IT Landscape Discovery via Runtime Instrumentation for Automating Enterprise Architecture Model Maintenance

Completed Research

Martin Kleehaus
Technische Universität München (TUM)
martin.kleehaus@tum.de

Matheus Hauder
Allianz Deutschland AG
matheus.hauder@allianz.de

Ömer Uludag
Technische Universität München (TUM)
oemer.uludag@tum.de

Florian Matthes
Technische Universität München (TUM)
florian.matthes@tum.de

Nicolas Corpancho Villasana
Technische Universität München (TUM)
nicolas.corpancho@tum.de

Abstract

Enterprise Architecture (EA) model maintenance is a challenging task which is still performed mainly manually. Recent research endeavours for automating this task have not addressed runtime data for gathering the architecture of the IT-landscape. In this design science work, we want to close this research gap and present an approach for discovering the EA by combining runtime data with further relevant information that reside in federated information sources. The implemented prototype Enterprise Architecture Discovery (EAD) allows stakeholders to explore EA information from different perspectives, which supports new use cases and analysis capabilities.

We evaluate our prototype by implementing the concept in a big German insurance company. The introduction of a validation workflow enables fully automated data integration, which minimizes the effort for manual tasks. Based on interviews with different stakeholders, we could prove that the concept is feasible for discovering the as-is IT landscape and to unveil multi-level dependencies from applications up to domains.

Keywords

Architecture discovery, enterprise architecture, EAM, EA documentation, EA model maintenance, IT landscape, IT landscape reconstruction, runtime, monitoring, microservice.

Introduction

Enterprise Architecture Management (EAM) has been established as an important instrument for managing the complexity of the IT landscape and enabling enterprise-wide transparency. EAM aims to visualize the relationships among regulations, business processes, software and the underlying infrastructure. It is typically conducted in order to analyze the current enterprise architecture (EA) and to define requirements and plans for changes. These planned changes to the IT-environment, however, are mostly executed in an unchecked manner by specialists and their status is often not validated against the planned enterprise architecture model.

IT-landscape modelling, as a sub-area of EAM, tries to discover and assess the IT-landscape of an organization. EAM tools support the modelling of the IT-landscape, however, the creation of these models is a time-consuming task (Buckl et al. 2010). Enterprise architects struggle to collect EA relevant information in order to unveil the as-is IT-environment (Roth et al. 2013). The EA models are often out-of-

date which leads to decisions made on wrong data or bad data quality (Lucke et al. 2010). The reasons derive from the approach of how the data is collected which is still performed mainly manually (Buckl et al. 2010). In addition, cloud environments, new software development methodologies like agile practices, as well as the delegation of architectural decisions to DevOps teams, makes IT-landscape modelling suffer more than ever from the problem of capturing the full essence of an ever-changing landscape.

As already identified by (Farwick et al. 2013) and (Hauder et al. 2012) monitoring tools that provide runtime data of the IT landscape could be a promising information source for delivering EA relevant data. However, only a few research endeavours (Buschle et al. 2012; Farwick et al. 2010; Holm et al. 2014) leverage runtime data for this purpose as the low maturity level of monitoring tools in this time did not allow to capture the complete IT landscape with its components, relationships and application communication behavior. This has changed nowadays. New monitoring vendors, like Dynatrace or AppDynamics as well as modern Platform-as-a-Service (PaaS) provides like Cloud Foundry or Amazon Web Services provide powerful instrumentations to gain insights from an end-to-end perspective. Although this information is gathered on API level and hence is too fine granular for EAM, it unveils on an aggregated level what applications run on which environment and how they exchange data.

As several frameworks and concepts like Archimate (Iacob et al. 2012) or TOGAF (Standard and Group 2013) proposes, the EA does not only consists of technology-related layers like hardware and software, it also defines business-related aspects like business domains, processes, projects, products, teams and others which cannot be extracted from runtime data. This information is generated and stored in federated information systems like project management (PM) tools, wikis, development environment, and furthers.

In this work, we present a solution how to discover the as-is EA by leveraging runtime instrumentation that provides information about 1) the technology layer of an EA that encompasses all technological-related aspects like hardware, network and other physical components and 2) the application layer that defines software components, information systems and services that run on the technology layer. In addition, we present a concept of how to link those runtime data with further EA relevant information that cannot be extracted out of runtime data and reside in federated information systems. The collection and mapping of all mentioned information sources contribute to support the automated discovery of the EA. In a prototypical implementation, the tool Enterprise Architecture Discovery (EAD) is presented and evaluated in a big insurance company located in Germany. Hereby, we aim to answer the following research questions:

- RQ1. *How to map runtime data provided by monitoring tools with federated information sources?*
- RQ2. *How to integrate the concept into the IT environment?*
- RQ3. *How does a prototype implementation of the automated EA discovery look like?*

The remainder of the paper is organized as follows: First, we list our related work. Afterwards, we reveal our approach on how to link runtime data with static information derived from federated information systems. In the next sections, we describe how the concept can be integrated into the IT infrastructure and provide further implementation-related aspects. We continue with the discussion of our evaluation results. We finish the paper with the conclusion and a description of our future work.

Related Work

There exist a few concepts on how to integrate runtime data from existing data sources for EA model maintenance. (Holm et al. 2014) as well as (Alegria and Vasconcelos 2010) make use of network analysis tools in order to infer information on the IT infrastructure. (Buschle et al. 2012) on the other hand, interpret the configuration of an Enterprise Service Bus (ESB) to include knowledge on communicating information systems in the EA model. These approaches have in common that they are limited to a specific layer of the EA model and the authors describe their concepts about how to map the data rather superficial.

(Farwick et al. 2010) presented a similar approach by tackling the challenge of EA model synchronization with different technologies and applications. In their approach, all infrastructure elements are tagged manually with a Universally Unique Identifier, as soon as they are planned in the project management tool. After recognizing changes on these elements, they will be propagated to the EAM tool. The main difference to this research is the manual efforts which have to be applied in initially capturing these tags and keeping them up to date.

A further approach for synchronization between runtime and design time architecture is proposed by (Breu et al. 2011) and similarly by (Akkiraju et al. 2012). In comparison to the traditional agile way of software development where changes are implemented iteratively, they consider the design to be done upfront and manifest itself in architecture models. All running artefacts are the result of generation from these models and changes are directly reflected on these respective models. The nature of generating code from models is framed by the term "living models". While this idealistic approach is sensible in theory, it requires a lot of design work up front, which is not compatible with agile practices. With the lessons that can be drawn from (Buckl et al. 2010) it remains to be verified whether this can work in practice, as no larger scale evaluations are presented.

(Landthaler et al. 2018) presents a machine-learning-based approach for identifying application components in the IT landscape of the enterprise. The presented approach discovers and classifies binary strings of application executables on target machines. The main challenge is that the binary strings of executables differ depending on the devices. That means the same binary string is different for the same application version, device type and OS version. This research differs from our concept, as we discover the EA based on runtime data and references to further federated information sources.

The tool "MicroART" is an approach for recovering the architecture of microservice-based systems described in (Granchelli et al. 2017a, 2017b). The approach is based on Model-Driven Engineering (MDE) principles and is composed of two main steps: recovering the deployment architecture of the system and semi-automatically refining the obtained system. The concept recovers static and dynamic information of microservices and their interrelations from the GitHub source code repository, Docker container engine, Vagrant platform, and TcpDump monitoring tool. In contrast to our concept, MicroART is inflexible regarding the incorporation of any information source.

Pivio.io [1] is an open-source tool for describing microservice-based architectures. It follows a similar approach as we do in our concept. Pivio provides a meta-model for describing microservices in a configuration file that is automatically sent to a server as soon as the microservice instance is started. However, in order to describe the services in its full essence over 18 attributes must be filled, which represents the difference to our concept. We extract already most of the attributes from runtime which keeps the content of the configuration file to a minimum.

Mapping of runtime data with federated information sources

For our objective to discover the EA, we leverage the development in the monitoring section and extract important information out of runtime data for unveiling 1) all running IT components from the infrastructure and application layer, 2) the business activities performed by the applications, 3) the interrelationship between those layers, like which application runs on which hardware and 4) the intrarelationship that means the communication behaviour and dependencies between applications. The approach of how we extract those data is described in detail in (Kleehaus et al. 2018).

However, runtime data can only provide an excerpt of the EA. Hence, the status of ongoing projects, software development progress, performance of products and business domains as well as team definitions are not available in runtime data. This information resides in federated information systems as (Hauder et al. 2012) elaborated. Consequently, it is necessary to extract this information and to map it with the runtime data in a common data model.

Mapping of different data sources demands indispensably a foreign key relationship. We propose to establish the mapping on application or where applicable on microservice (Lewis, James; Fowler 2014) level as the smallest unit. All other information can be derived from this lowest level. However, monitoring tools unveil the IT landscape mainly from a developer and operator perspective. Hence, the name of the running applications might not follow the same standard as of how they might be used in federated information systems. For instance, consider the element "Payment Service" which name could be used for the model element. The deployed software artifact is mostly identified by their runtime artifact which is named as "application.jar". Hence, a connection is not apparent immediately.

[1] http://pivio.io

We solve this issue by establishing the links from the software artifacts to the particular information systems via a configuration file that is assigned to every runtime artifact before it is released to production. The single file ead.json is located in the root directory of the software project. We hereby leverage the technology pivio.io and modified it to our benefit. The configuration file contains the following information.

- name: A human understandable name of the artifact. This name must be unique (mandatory).
- description: Information about the purpose of the artifact (mandatory).
- application: The application as the parent IT artifact might be split into microservices (mandatory).
- boundedcontext: The domain in which the artifact "lives" (optional).
- references: The abbreviation of the information source, like "pm", "eam", "cr", "cmdb", "wiki" (optional).

The reference contains the following attributes:

- tool: The name of the tool (mandatory).
- domainURL: The URL address pointing to the information source (mandatory).
- apiURL: The URL for the API interface (mandatory).
- id: The ID of the artifact representation (optional).
- apitoken: Access credential API token (optional).

All attributes indicated by mandatory are required for the functionality of the system. The ID in the reference section is optional as it is also possible to establish the mapping via the application name or the bounded context. This is often the case for EAM tools, as applications or microservices are hardly documented in these tools. On the other hand, the bounded-context could also be derived from EAM tools if the documentation of applications is up-to-date. The reference list can be extended by any other information source as long as an adapter for accessing the tool is available. In the current state, we support the tools Iteraplan, JIRA, GIT and Jenkins.

Concept Integration

In order to ensure that the configuration file was created, and the content follows the predefined schema, we suggest integrating our concept in a continuous integration (CI) pipeline. The workflow is illustrated in Figure 1. First, the involved stakeholders agree on which references must be at least included in the file. The agreement or contract between the stakeholders is persisted as a JSON schema[2]. With JSON schema, it is simple to define a meta-model specifying mandatory and optional fields, attribute types and their properties. An additional test case in the CI pipeline ensures that the configuration file exists and meets the predefined schema. If not, the file is rejected which leads to a failed test case. If the developers do not adhere to regulations, the service cannot be deployed to production.

The CI pipeline sends the configuration file to the EAD server which registers the service for the first time and sets the `createdAt` timestamp. In case the service is already known the `lastSeen` timestamp is updated. Afterwards, EAD establishes a connection to the monitoring tool based on the returned unique ID. Hereby, runtime data as well as information from the federated information sources are not stored in a database but only referenced. The information is gathered as soon as it is requested by the user which realizes a real-time EA discovery.

As the EA discovery is interesting for stakeholders that mainly occupy an architectural or management role, like Enterprise Architects, IT Architects, or Product Owner, we suggest that those stakeholders are also responsible to define the JSON schema. The Developers make sure the configuration file is assigned to each runtime artifact and the schema validation does not reject the content due to syntax errors or reference exceptions.

Unified Service Meta-Model

The runtime artifact as the modular IT service is the smallest unit in the IT-landscape and represents the connecting link to all information sources. For the purpose of abstraction, a unified service was defined that

[2] https://json-schema.org/

is generalizing the entity for all data sources. These entities are joined by reference under the `UnifiedService` class, as depicted in the meta-model shown in Figure 2. The `UnifiedService` defines the holistic view on a runtime artifact in the IT landscape. It stores the `Properties` that are provided by the configuration file. In addition, the `UnifiedService` can also point to other parent services (predecessors) that call the `UnifiedService` via interfaces or child services (successors) that get called by the service. The `UnifiedService` contains multiple `ServiceReferences` to all provided data sources, most notably one to the runtime artifact identified by the monitoring system. Each `ServiceReference` contains important information about this runtime artifact. This allows connecting an arbitrary amount of references to other systems. The detailed implementation of a `ServiceReference` is dependent of the referenced data source and can be extended as requested. Hereby, it is necessary to write specific adapters to access the data source.

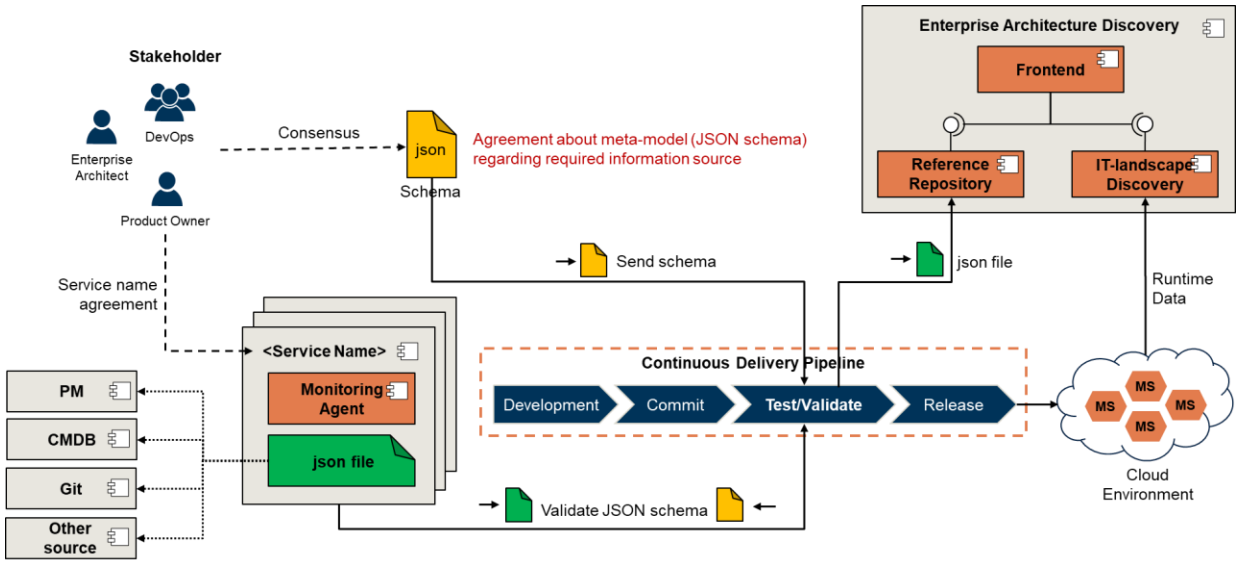

Figure 1: Workflow for validating the JSON file against the JSON schema

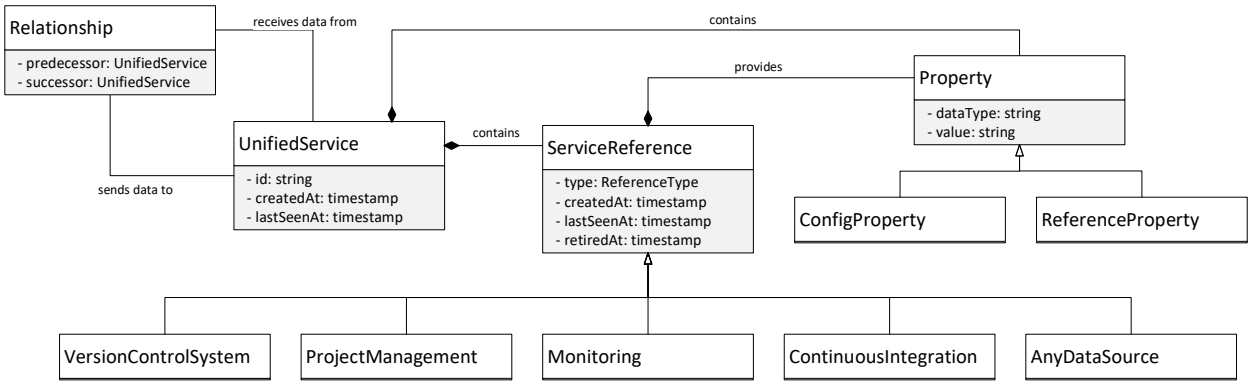

Figure 2: Unified Service Meta-Model

IT Artefact Synchronization with EAD tool

The primary goal of the synchronization process is to identify missing IT services that are not referenced in the EAD tool but discovered by the monitoring tool. EAD validates the incoming monitoring data against the own database. Unknown services lead to the creation of a new `UnifiedService` without any further reference information, that is normally provided by the configuration file. The stored name of the service is the discovered artifact by the monitoring system. As soon as a configuration file is available the update

workflow is triggered. Hereby, EAD exposes APIs that can be used to start the synchronization process manually or by a running synchronizer that frequently triggers the process automatically. The synchronization process itself is designed to be idempotent as long as no changes have occurred in the architecture, hence running it multiple times has no further impact on the result.

A new software release or updates in the connected data sources do not affect the `UnifiedServices` and their references maintained in EAD. New references can be easily added by appending them in the configuration file. Changes in product names or bounded context will be updated accordingly as well. In case of changes to the unique name of the runtime artifact, it will inevitably be considered as a new `UnifiedService`. At some point in time, the deletion flow will remove the references to the old artifact which was renamed. In case, an identified artifact is encountered whose `lastSeen` timestamp is beyond a threshold which can be defined individually to the needs of different environments, EAD will mark the service as to be retired by setting the `retiredAt` timestamp. That means services are never removed automatically from the database. This approach was chosen, as otherwise temporally stopped services that do not deliver runtime data anymore would be wrongly recognized as removed services.

Evaluation

We evaluate our concept and the developed prototype, resulted from our design science research, based on the criteria collected by (Prat et al. 2015). The criteria are rated by 12 experts in that field. However, we were not able to incorporate all the criteria as some of them demand a long-term evaluation.

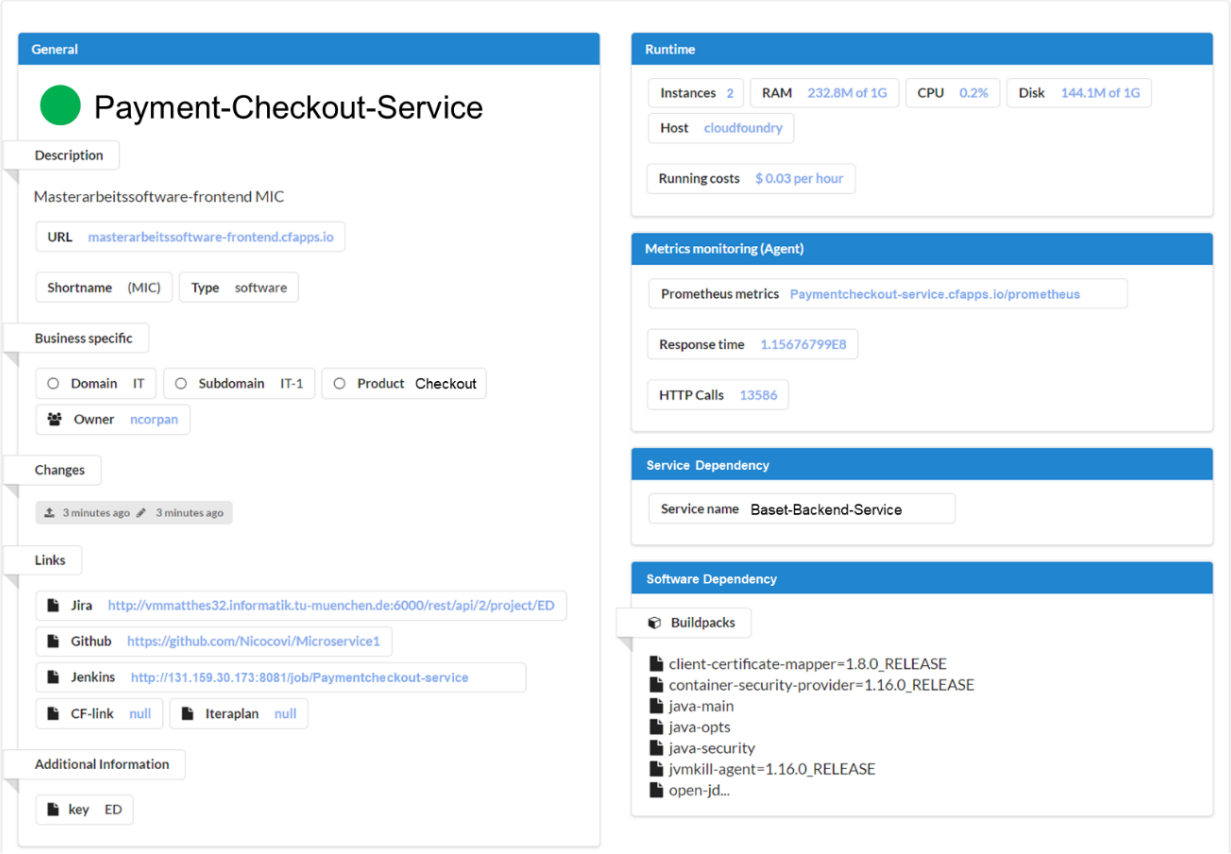

Figure 3: Web frontend of the EAD prototype

Environment

The prototype was integrated into a controlled environment of a big German insurance company. The technical environment of the company is compiled as follows: the containerized microservices of the

company are hosted by the open-source platform-as-a-service (PaaS) technology Cloudfoundry [3]. Cloudfoundry also provides the required monitoring solution. For validating the configuration files (ead.json) against the agreed JSON schema, we wrote a test case for the CI pipeline Jenkins[4], that is primarily used for releasing the microservices into production. For visualizing the result of the EA discovery, we developed a web-based application that categorizes all discovered information in the section general, runtime, metric, service- and software dependencies, project management, code repository and CI pipeline. An excerpt of the tool is illustrated in Figure 3. Further visualization (see Figure 4) unveils the relationships between the IT components, including the communication paths between microservices, based on a dynamic extendable tree diagram.

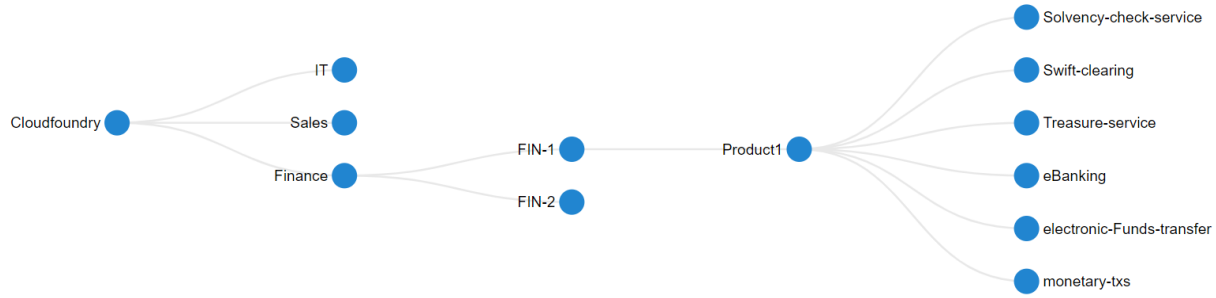

Figure 4: Dynamic extendable tree diagram for visualizing IT component dependencies. Level 1: Cloud Environment, Level 2: Domains, Level 3: Subdomains, Level 4: Products, Level 5: Microservices

Population

We conducted interviews with 12 experts that work in the mentioned insurance company. Nine experts possess the role of Enterprise Architect with 138 years working experience in total. The other three experts were Product Owners with totally 15 years working experience.

We set up a workshop with all 12 experts to find an agreement which information sources are supposed to be referenced by our prototype. Currently, a configuration management database (CMDB) is the single point of truth for unveiling the as-is IT landscape. However, as most of the data is maintained manually in the CMDB, the data quality is questioned by most of the experts. Hence, it was decided not to use it as a reference. Finally, the experts agreed on the following information sources: 1) Jenkins, which represents the CI pipeline, 2) Jira used as the project management tool and 3) Github for analyzing code commits.

Interview result

The interview experts rated the evaluation criteria within the scope of 1=strongly agree, 2=agree, 3=disagree and 4=strongly disagree. The result of the interview is depicted in Table 1 and described in the following in more detail.

Criteria	Description	1	2	3	4	Grade
Goal						
Efficacy	The software fulfills its purpose	67%	33%	0%	0%	1,3
Efficiency	Good ratio between cost and benefit	8%	75%	17%	0%	2,1
Utility	EAD is useful for achieving the goal	58%	42%	0%	0%	1,4
Feasibility						
Technical feasibility	EAD is from a technical point of view a feasible approach	58%	42%	0%	0%	1,4

[3] https://www.cloudfoundry.org

[4] https://jenkins.io

Operational feasibility	Would you support the proposed artifact and operate it	17%	75%	8%	0%	1,9
Economic feasibility	The benefits of the proposed artifact outweigh the costs of building and operating it	33%	67%	0%	0%	1,7
Environment						
Usefulness	The artifact positively impacts your task performance	33%	67%	0%	0%	1,7
Ease of use	The software is easy to learn and to use	8%	58%	33%	0%	2,3
Alignment with business	The artifact goal is aligned with the organization and its strategy	67%	33%	0%	0%	1,3
Technology fit	It is easy to integrate the artifact into the organization's IT infrastructure	17%	58%	25%	0%	2,1
Structure						
Completeness	The artifact contains all necessary EA elements and relationships between elements	33%	67%	0%	0%	1,7
Simplicity	The artifact contains the most important EA elements	67%	33%	0%	0%	1,3
Presentation	The presentation of the EA elements is appropriate	8%	92%	0%	0%	1,9
Activity						
Functionality	The artifact provides most of the functions which meet the needs	8%	67%	25%	0%	2,2
Accuracy	The degree of the accuracy of the expected output is satisfying	83%	17%	0%	0%	1,2
Performance	The artifact accomplishes its functions within an acceptable time	67%	33%	0%	0%	1,3

Table 1: EAD evaluation result. 1=strongly agree, 2=agree, 3=disagree and 4=strongly disagree, N=12

All interviewed experts confirmed that EAD is a valid approach for discovering the EA. However, EAD requires a set of tools to become fully operational, a CI pipeline and a fully monitored environment. In case both tools are not present in the company, they must be integrated beforehand which lead to an inevitable additional expenditure. Hence, some of the experts disagree with the efficiency of the approach.

The technical feasibility is perceived high amongst the experts in case the prerequisites are fulfilled. Most of the experts agree that imposing the team to incorporate a configuration file in the code repository as well as imposing the team to incorporate a test case in the CI pipeline is feasible to establish. However, one of the experts admit that he would probably not use EAD as a standalone software but as a tool for supporting the automation of EA model maintenance.

All experts agree that EAD positively impacts their performance on maintaining EA models and the purpose of the software is aligned with the organization's goals and its strategy. However, some of them initially had difficulties to learn the JSON schema for validating the configuration files, hence some experts disagree that the software is easy to use. In addition, some doubts were expressed whether the software can be easily integrated into the IT landscape as it depends on the availability of monitoring tools.

The EAD web frontend provides much information about the selected microservice, which most of them are considered to be useful. Especially the number of instances per microservice, the running costs and the number of user requests marked as important KPIs. However, one EA expert stated that the business capability is missing in this section. In addition, the separation of business owner and IT owner would be further valuable information. Missing metrics are total cost of ownership (TCO), mean time to recover (MTTR), mean time to failure (MTTF) and average number of concurrent sessions.

The dependency section (service and software dependency) unveils, on the one hand, the communication relationship between microservices and on the other hand the software dependency on the microservice. All the experts regard this section as the most important one, as it indicates the complexity of the software,

dependencies on third-party software and outdated components, as well as the loose coupling degree. The visualization approaches are considered to be useful for obtaining an overview of the IT-landscape dependencies. The link to other information sources like Jira, Github and Jenkins is perceived useful by nine experts as otherwise, they have to be searched manually.

As EAD is a prototype in an early stage, many useful features and comfort functions are missing. For that reason, some of the experts disagree with the criteria functionality. As the data sources are accessed in real-time and the queries do not fetch a huge amount of data, the response time is mainly under 1 sec, hence the performance was rated high.

Limitation

EAD was designed to be independent of the choice of the APM tool, which is reflected by the unified data model described in Figure 2. To this end, however, a few of the capabilities from the APM software were seen as given and not validated across other tools offered by different vendors. Especially, the propagation of APIs and the capability to trace requests for discovering communication dependencies between the microservices need to be fulfilled.

Furthermore, the connection to other information sources can only be realized if a proper API adapter is developed. In the current state, we are able to connect to JIRA, Github and Jenkins. The same applies for the monitoring tool. During the creation of this prototype, we developed an adapter for Cloudfoundry.

During the evaluation of EAD, we integrated the concept to one specific environment. Hence, a full discovery of the whole IT landscape of the company was not possible. A long-term evaluation on several environments could lead to more insights and suggestions for improvement. In general, the proposed solution struggles to discover the following artefacts: virtual machines, non-instrumented applications like standard software and legacy systems, as well as streaming services like MQTT or Apache Kafka. The discovery of those artefacts is part of future work.

Conclusion and future work

In this paper, we introduced a solution for discovering the enterprise architecture by analyzing runtime data. We extract application and infrastructure elements from monitoring data and enhance those data with further EA relevant information maintained in federated information systems. The concept was evaluated in one environment of a big German insurance company. Hereby, we answer three research questions:
Regarding to RQ1, we established a link between the discovered application artifacts to the particular federated information system via a configuration file that is assigned to every runtime artifact. With this link, all information reside in the federated information systems are automatically mapped to the runtime artifact. Hereby, our concept does not duplicate the information in a separate database, but we store only references via the concept of unified services, which represent the central element from which all relations diffuse. As Table 1 shows, 7 experts fully agree, and 5 experts agree that this approach is technical feasible.
In order to answer RQ2, we integrate the concept in the company's continuous integration pipeline, as this place is predestined to ensure the prerequisites are fulfilled and to validate the content of the configuration file against predefined JSON schemas.
The expert feedback for the proposed web frontend solution has been positive in general. Most of the visualizations are regarded as useful. Especially, the dependency section was mentioned as the most important visualization. Some KPIs were proposed as missing, such as TCO, MTTR, MTTF. The evaluation answers RQ3.

Based on the evaluation some potential future work could be identified. 1) The selection of what data should be visualized on the web frontend is challenging as different roles want to see different perspectives on the IT landscape. In our future work, we want to solve this problem by integrating GraphQL as a server-side query engine for executing any query against the APIs provided by the referenced information sources. 2) Furthermore, services and their interfaces represent different activities of an overall business process. We want to incorporate business process mining in our EA discovery approach and link the identified business activities with service executions. This could further help in analyzing business impact from any type of operational problem. 3) In the current prototype, we do not import and synchronize the discovered IT artifacts with an EAM tool like Iteraplan or PlanningIT. This task will be accomplished in future work.

References

Akkiraju, R., Mitra, T., and Thulasiram, U. 2012. "Reverse Engineering Platform Independent Models from Business Software Applications," in *Reverse Engineering - Recent Advances and Applications*, IntechOpen, pp. 83–94.

Alegria, A., and Vasconcelos, A. 2010. *IT Architecture Automatic Verification: A Network Evidence-Based Approach*, Fourth International Conference on Research Challenges in Information Science (RCIS), pp. 1–12.

Breu, R., Agreiter, B., Farwick, M., Felderer, M., Hafner, M., and Innerhofer-oberperfler, F. 2011. "Living Models – Ten Principles for Change-Driven Software Engineering," *International Journal of Software and Informatics*.

Buckl, S., Matthes, F., Schweda, C., and Winter, K. 2010. "Investigating the State-of-the-Art in Enterprise Architecture Management Method in Literature and Practice," in *Proceedings of the 5th Mediterranean Conference on Information Systems (MCIS)*.

Buschle, M., Grunow, S., Matthes, F., Ekstedt, M., Hauder, M., and Roth, S. 2012. "Automating Enterprise Architecture Documentation Using an Enterprise Service Bus," in *AMCIS Proceedings*.

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

Farwick, M., Breu, R., Hauder, M., Roth, S., and Matthes, F. 2013. "Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation," in *Proceedings of the Annual Hawaii International Conference on System Sciences*, Wailea, Maui, pp. 3868–3877.

Granchelli, G., Cardarelli, M., Di Francesco, P., Malavolta, I., Iovino, L., and Di Salle, A. 2017a. "MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems," in *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*.

Granchelli, G., Cardarelli, M., Di Francesco, P., Malavolta, I., Iovino, L., and Di Salle, A. 2017b. "Towards Recovering the Software Architecture of Microservice-Based Systems," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, pp. 46–53.

Hauder, M., Matthes, F., and Roth, S. 2012. "Challenges for Automated Enterprise Architecture Documentation," in *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation*, Berlin, pp. 21–39.

Holm, H., Buschle, M., Lagerström, R., and Ekstedt, M. 2014. "Automatic Data Collection for Enterprise Architecture Models," *Software and Systems Modeling* (13:2), pp. 825–841.

Iacob, M.-E., Jonkers, H., Lankhorst, M. M., Proper, H. A., and Quartel, D. A. C. 2012. "ArchiMate 2.0 Specification," *ArchiMate 2.0 Specification*.

Kleehaus, M., Uludağ, Ö., Schäfer, P., and Matthes, F. 2018. "MICROLYZE: A Framework for Recovering the Software Architecture in Microservice-Based Environments," in *CAISE Forum*, pp. 148–162.

Landthaler, J., Uludag, Ö., Bondel, G., Elnaggar, A., Nair, S., and Matthes, F. 2018. "A Machine Learning Based Approach to Application Landscape Documentation," in *The Practice of Enterprise Modeling*, R. A. Buchmann, D. Karagiannis, and M. Kirikova (eds.), Cham: Springer International Publishing, pp. 71–85.

Lewis, James; Fowler, M. 2014. "Microservices - A Definition of This New Architectural Term," *Martinfowler.Com*.

Lucke, C., Krell, S., and Lechner, U. 2010. "Critical Issues in Enterprise Architecting – A Literature Review," in *Americas Conference on Information Systems (AMCIS)*.

Prat, N., Comyn-Wattiau, I., and Akoka, J. 2015. "A Taxonomy of Evaluation Methods for Information Systems Artifacts," *Journal of Management Information Systems* (32), pp. 229–267.

Roth, S., Hauder, M., Farwick, M., Breu, R., and Matthes, F. 2013. "Enterprise Architecture Documentation: Current Practices and Future Directions.," in *11th International Conference on Wirtschaftsinformatik Proceedings*, pp. 911–925.

Standard, O. G., and Group, T. O. 2013. "Open Group Standard The Open Group," *The TOGAF® Standard, Version 9.2*.