitlicht hattest.

Q: Wie würden Sie die Verbesserung der IT-Landschaftsdokumentation bewerten innerhalb der Dimensionen (Daten-) Vollständigkeit, Aktualität, Konsistenz und Zuverlässigkeit?

A2: Die Daten sind natürlich dann irgendwie aktuell, vollständig sind die nicht unbedingt, weil das ist nur ein Teilausschnitt unseres Ganzen, ob sie vollständiger werden, als das, was wir haben, weiß ich nicht, vielleicht. Zuverlässigkeit hast du das Problem, dass du nie so genau weißt, ob du jetzt alles erwischt hast oder nicht in deinem Zeitfenster bzw. ob etwas nicht schon wieder veraltet ist. Konsistent ist es wahrscheinlich schon, ja.

A1: Ich glaube, es zahlt auf alle diese Sachen ein. Viele Dinge bzgl. EAM sind in die Zukunft gerichtet. Bisher betrachten wir in der Diskussion nur die Vergangenheit bis zum Status Quo, bis heute minus wahrscheinlich sogar ein paar Stunden. Was jetzt gerade passiert, siehst du auch nicht. Ich glaube trotzdem, dass es die Situation wesentlich besser beschreibt als alles, was wir heute haben. Beim Live-Zugriff ist

die Frage: Welchen Mehrwert bietet der Einsatz des Tools gegenüber dem direkten Einsatz des Monitoring-Tools. Die einheitliche Anfragesprache ist nett aber nicht das Killerargument. Schön wäre es tatsächlich die Datentöpfe verschiedener Monitoring-Tools zu verbinden. Du hast einen Absprungpunkt vom einen in das andere, da dann trotzdem noch die Sichtbarkeit zu behalten.

A2: Das würde ich wirklich als Mehrwert sehen, wenn man ein Modell für verschiedene Lösungen nutzen kann.

A1: Im Business-Bereich sehe ich noch großes Potenzial, dass man den AppMon PurePath stärker mit einem Service, einem kundengerichteten Service oder einer Public API, in Verbindung bringt. Da gibt es verschiedenste Systeme, die notwendig sind, um so einen Dienst zu erbringen und den erfassen wir als PurePath. Es gibt aber im Moment keine direkte Beziehung, die irgendwo dokumentiert ist. Dass diese Public API diese und jene fachliche Funktionalität erbringt, das ist diese Erklärung, die da fehlt im Moment und welche Applikationen sind denn eigentlich daran beteiligt, den zu erbringen. Diese beteiligten Applikationen sind eher noch monolithisch gebaut, d.h. sie haben Hunderte von Schnittstellen und nur zwei davon sind notwendig, um diesen Service zu erbringen. Das kann dir heute keines von diesen APM Tools geben. Es gibt noch nicht Mal die Konfigurationsmöglichkeiten, dass du die Metadaten hinzufügst, d.h. mit deinem externen Tool sind genau da die Möglichkeiten, einen Mehrwert zu schaffen, weil die anderen haben es nicht.

Q: Wie bewerten Sie die einzelnen Darstellungen auf einer Skala von 1-5?

Business Landscape View:

A1: Während der Reorg-Planung habe ich gelernt, dass so ein Diagramm ganz schön wertvoll sein kann. Dementsprechend würde ich dem eine 2 geben. Das ist ja auch recht stabil. Es könnte sein, dass bei einer Snapshot-Betrachtung eine Applikation hier verschwindet, eher unwahrscheinlich.

A2: Ich sag Mal 3. Ich bin mir nicht so sicher, ob das die beste Darstellung ist, weil im Prinzip hast du eine Baumdarstellung und ob man es nicht einfach als Baum darstellt. Die Anordnung im Raum hat eigentlich keine Bedeutung. Vom Inhalt her, ja, es ist schon interessant.

Application Landscape View:

A2: Wenn du auf ein System fokussieren könntest, dann würde es vielleicht Sinn machen, aber für die ganze Landschaft, es wird einfach viel zu groß. Ich glaube fast, es ist nicht hilfreich, wenn man die gesamte Landschaft anzeigt, da siehst du meistens sowieso nichts, außer du bist irgendwie in der Lage so zu abstrahieren, dass du am Ende nur noch so zehn Kästen hast oder so, aber sobald du mehr als zehn Kästen hast, wird es schwierig. Du interessierst dich für ein bestimmtes System und willst auf dieses System fokussieren und da willst du dann vielleicht sehen, wo deployed das darauf, das macht dann mehr Sinn. In der Form, wie es aktuell ist, ist es nicht hilfreich, also 5.

A1: Note 2.

Table View:

A2: Finde ich schon wichtig. Note 2.

A1: In den letzten sechs Wochen habe ich auch gelernt: Es ist eine wichtige Liste. Note 2.

Communications View:

A1: Die Grundvoraussetzung dieses Tool zu verwenden ist ein APM Tool zu haben. Die nennen das hier Transaction View, das ist Butter und Brot, ehrlich gesagt, die View ist eine 4, weil ich nicht weiß, was der Mehrwert ist. Damit das funktioniert hat jemand die JSON-Dateien gebaut. Es gibt ja das Konzept der Application in Dynatrace auch oder auch in AppDynamics. Die Konfiguration kann man auch dort vornehmen.

A2: Note 3. Das hilft dir wenig, weil du siehst vor lauter Bäumen quasi den Wald nicht mehr. Deswegen glaube ich so Überblicksbilder sind ganz schön, um zu sehen, wie komplex ist eigentlich unsere Landschaft, aber viel mehr hilft es dir an dieser Stelle nicht.

Application Interactions View:

A1: Das ist sweet. Das ist nah dran an dem, was ich vorhin meinte mit den Services, noch nicht ganz da aber im Rahmen von der 1 bis 5 Bewertung ist das für mich eine 1, weil das ist der Mehrwert, den die anderen so nicht haben, weil das ist viel mehr als die Transaction view, die Dynatrace bietet.

A2: Würde ich auch eine 1 geben. Man weiß jetzt natürlich immer noch nicht warum, also man hat keine Erklärung der Dinge. Das finde ich ziemlich cool, weil man hier einen PurePath auswählen kann und sich anschauen kann. Das ist schon echt nice. Wenn du oben noch die Erklärung hättest, also was ist da für ein Business Use Case eigentlich dahinter, wenn wir jetzt ein Mapping hätten von URL auf Erklärung, was da eigentlich passiert. Das finde ich cool. Was es noch besser machen könnte, ist die Kanten zur Application Collaboration rauszuschmeißen, die interessieren uns nicht, weil das Bild kann man dadurch einfacher machen. Die würde ich weglassen für mehr Übersichtlichkeit.

Comparison View:

A1: Bei einer Root-Cause-Analyse ist es natürlich auch interessant zu wissen, warum geht jetzt das Feature XYZ nicht mehr, geht man rein, vergleicht, seit letzter Woche haben sich diese und jene Pfade geändert und dann weiß ich welche drei Applikationen ich ansprechen muss. Tritt das jetzt häufig auf, wissen die Beteiligten wirklich nicht, wen sie ansprechen müssen? Hmm. In die Zukunft gerichtet um es mit dem Zielzustand zu vergleichen, wenn ich dann sagen kann, diese und jene Abweichung, die ist OK, zeig die gar nicht mehr an, ich will den Wald sehen und nicht die einzelnen Bäume, ja, das ist nützlich.

A2: Da ist das Problem, dass mir die Begründung fehlt, warum ist das nicht mehr da. Ist das einfach nur, weil in dem Zeitraum der Aufruf nicht da war oder ist es jetzt wirklich weg? Das muss man wieder interpretieren, es hilft also nichts darauf zu schauen. Jetzt müsste man sich jede Kante einzeln anschauen und gucken, warum ist das jetzt eigentlich so. Dann gibt es doch wieder so eine Fehlerrate, paar false positives. Bei den PurePaths haben wir sowieso generell das Problem, dass wir da niemals vollständig waren. Gut, das ist wieder ein Dynatrace-Problem, aber da müsste man schon sehr sehr sicher sein, ob man da eigentlich vollständig ist. Sonst kriegt man aufgrund verschiedener Ausschnitte Differenzen angezeigt, die eigentlich irrelevant sind, weil es einfach nur ein Dynatrace-Ausschnittsproblem ist. Da glaube ich hilft es nicht so viel an der Stelle.

A1: So wie sie jetzt ist eine 3, weil die Zukunft nicht implementiert ist. Wenn die Zukunft dazu kommt, dann ist es ein Alleinstellungsmerkmal und eine 1.

A2: Ich finde es von der Grundidee ziemlich cool, aber von den Daten her ist es einfach schwierig. Ich wüsste jetzt keinen Use Case, wo mir das wirklich helfen würde, das automatisch zu erstellen und angezeigt zu bekommen. Es hat Potenzial, 3.

GraphQL View:

-

Q: Würden Sie das Tool einsetzen?

A1: Ja.

A2: Ja.

[Note: Interview partner A1 had to leave at this point due to time restrictions.]

Q: Haben Sie allgemeine Verbesserungsvorschläge?

A2: Mir fällt ein Use Case ein für die Comparison. Wir haben ja verschiedene Hubs. Was ich eigentlich ganz cool fände, wäre, wenn man verschiedene Hubs nebeneinander legen könnte, das wär wirklich hilfreich, um Unterschiede zwischen den Hubs festzustellen. Das fände ich recht spannend, weil da habe ich schon üfter darüber nachgedacht über sowas und das wär ein Use Case, mit dem ich viel anfangen könnte, was Dynatrace auch nicht liefern kann. Dynatrace sieht die einzelnen Hubs getrennt voneinander. Wenn man das in einem Tool hätte und wäre sogar in der Lage das miteinander zu vergleichen, wäre das ganz interessant.

Q: Welche weiteren Anwendungsfälle erkennen Sie neben der IT-Landschaftsdokumentation?

A2: Wenn man eine gute IT-Dokumentation hat, kann man das natürlich für verschiedene Use Cases verwenden, z.B. können Tester das nutzen, um zu verstehen, wie die Systeme miteinander zusammenhängen, man kann es in Operations verwenden, usw. Cloud-Ready Analysis oder Guideline Compliance würde ich sehen als Use Case für eine gute IT-Dokumentation. Daraus kann ich dann viele Dinge tun.

Q: Haben Sie weitere Anmerkungen oder Anregungen?

A2: Ihr müsst glaube ich sehr stark herausarbeiten, was wirklich der Unterschied zu einem reinen Dynatrace ist, weil da ist ja auch schon alles da, mehr oder weniger. Für mich gilt: Ich bin im Endeffekt immer noch davon überzeugt, dass man trotzdem noch eine manuelle Dokumentation braucht, dass du nicht ohne eine manuelle Dokumentatin auskommst. Das Tool zeigt dir relevante Information, aber da steckt noch so viel mehr drin hinten dran einfach um das System zu verstehen. Wenn du schon ein ganz gutes Verständnis hast für die Landschaft, dann kannst du da draufgucken und dann hilft es dir vielleicht. Wenn du das aber nutzen willst, um die Architektur zu verstehen, dann kannst du es eigentlich nicht benutzen aus meiner Sicht. Du weißt nur Kasten A ruft Kasten B auf, du hast den Namen vielleicht noch und das war es. Dann kannst du damit nichts anfangen. Dafür brauchst du, glaube ich schon, eine manuelle Dokumentation, da bin ich relativ stark von überzeugt. Die Argumente gegen manuelle Dokumentation fand ich relativ schwach. Aus meiner Sicht ersetzt das keine manuelle Dokumentation, es ist eine Ergänzung oder vielleicht zur Validierung kann man es gut nutzen, weil komplett ersetzen tu ich mich ein bisschen schwer. Ich find das Beste an dem Ganzen ist, was du am Ende gezeigt hast, die Application Interaction view, weil das echt hilfreich ist. Im nächsten Schritt, wenn ich jetzt Use Case-Modellierung mache, was ihr habt ist die statische Sicht, aber was oft fast wichtiger ist, ist die dynamische Sicht. Mit dynamisch mein ich jetzt nicht zur Laufzeit, sondern einzelne Use Cases. Wenn ich einen Door Unlock mache als Use Case, was passiert dann in meiner Landschaft, welche Systeme werden aufgerufen, diese Pfade hier zu zeigen. Vielleicht kann man hier noch eine Nummer hinzufügen und einblenden, dass man sieht in welcher Reihenfolge das durchläuft. Da fehlt halt jetzt die Verknüpfung wiederum in die Business-Welt, weil das ist auf technischer Ebene. Auf Business-Ebene bräuchtest du jetzt noch einen Use Case oder ein Szenario und eine Verküpfung von diesem PurePath auf dieses Szenario. Das wäre ziemlich cool, glaube ich. Das hat am meisten Potenzial von allem aktuell. Das andere ist auch gut und wichtig, solange es übersichtlich bleibt. Wo ich mich schwer tu, ist wenn man so Strukturen visualisiert, die man vielleicht nicht unbedingt als Graph visualisieren sollte, so hierarchische Strukturen eigentlich, die man eigentlich besser fast oft als Tabelle visualisieren kann. Sonst werden die hierarchischen Übersichten einfach unübersichtlich und dann helfen sie auch nicht viel. Man muss es halt auch irgendwie für den Benutzer konfigurierbar machen, glaube ich. Je nach Situation, in der man sich befindet, braucht man Mal was oder man braucht es halt Mal nicht.

List of Figures

List of Tables

Glossary

Agent Software that gathers performance data and sends it to a collector. 19–21, 23, 24, 29, 119

Agent Group Logical grouping of multiple Agents. 20, 21, 23, 29, 30, 39, 51, 57

ArchiMate An EA-related reference framework and modelling language notation. 6, 88, 93

Dynatrace A software intelligence company that provides several software solutions including APM solutions. 19, 98

PurePath Detailed trace of a request directed at a monitored application. 21, 23, 24, 29, 33, 39, 49, 51, 52, 85, 91

Transaction flow Collection of traces for every transaction in a monitored application. 21, 23, 24, 29, 39, 49, 52

TUM A university in Germany, Bavaria, in the city of Munich. 10, 93

Acronyms

AADA Automated Architecture Discovery Algorithm. 11, 37, 38, 40, 48–50, 53, 56–58

API Application Programming Interface. 21–23, 93, 94

APM Application Performance Monitoring. iv, 2, 5, 10, 11, 15, 19, 26, 27, 29, 30, 38, 51, 55, 69, 76, 77, 89, 92–94, 96–98, 119

CD Continuous Delivery. 10, 14, 71, 72, 97, 99

CI Continuous Integration. 10, 14, 71, 72, 97, 99

CSRF cross-site request forgery. 22

EA Enterprise Architecture. iv, 1–3, 6, 10, 11, 13, 15, 41, 48, 50, 53, 59, 63, 68, 74, 80, 84, 88, 91, 93, 95–100

EAD Enterprise Architecture Documentation. iv, 1, 93

EAM Enterprise Architecture Management. iv, 1, 3, 13, 97, 98

ESB Enterprise Service Bus. 13

IaC Infrastructure as Code. 71, 72

IS Information Systems. 3

IT Information Technology. 3, 51, 55–57, 61, 63, 70, 81, 83

JS JavaScript. 15, 22, 23, 41

JSON JavaScript Object Notation. 10, 14, 21, 30, 35, 62–69, 72, 76, 77, 85, 90, 92, 93, 96–99

KPI Key Performance Indicator. 63, 87, 88

QoL Quality of Life. 40, 69

REST Representational State Transfer. 21, 22, 35

SQL Structured Query Language. 19

URL Uniform Resource Locator. 22, 24, 26, 29, 39, 52

VPN Virtual Private Network. 48, 49

Bibliography

[1] M. Farwick, B. Agreiter, R. Breu, M. Häring, K. Voges, and I. Hanschke. "Towards Living Landscape Models: Automated Integration of Infrastructure Cloud in Enterprise Architecture Management". In: *2010 IEEE 3rd International Conference on Cloud Computing*. July 2010, pp. 35–42.

[2] J. Ross, P. Weill, and D. Robertson. *Enterprise Architecture as Strategy — Creating a Foundation for Business Execution*. May 2006. ISBN: 1-59139-839-8.

[3] A. Balalaie, A. Heydarnoori, and P. Jamshidi. "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". In: *IEEE Software* 33.3 (May 2016), pp. 42–52.

[4] W. Hasselbring and G. Steinacker. "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. Apr. 2017, pp. 243–246.

[5] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke. "Requirements for Automated Enterprise Architecture Model Maintenance - A Requirements Analysis based on a Literature Review and an Exploratory Survey." In: vol. 4. Jan. 2011, pp. 325–337.

[6] M. Farwick. "Towards Automation of Enterprise Architecture Model Maintenance". In: vol. 863. June 2012.

[7] M. Hauder, F. Matthes, and S. Roth. "Challenges for Automated Enterprise Architecture Documentation". In: *TEAR/PRET*. 2012.

[8] H. Holm, M. Buschle, R. Lagerström, and M. Ekstedt. "Automatic data collection for enterprise architecture models". In: *Software & Systems Modeling* 13.2 (May 2014), pp. 825–841.

[9] S. Aier, S. Buckl, U. Franke, B. Gleichauf, P. Johnson, P. Närman, C. M. Schweda, and J. Ullberg. "A survival analysis of application life spans based on enterprise architecture models". In: *Enterprise modelling and information systems architectures*. Ed. by J. Mendling, S. Rinderle-Ma, and W. Esswein. Bonn: Gesellschaft für Informatik e.V., 2009, pp. 141–154.

[10] M. Farwick, C. Schweda, R. Breu, and I. Hanschke. "A situational method for semi-automated Enterprise Architecture Documentation". In: *Software & Systems Modeling* (Apr. 2014).

[11] A. R. Hevner, S. T. March, J. Park, and S. Ram. "Design Science in Information Systems Research". In: *Management Information Systems Quarterly* 28.1 (Mar. 2004), pp. 75–105. ISSN: 0276-7783.

[12] T. O. Group and V. H. Publishing. *ArchiMate® 3.1 Specification*. 1st ed. Zaltbommel, Netherlands: Van Haren, 2019. ISBN: 1-947754-30-0.

[13] M. Kleehaus, M. Hauder, F. Matthes, and Ö. Uludag. "Enterprise Architecture Discovery via Runtime Instrumentation for Automating Enterprise Architecture Model Maintenance". In: *25th Americas Conference on Information Systems, AMCIS 2019*. Cancún, Mexico, Aug. 2019.

[14] M. Kleehaus, N. Corpancho, and F. Matthes. "Discovery of Microservice-based IT Landscapes at Runtime: Algorithms and Visualizations". In: *Hawaii International Conference on System Sciences (HICSS)*. Hawaii, 2020.

[15] M. Buschle, M. Ekstedt, S. Grunow, M. Hauder, F. Matthes, and S. Roth. "Automating Enterprise Architecture Documentation using an Enterprise Service Bus". In: *AMCIS*. 2012.

[16] A. Alegria and A. Vasconcelos. "IT Architecture automatic verification: A network evidence-based approach". In: *2010 Fourth International Conference on Research Challenges in Information Science (RCIS)*. May 2010, pp. 1–12.

[17] F. Cuadrado, B. García, J. Dueñas, and H. G. "A Case Study on Software Evolution towards Service-Oriented Architecture". In: Apr. 2008, pp. 1399–1404.

[18] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. *Continuous Monitoring of Software Services: Design and Application of the Kieker Framework*. Tech. rep. 0921. Department of Computer Science, University of Kiel, Germany, 2009, p. 26.

[19] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE '12. Boston, Massachusetts, USA: ACM, 2012, pp. 247–248. ISBN: 978-1-4503-1202-8.

[20] M. Välja, R. Lagerström, M. Ekstedt, and M. Korman. "A Requirements Based Approach for Automating Enterprise IT Architecture Modeling Using Multiple Data Sources". In: *Proceedings of the 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. EDOCW '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 79–87. ISBN: 978-1-4673-9331-7.

[21] M. Välja, R. Lagerström, U. Franke, and G. Ericsson. "A framework for automatic IT architecture modeling : applying truth discovery".

[22] F. Fittkau, S. Roth, and W. Hasselbring. "ExplorViz: Visual Runtime Behavior Analysis of Enterprise Application Landscapes". In: *ECIS*. 2015.