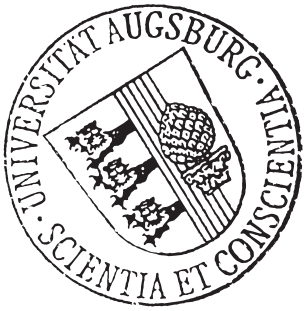
Generic analysis support for understanding, evaluating and comparing enterprise architecture models

Melanie Langermeier

Dissertation

for the degree of

Doctor of Natural Sciences (Dr. rer. nat.)



University of Augsburg

Department of Computer Science

Software Methodologies for Distributed Systems

April 2019

Generic analysis support for understanding, evaluating and comparing enterprise architecture models

Supervisor:	**Prof. Dr. Bernhard Bauer**, Department of Computer Science, University of Augsburg, Germany
Advisors:	**Prof. Dr. Alexander Knapp**, Department of Computer Science, University of Augsburg, Germany
	Prof. Dr. Bernd Heinrich, Faculty of Business, Economics and Management Information Systems, University of Regensburg, Germany
Defense	24.Juli 2019

Abstract

Enterprise Architecture Management (EAM) is one mean to deal with the increasing complexity of today's IT landscapes. Architectural models are used within EAM to describe the business processes, the used applications, the required infrastructure as well as the dependencies between them. The creation of those models is expensive, since the whole organization and therewith a large amount of data has to be considered. It is important to make use of these models and reuse them for planning purposes and decision making. The models are a solid foundation for various kinds of analyses that support the understanding, evaluation and comparisons of them. Analyses can approximate the effects of the retirement of an application or of a server failure. It is also possible to quantify the models using metrics like the IT coverage of business processes or the workload of a server. The generation of views sets the focus on a specific aspect of the model. An example is the limitation to the processes and applications of a specific organization unit. Architectural models can also be used for planning purposes. The development of a target architecture is supported by identifying weak points and evaluating planning scenarios.

Current approaches for EAM analysis are typically isolated ones, addressing only a limited subset of the different analysis goals. An integrated approach that covers the different information demands of the stakeholders is missing. Additionally, the analysis approaches are highly dependent on the utilized meta model. This is a serious problem since the EAM domain is characterized by a large variety of frameworks and meta models.

In this thesis, we propose a generic framework that supports the different analysis activities during EAM. We develop the required techniques for the specification and execution of analyses, independently from the utilized meta model. An analysis language is implemented for the definition and customization of the analyses according to the current needs of the stakeholder. Thereby, we focus on reuse and a generic definition. We utilize a generic representation format to be able to abstract from the great variety of used meta models in the EAM domain. The execution of the analyses is done with Semantic Web Technologies and data-flow based model analysis.

The framework is applied for the identification of weak points as well as the evaluation of planning scenarios regarding consistency of changes and goal fulfillment. Two methods are developed for these tasks, as well as respective analysis support is identified and implemented. These are, for example, a change impact analysis, specific metrics or the scoping of the architectural model according to different aspects.

Finally, the coverage of the framework regarding existing EA analysis approaches is determined with a scenario-based evaluation. The applicability and relevance of the language and of the proposed methods is proved within three large case studies.

Zusammenfassung

Unternehmensarchitekturmanagement (EAM) ist eine Möglichkeit, die steigende Komplexität in heutigen IT Landschaften zu bewältigen. Dabei werden Architekturmodelle eingesetzt, um die Geschäftsprozesse, die verwendeten Anwendungen, die benötigte Infrastruktur sowie deren Beziehungen zu beschreiben. Die Erstellung dieser Modelle ist aufwändig, da das gesamte Unternehmen und damit eine große Datenmenge zu berücksichtigen ist. Umso wichtiger ist es, diese Modelle aktiv zu verwenden und zu Planungszwecken sowie zur Entscheidungsunterstützung einzusetzen. Sie ermöglichen verschiedenste Analysen, die das Verständnis der Architektur verbessern, die Architektur bewerten oder vergleichen. Analysen können beispielsweise die Auswirkungen eines Serverausfalls oder der Abschaltung einer Anwendung bestimmen. Es ist auch möglich das Modell unter Verwendung von Kennzahlen zu quantifizieren. Die Generierung von Sichten auf ein Architekturmodell ermöglicht das Fokussieren auf einen bestimmten Aspekt. Ein Beispiel hierfür ist das Eingrenzen des Modells auf die entsprechenden Geschäftsprozesse und Anwendungen einer bestimmten Organisationseinheit des Unternehmens.

Aktuelle Ansätze zur Analyse von Unternehmensarchitekturmodellen stehen typischerweise für sich alleine und erfüllen nur eines der vielen Analyse-Ziele. Ein integrierter Ansatz, der die unterschiedlichen Informationsbedarfe der Stakeholder berücksichtigt, existiert nicht. Zusätzlich sind die bestehenden Ansätze abhängig von dem verwendeten Metamodell. Da die EAM Domäne von einer Vielzahl von Frameworks und Metamodellen geprägt ist, erschwert dies die Entwicklung eines universell einsetzbaren Ansatzes.

In dieser Arbeit stellen wir ein allgemein einsetzbares Framework zur Unterstützung der verschiedenen Analyseaktivitäten im EAM vor. Wir entwickeln geeignete Techniken zur Spezifikation und Ausführung der Analysen, unabhängig vom verwendeten Metamodell. Zur Definition und Anpassung der Analysen an die konkreten Bedürfnisse der Stakeholder wird eine Analysesprache entwickelt. Im Vordergrund stehen dabei Wiederverwendung und eine generische Definition der Analysen. Durch den Einsatz eines generischen Metamodells als Grundlage, sind wir in der Lage von der Vielfalt an Metamodellen zu abstrahieren. Die Analysen werden mit Hilfe der Semantic Web Technologien sowie einer Datenfluss-basieren Modellanalyse ausgewertet.

Das Framework wird zur Identifizierung von Schwachstellen sowie zur Bewertung von Planungsszenarien, hinsichtlich der Konsistenz der Änderungen und der Zielerreichung, eingesetzt. Hierfür werden zwei entsprechende Methoden entwickelt, der Analysebedarf identifiziert und mit Hilfe des Frameworks umgesetzt.

Die Abdeckung des Frameworks hinsichtlich der existierenden EAM Analysen wird mit einer Szenarien-basierten Evaluation bestimmt. Die Anwendbarkeit und Relevanz der Sprache sowie der vorgestellten Methoden wird in drei verschiedenen größeren Fallstudien gezeigt.

Acknowledgments

During the last years I spent on research and writing the thesis I received great support from several people.

First of all I want to thank my supervisor Prof. Dr. Bernhard Bauer. I am very grateful for the time he spent with discussions about my research as well as his guidance during writing this thesis. I also want to thank Prof. Dr. Alexander Knapp and Prof. Dr. Bernd Heinrich to be the further advisors of my thesis.

In addition I want to thank my colleagues at my research group *Software Methodologies for Distributed Systems*. I received valuable feedback at the workshops and the great working atmosphere supports the development of the thesis concepts. In specific, the successful cooperation with Christian Saad within several projects in the last years provided valuable results for my thesis. Also the research results gained together with my master and bachelor students improved the outcome of this thesis.

Finally, the industry partners within the different projects of the last years enabled a deep insights in current problems as well as the currently available solutions to deal with them. I am very thankful for this opportunity as well as the availability of case study data, which increases the quality of this thesis significantly. In specific I want to mention Andreas Ditze from the MID GmbH for making their modeling tool available for my research.

At least I want to thank my family and friends for the encouragement during the last years. Especially their support within the last month of writing was very helpful for me to concentrate on the thesis. I would like to thank Stephan for the time spent with discussions about my thesis. And my mother Brigitte and my sister Mareike for their tireless support in looking after my son. I also want to thank my father for his support during thesis writing as well as with the papers in the last years.

Finally, I want to thank my husband Dirk for his support in every aspect. He and my child Fabian were the best distraction from my thesis work and allowed me to clear my mind.

Contents

Part I

Introduction and Basics

1
Introduction

1.1 Problem and Challenges

Today's information technology (IT) landscapes in organizations are typically the product of evolved structures [AKRS08], extended with additional parts through merger and acquisitions. Since IT was not always seen as critical factor, its management and alignment were partially unattended. This faces organizations with the problem of a high complexity and heterogeneity regarding their IT landscapes. One of our case studies, the backend system of a large customer service application, comprises over 450 services with about 650 usage dependencies. The services work together to provide customer functionality within over 140 application scenarios. Thereby, about 20 different communication technologies are used, and the services are subject to different security requirements regarding data transfer, encryption and authentication. The operator of the backend systems has a limited overview of the dependencies between the services and fulfilling the overall goal to obtain a stable and performant system is critical. Exemplary challenges that occur during system operation are the failure of a system, performance problems and also architecture planning is hard. Within a survey 76% of the CIOs (Chief Information Officer) mention complexity as one factor that affects the provisioning of a stable IT system. The related costs to solve the performance problems are on average $ 2.5 million per organization [Dyn19b]. The challenges IT departments have to address let them struggle with the implementation of new business demands. Nevertheless, digital transformation forces organizations to adapt their business models and restructure their IT infrastructure in order to stay competitive. Microservices are a promising architectural style to create scalable, reliable and flexible IT systems [HS17]. A shortcoming of this technology is the increasing complexity, since one monolith is replaced with hundreds of independent services that communicate via message massing [DGL+17, GCF+17].

Enterprise Architecture Management (EAM) has been proposed as a way to manage large IT landscapes and the organizational changes with their inherent complexity. It provides means to capture the essential business and IT elements as well as the dependencies between them. This provides a clear understanding of the structure and enables an organization-wide optimization of the architecture [Lan12]. Therewith, EAM addresses the challenges of Business-IT Alignment and the development of new businesses models. An optimal alignment of business and IT is crucial for the success of any organization. And while analyzing the dependencies from business down to the IT infrastructure, EAM reduces the risks during enterprise transformations [NFT+17]. To support the different arising requirements from the stakeholders, an integrated set of methods is required for the documentation and analysis of the Enterprise Architecture (EA) [Lan12].

One reason for pitfalls of EAM projects are outdated and unused EA models [AKRS08]. An EA model is the central element within EAM, and used to capture the business, application and infrastructure elements of an organization. The models are the foundation for the subsequent activities including EA analyses. These analyses increase the understanding of the architecture and provide aggregated information to the management, e.g. through a dashboard. Through an evaluation of the current and target architecture as well as potential change scenarios, they support decision-making and architecture evolution [SK11].

A preliminary for performing analyses is that the relevant data about the enterprise architecture has to be captured within a model. Only then, an in-depth understanding of the architecture can be provided using different analysis approaches. These approaches enable

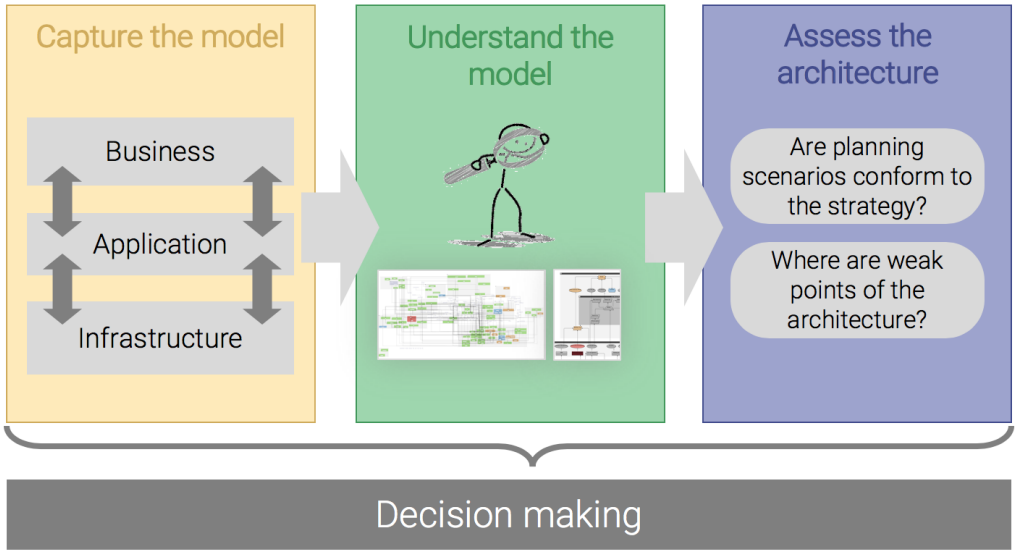


Figure 1.1: Analyses as foundation for decision-making.

the assessment of the architecture to identify weak points or to evaluate different planning scenarios regarding their strategy conformance. Figure 1.1 illustrates the dependencies between these three steps required for decision-making. In the following, the challenges of each step are described in detail.

1.1.1 Capture the architecture model

A major pitfall for current EA initiatives are outdated and unused models [AKRS08]. The low quality of the provided information within the EA models makes it difficult to generate benefit from them. Reports which include retired applications or outdated information cannot be used as foundation for decision-making. This leads to the point that the EA models are not used during planning tasks. Additionally, EA models are mostly created manually which requires much effort in terms of cost and time [RHF+13]. Thus, EA models currently deal with a low cost-benefit ratio that hampers the acceptance of the EA initiative and demotivates employees to maintain the architecture model. Considering the concerns from the different stakeholders would increase the cost-benefit ratio and enhance the acceptance of the EA initiative [AKRS08].

Challenge 1: Unexploited existing models

The first challenge addressed in this thesis is to make use of the established EA models in order to answer the different concerns of the stakeholders. To increase the benefit of the models, they have to be used for decision-making and planning purposes. Fulfilling the information demands of the stakeholders increases the acceptance of EAM and the responsible stakeholders are more likely to maintain the models. Thus, the quality and the completeness of the models can be increased. It is important that any approach should consider the high number of existing models and information for further processing.

The concepts used to describe an EA model are defined within an EA meta model [BBJ+11]. There is no consensus about the concepts described within an EA model, thus there exist a plethora of different EA meta models and no standard [BBJ+11,JNL07]. Several EA frameworks propose elements and layers to describe an enterprise architecture. These are well-known generic frameworks like Zachman [Zac87] and The Open Group Architecture Framework (TOGAF) [The18]. There exist also domain specific frameworks like the Department of Defense Architecture Framework (DoDAF) [Dep10] for the military domain or the Federal Enterprise Architecture Framework (FEAF) [Exe12] for US federal agencies and other governmental agencies. The frameworks differ widely regarding their concepts but also their coverage [BBJ+11]. In current practice they are often only used as starting point and further adapted to the individual needs of the organization according to the specific stakeholder needs [BBJ+11, AKRS08]. It is obvious that this variety can easily lead to problems, when applying techniques that aim to capture the information from enterprise models. Existing analysis and modeling approaches also rely on a specific meta model [KA09], an adaption of them to a specific EA model requires high effort.

Challenge 2: Different meta models

The second challenge is to provide analysis techniques with a widespread use, since the EA domain is characterized by a large variety of meta models. The overarching commonalities of the different meta models should be utilized to provide a solid foundation for further processing. It is important that such an approach also retains the specifics of each meta model, while decreasing the adaption effort to a minimum. Additionally, such a foundation should provide enough structure to enable stakeholder-specific analyses of the model.

1.1.2 Understand the architecture model

The decision-making process within EAM is supported with EA analyses [JLNS07a]. The results, for example metrics or views, are important to understand the architecture and its weak points. The analysis activities during EAM are unforeseeable, since they are not known in detail at the beginning [NSV15]. Changing business or IT requirements as well as strategic changes trigger changes in the analysis necessities. For example, a new strategy will lead to new goals for business as well as IT. Goals are often monitored using Key Performance Indicators (KPIs), thus, changing the goals requires an adaption of the KPIs. Additionally, generated analysis results provide new information which influences the future proceeding. For example, if a severe impact is calculated for a technology change, the architect wants to analyze the impact in detail.

Visualization and analysis of EA models cannot be done manually [Lan12]. To master the complexity of EA models an automated analysis of them is required. Additionally, they have to serve a large diversity of different requirements. The various different stakeholders like business architects, IT architects, enterprise architects but also management roles like Chief Executive Officer (CEO) and Chief Technology Officer (CTO), have different requirements when assessing the EA model. To ensure a high acceptance of the EAM initiative all different information demands have to be considered. CEOs and CTOs are more interested in quantitative measures to determine the current goal fulfillment and in high-level overviews. Business and IT architects have a high interest in detailed context views of their responsible business process or IT system to understand the dependencies.

Challenge 3: Varying and changing analysis needs

Challenge three demands for an automated and flexible analysis approach that is able to deal with the various different requirements of the stakeholders as well as with the high amount of different EA meta models. It is important that the stakeholder itself can make adaptions and configure the analysis according to his current needs in an easy way.

EA models are typically very large and complex models and created by using different, often disconnected modeling languages [NSV15]. Current trends like microservices increase their complexity and enhance the need for proper analysis support. In related work a plethora of different EA analysis approaches can be found. The approaches fulfill various different goals and use a wide selection of techniques to reach them [RLB17]. Each analysis approach typically serves a specific use case and incorporates a specific implementation. Its adaption to other use cases or its use within different EA models is a complex task and makes reuse difficult.

Current approaches that try to cover different analysis types are rare. The majority of the approaches are isolated ones that cannot be related to each other. A uniform interface to the different EA analyses is missing. Only a few approaches (e.g. [KA09]) deal with the indirect relationships in EA models. [SKR13a] addresses EA analysis using ontologies. Several approaches utilize probabilistic techniques (e.g. [NSJ+08, FFJ09]). But none of these approaches is able to cover the various different goals that are addressed by EA analyses. Within current approaches only little attention is spent to recursive analysis definitions, cyclic dependencies as well as incomplete models.

Challenge 4: Different analysis approaches

Challenge four describes the necessity to integrate different analysis approaches into one unified approach [NSV14, BMS09, JNL07]. Such an integrated analysis support has to include comparisons of different architecture states or scenarios, impact analyses of changes or failures and also the quantitative assessment of the model [Lan12]. This enables the combination of different approaches in order to receive more expressive analyses, as proposed by [NSV15]. Such a unified approach eases the development of a common analysis catalog as proposed by [LJW+16].

1.1.3 Assess the architecture model

A major use case for EA models is their utilization for planning and optimizing of the IT landscape and its business support. Through merger and acquisitions and the long-term growing IT landscapes, today's enterprise architectures are characterized by redundant IT support and a large heterogeneity. The overall goal is to retrieve a redundancy free, homogeneous, integrated and consistent architecture [Nie06]. Therefore, weak points have to be identified to determine optimization potentials and planning scenarios. Thereby, a major problem is the large amount of information provided by the EA models [ACS15]. Only a subset of this information is required for answering the current questions. It is important to identify the relevant subset, i.e. part of the architecture, and provide them, enhanced with further information from analyses, to the architects.

While implementing a desired target architecture, organizations are faced with new business and technology demands [The18]. Demands can be driven by new technologies and trends, the need for cost reduction or the integration of standards. A current example is microservice architectures which require new built, deployment and monitoring procedures.

Sources for business-driven demands are for example new business developments, new innovations and strategy changes. Additionally, the execution of legal regulations like Basel IV for the financial sector or merger and acquisitions lead to changes in the architecture. Such changes do not necessarily conform to the architectural strategy, i.e. the specified principles and goals. The resulting phenomena of a moving target is known problem within EA planning [Nie06,AGSW09]. Planning scenarios have to be integrated within the current EA strategy, i.e. to plan and execute them in an EA compliant way [Nie06,FB08,ASML12]. Scientific approaches to evaluate the fitness of those scenarios have several weak points [ACS15]. They are not widely adopted in current practice, often because of insufficient data quality but also because practical approaches, especially for comparisons, are missing [NFT+17]. Only if a scenario follows the defined strategy, the benefits from the EA initiative and the desired target architecture will be reached.

Challenge 5: Stakeholder-specific views and evaluations

Challenge five addresses the importance of providing the relevant information for the different arising questions from the stakeholders [ACS15]. Due to the importance and complexity of the IT landscapes a systematic approach is required to identify weak points and to evaluate planning scenarios. Adequate evaluation possibilities are required to decide about the fitness and conformance of a planning scenario regarding the current strategy [ACS15]. For an effective use of the available resources, the identification of weak points using quality attributes of the architecture like heterogeneity, goal fulfillment or non-redundancy is important.

1.2 Objectives, Approach and Contributions

In the following we present the four major objectives we want to address within this thesis. The objectives are defined in response to the presented problems and challenges in the previous section.

1.2.1 Generic representation

Creating an EA model requires much effort. Thus, it is important to use the already captured information for planning purposes and decision-making (challenge 1). Within challenge 2 we examined the large variety of EA meta models that are used in current practice. To address both challenges, we define the following first research objective.

Objective 1 Enable a generic representation of the various different EA meta models that enables later analysis execution. Additionally, provide support for the integration of models from different information sources.

Approach For the creation of such a generic representation we will, in a first step, examine existing EA frameworks and meta description languages to extract commonalities between them. Additionally, we identify requirements that should be met for later analysis execution. Based on this information a generic representation is developed.

Contribution As result we provide the Generic Meta Model (GMM) which can be used for the representation of EA models, independently from the utilized meta model or tool. The GMM provides enough structure to be used within later analyses. Additionally, the developed import concept enables a fast integration of existing sources into the GMM.

1.2.2 Generic architecture analysis framework

Understanding the enterprise architecture is essential for an effective decision-making. EA analyses are an important mean to evaluate the architecture and provide the stakeholder with the relevant information. It is essential to consider the large variety of different EA meta models (challenge 2) in order to provide a solution that can be generally applied. Challenge 3 describes the large variety of requirements of the stakeholders as well as their variation over time. This requires a customization of the analyses to be able to fulfill the changing needs of the stakeholders. There exist several analysis approaches for EA that support different goals and provide different types of information. A major challenge is their integration into a single analysis framework. This would enable more powerful analyses and the user will be able to fulfill his information demands. These issues lead us to the second research objective:

Objective 2 Provide a generic architecture analysis framework that is independent from the utilized EA meta model. The framework should provide a unified interface to different analysis approaches addressing different goals and enable customization of the analyses by the user.

Approach To fulfill this objective, we first analyze current EA literature and identify the different analysis types. The different types are categorized according to their functional and technical dimensions and typical characteristics for each dimension are derived. Based on these results, we develop an EA analysis definition language. The language allows the architect to define and customize analyses, while abstracting from the technical details. Examples for supported analyses are the determination of performance indicators or views. To execute the specified analysis definitions, an executor transforms them into processable rules. In order to support a broad spectrum of EA analyses a combination of SPARQL (short for SPARQL Protocol And RDF Query Language), a query language for RDF data, and a data-flow based approach is used for the execution. For data representation and as foundation for the analysis definition language, the generic meta model (objective 1) is utilized to enable the generic application of the framework.

Contribution The design artifacts for this objective are a declarative EA analysis definition language for the specification and customization of analyses and an analysis framework that enables the execution of those analysis definitions. The analysis language also supports the definition of generic analysis templates to support reuse. The overall definition and execution of the analyses is independent from the used tool and meta model in a specific scenario.

1.2.3 Employ framework for architecture assessment

For proper decision-making support it is essential to provide the required information within an architecture assessment as described in challenge 5. Otherwise the stakeholder will be flooded with information and it is hard for him to extract the important one. We will assess this issue of providing the relevant information within two use cases. The first one addresses the evaluation of the conformance of planning scenarios to the current goals and strategy during EA planning. The second one deals with the identification of weak points in the architecture in order to support its optimization. The third research objective regarding the architecture assessment comprises:

Objective 3 Provide a method and the required analysis templates to validate the conformance of planning scenarios. Moreover, provide a method and analysis templates for the identification of weak points within an enterprise architecture.

Approach For the development of the methods we first analyze current literature about EA planning and weak points and extract the common method blocks for the respective assessment. Based on these results we identify automation potential within the blocks and develop a (semi-) automated analysis approach. The required analysis templates are defined using the previously mentioned analysis framework. It provides the possibilities to perform different kinds of adaptable analyses as well as it is independent from the EA meta model used in an organization.

Contribution As contribution two practicable methods for architecture assessment are developed. A method supporting the evaluation of conformance of planning scenarios with the current strategy and a method for the identification of weak points within an

architecture. The analysis support for these methods is provided with flexible analysis templates that enable their reuse and adaption to existing EA approaches.

1.2.4 Evaluate the methods and the framework

To verify the goal achievement an evaluation of the created artifacts is required. Thereby, it is important to ensure the applicability and usefulness of the proposed contributions. This is addressed within the forth objective:

Objective 4 Provide a scenario-based evaluation of the analysis language and execution framework and apply the developed methods within several case studies.

Approach The analysis language and the execution framework are evaluated towards its desired characteristics of being an integrated, applicable and generic approach towards EA analysis. Therefore, its completeness regarding supported analysis types is discussed. The generic applicability and the coverage are evaluated through a scenario-based implementation of different analyses using three different EA models. The applicability is shown through the implementation of the two proposed methods using this framework and their application in three additional case studies. This application is used to show the flexibility and adaptability of the approach as well as the usefulness of the provided results for decision-making.

Contribution As outcome the practicability of the proposed methods for architecture assessment and the configurability of the provided analysis execution framework is shown in different use cases. Regarding the analysis framework its generic applicability is proved. Therefore, several analysis scenarios are defined and implemented with the framework. Finally, the fulfillment of the design goals for the analysis framework are discussed.

Figure 1.2 provides a summary of the objectives of this thesis and their addressed challenges as described in this section.

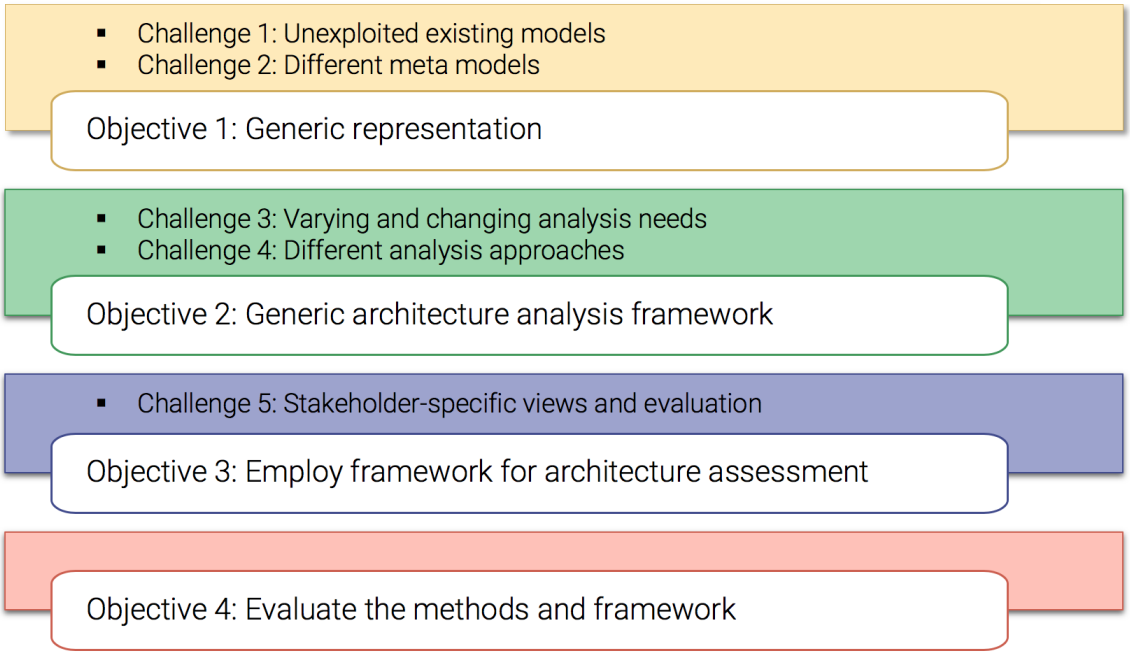


Figure 1.2: Objectives of this thesis with their addressed challenges.

1.3 Research Methodology

The artifacts of this thesis, described in the contributions of the previous section, are developed according to the design science research framework proposed by [HMPR04]. Figure 1.3 summarizes the application of this framework to our research goals.

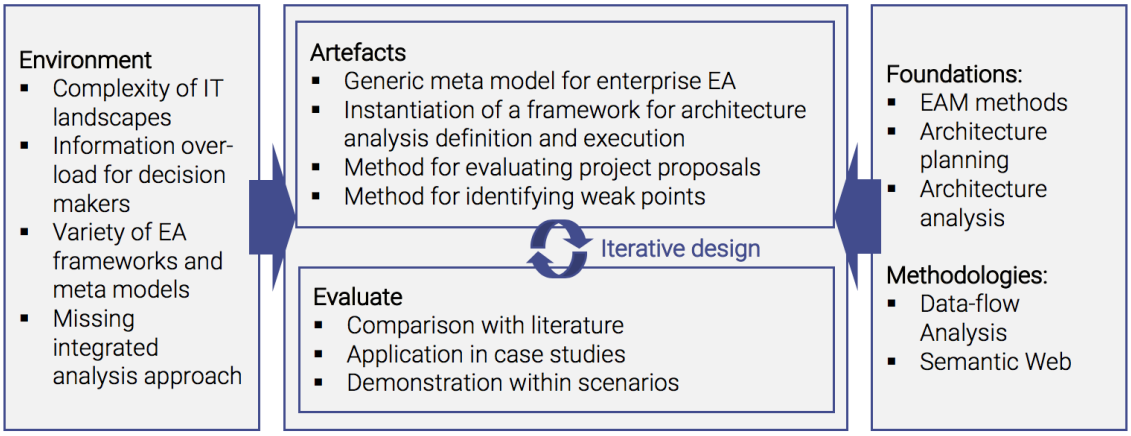


Figure 1.3: Research framework for this thesis according to [HMPR04].

Today's organizations are faced with the problems of high complexity and heterogeneity when regarding the IT landscapes. Trends like microservices and digital transformation foster the need for an effective management of the IT support to ensure their competitiveness. Enterprise architecture models and respective analyses are established to support decision-making in this context [JLNS07a]. Comprehensive analysis support is relevant in order to understand the organizational and IT structures. Therefore, the available information in the established EA models is processed to effectively support the various different stakeholders with their information demands. Several authors state the need for an integrated approach towards EA analysis [NSV14, BMS09, JNL07] and to address the missing applicability of existing planning approaches [NFT+17]. The plethora of different frameworks for enterprise architecture shows the necessity for providing a generic applicable method.

Rigor is achieved through an extensive review of existing methods for Enterprise Architecture Management including planning methods. Additionally, the existing instantiations of architecture analysis approaches for EAM are considered, for the later development of the proposed artifacts. The utilized methodologies data-flow analysis and semantic web are well-proven techniques for analysis execution.

Artifact construction and evaluation is done in an iterative process to make use of the feedback and insights provided during evaluation. Therewith the quality of the final artifacts can be improved and ensured. The major artifacts are an instantiation of the framework for analysis definition and execution, the meta model for generic representation of EA models and finally the two developed methods for architecture assessment. One for the identification of weak points and one for evaluating the conformance of planning scenarios.

For evaluation purposes the framework is implemented and applied within the two proposed methods. Therefore, different case studies are used to show the relevance of the proposed methods for decision-making and the generic applicability of the implemented

framework. Additionally, a scenario-based evaluation is employed to ensure the quality of the artifacts and a comparison with current approaches from the literature is used to show its coverage.

The results of our research are presented within technology-oriented as well as business-oriented workshops and conferences to enable further research and extend the knowledge base.

1.4 Publications

Parts of this thesis were previously published in several publications:

1. Langermeier M., Bauer B. [LB18a]: *A model-based method for the evaluation of project proposal compliance within EA planning.* In: 2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop.

   In this paper the authors propose a tool-supported method for EA planning to evaluate the project compliance based on established models. The developed method addresses objective 3 of this thesis. Thereby, different analyses are used to support the architect during project planning. Foundation of the method is the generic analysis architecture execution environment proposed in [LB17] that provides the required flexibility to adapt to different needs and meta models. The author of this thesis is the main contributor to the developed method. The results of this work are presented in chapter 6 and the presented case study is used for evaluation in chapter 8.
2. Engel T., Langermeier M., Bauer B., Hofmann A. [ELBH18] *Evaluation of Microservice Architectures: A Metric and Tool-Based Approach.* In: CAiSE 2018: Information Systems in the Big Data Era, LNBIP 317, Springer.

   In this work the authors propose an evaluation approach for microservice architectures based on identified architecture principles from research and practice. These are for example the small size of the services, domain-driven design or loose coupling. Based on a study showing the challenges in current microservice architectures, principles and metrics for the evaluation of the architecture design are derived. The metrics are used within the weak point identification in chapter 7. The required architecture data is captured with reverse engineering approaches from traces of communication data. The case study from this paper is used for evaluation in chapter 8. The paper presents the results of a master thesis the author of this thesis supervised.
3. Langermeier M., Bauer B. [LB18b]: *Evaluating Project Compliance during EA Planning: A model-based semi-automatic method for Enterprise Architecture Planning.* In: 40th International Conference on Software Engineering Companion (ICSE '18 Companion). ACM.

   This paper provides a short version of the results presented in [LB18a], where the authors propose a model-based and tool-supported method for EA planning and in specific for the evaluation of project compliance.
4. Langermeier M., Bauer B. [LB17] *Generic EA Analysis Framework for the definition and automatic execution of analyses.* In: Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017).

   In this work the authors develop a language for the definition of EA analyses as well as an execution environment for their evaluation. Therefore, existing EA approaches within literature are studied and a common interface to analysis activities is established by providing a domain-specific language. To cope with the high variety of meta models in the EA domain, the framework enables a meta model independent

access to analysis activities. The major part of this framework was developed by the author of this thesis. The proposed framework is described in detail in chapter 5, the developed analysis language in chapter 4.

5. Rauscher J., Langermeier M., Bauer B. [RLB17]: *Classification and Definition of an Enterprise Architecture Analyses Language.* In: Business Modeling and Software Design, 6th Int. Symposium, BMSD 2016, LNBIP 275, Springer.

   In this work the authors examine the different EA analysis approaches according to their characteristics and requirements. For that purpose, they design a generic analysis language which can be used for their description. In order to manage the numerous approaches from literature they develop a categorization. The categories are created based on the goals, constructs and kind of results. They propose a two-dimensional classification into functional and technical categories. The goal is to provide a common description for EA analyses for an easy access to their goals and execution requirements. The paper is an extended version of [RLB16]. The author of this thesis was the supervisor of the corresponding master thesis for this paper. The results of this work are used to ensure the completeness of the proposed analysis framework during evaluation (chapter 8).

6. Rauscher J., Langermeier M., Bauer B. [RLB16]: *Characteristics of Enterprise Architecture Analyses* In: Proceedings of the Sixth International Symposium on Business Modeling and Software Design (BMSD 2016).

   This paper is the short version of [RLB17], where the authors analyzed current literature about EA analyses and propose a two-dimensional classification approach for them.

7. Osenberg M., Langermeier M., Bauer B. [OLB15]: *Using Semantic Web Technologies for Enterprise Architecture Analysis* In: ESWC 2015: The Semantic Web. Latest Advances and New Domains, LNCS 9088, Springer.

   In this paper the authors propose the use of semantic web technologies in order to represent the EA and perform analyses. They present an approach how to transform an existing EA model into an ontology. Using this knowledge base, simple questions can be answered with the query language SPARQL. The major benefits of semantic web technologies can be found when defining and applying more complex analyses. Change impact analysis is important to estimate the effects and costs of a change to an EA model element. To show the benefits of semantic web technologies for EA, we implemented an approach to change impact analysis and executed it within a case study. This paper presents the results of a master thesis the author of this thesis supervised. The results provided in this paper are used during the development of the generic meta model for EA and the model storage strategy (chapter 3 and section 5.3).

8. Langermeier M., Saad C., Bauer B. [LSB14a] *Adaptive approach for impact analysis in enterprise architectures.* In: Business Modeling and Software Design, LNBIP 220, Springer.

   In this paper the authors propose a generic, context-sensitive approach to the implementation of impact analysis. Impact analysis determines the effects of changes

or failures on other architectural elements. The proposed method relies on the technique of data-flow analysis to propagate the effects through the model. Since the analysis specification only relies on a set of relationship classes, it can be easily adapted to the needs of organization-specific EA meta models by providing custom mappings for the respective types. The author of this thesis developed the classification concept for the relationships and the functional logic of the required change propagation rules. The classification concept is integrated in the generic meta model approach of this thesis presented in chapter 3 and the impact analysis as part of the analysis framework is described in section 4.3 and 5.5.

9. Langermeier M., Saad C., Bauer B. [LSB14b]: *Context-Sensitive Impact Analysis for Enterprise Architecture Management.* In: Proceedings of the Fourth International Symposium on Business Modeling and Software Design (BMSD 2014).

   This paper provides a short version of the results presented in [LSB14a]. and presents a context-sensitive approach for the implementation of impact analyses.

10. Langermeier M., Saad C., Bauer B. [LSB14c]: *A unified framework for Enterprise Architecture analysis.* In: 18th IEEE International Enterprise Distributed Object Computing Conference Workshops.

    In this paper the authors propose a meta model independent framework for the analysis of architecture models. Based on a generic representation of architectural data, they employ a data-flow based analysis approach to enable a context-sensitive evaluation of organization specific measures. The generic applicability of this framework is demonstrated through a re-implementation of three different analyses approaches from related work. The author of this paper developed the transformation concept for EA models into the internal representation and the specification of the rules for re-implementation of existing analysis approaches. An extended version of the proposed concept in this work is presented as generic meta model in chapter 3. The analysis techniques used for analysis execution is integrated into the analysis framework in chapter 5.

11. Lautenbacher F., Diefenthaler D., Langermeier M., Mykhashchuk M., Bauer B. [LDL+13] *Planning Support for Enterprise Changes.* In: The Practice of Enterprise Modeling, LNBIP 165, Springer.

    In this work the authors want to improve the current manual creation of the transformation paths in enterprise architecture planning by providing possible and sound sequences of actions as part of a road map from the current to a desired target architecture. Therefore, they present a solution that supports the enterprise architect with proposals for a transformation path from the current to the target state considering also dependencies during the enterprise transformation. This includes an identification of successor relationships between different architectural states and if necessary the architecture can be divided into smaller segments. The author of this thesis participated especially in the segmentation analysis and the identification of successor relationships. Both results are used for the provisioning of different analysis types within the analysis framework in chapter 5.

12. Chen W., Hess C., Langermeier M., Stülpnagel J., Diefenthaler P. [CHL+13]: *Semantic Enterprise Architecture Management* In: Proceedings of the 15th International

Conference on Enterprise Information Systems (ICEIS 2013).

The authors propose in this paper the use of semantic technologies for integrating heterogeneous EAM relevant information sources. They define a process for building an EA repository that uses linked data within an enterprise. Thereby, existing enterprise data sets are exposed as ontologies and linked into the enterprise architecture. The approach is demonstrated by linking EA documentation (according to TOGAF), business process models (in BPMN) and aspects on services (according to the SOA ontology) with each other. The author of this thesis participated in developing the ontologies and the mapping method. Experiences from this work influenced the approach for integrating and representing the architectural models within this thesis, presented mainly in chapter 3.

1.5 Outline of this Thesis

The structure of this thesis is summarized in figure 1.4.

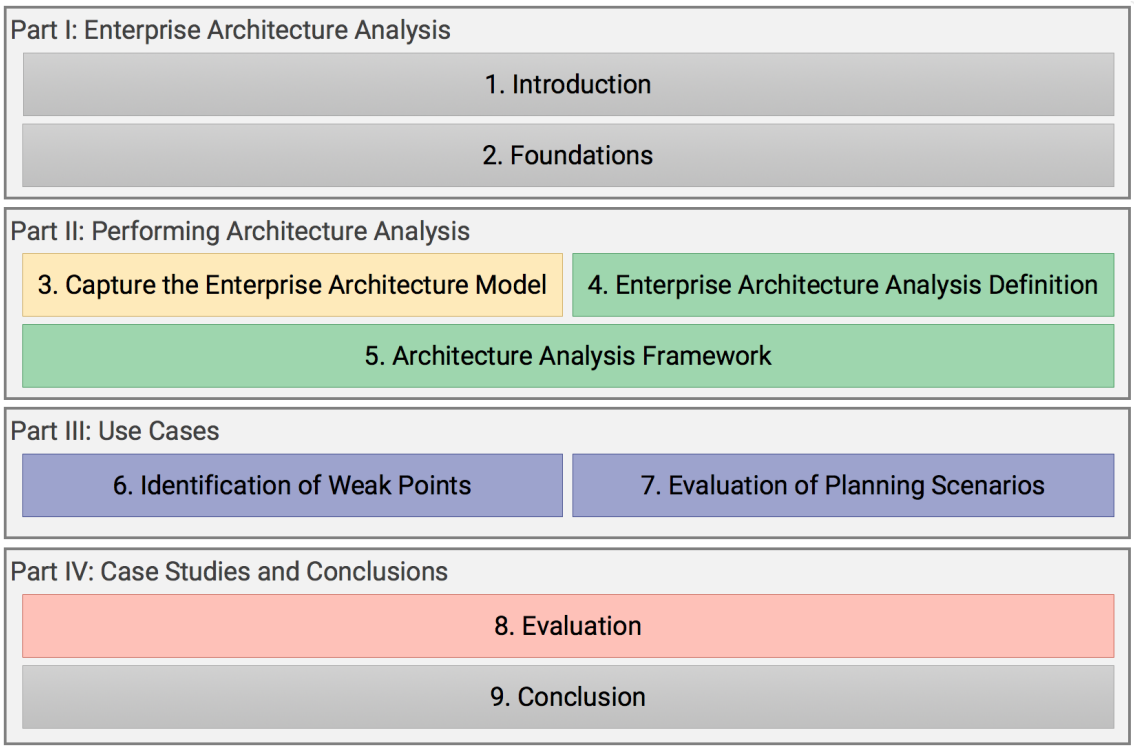


Figure 1.4: Outline of this thesis.

Chapter 1 This thesis started with a description of the problems within the enterprise architecture domain and the current challenges while addressing them. In the following the three research objectives of this thesis were presented together with the approach and main contributions to address them.

Chapter 2 In the next chapter the foundations for the subsequent work are introduced. This encompasses the domain of Enterprise Architectures with its processes, analyses and planning methods as well as the basics of data-flow-based analysis and semantic web technologies.

Chapter 3 In chapter 3 one part of the main contribution addressing objective 1 is described. We introduce the generic approach for representing Enterprise Architecture models and propose techniques to capture the required data for its construction.

Chapter 4 This chapter focuses on the different analysis approaches used for an EA assessment. To provide a universal interface to the analysis activities a domain specific language is developed. This language enables the definition of the analyses from a functional perspective. The description of the supported analyses is illustrated with a running example.

Chapter 5 In chapter 5 the architecture analysis framework for performing EA analyses is proposed. It builds upon the results of chapter 3 (generic meta model) and chapter 4 (analysis definition language). The execution strategies for the different analyses are described as well as details for model storage and analysis definition.

Chapter 6 In chapter 6 the method for the identification of weak points is presented. The method encompasses the activities that have to be done and describes how they can be supported by the analysis execution environment presented in chapter 5.

Chapter 7 This chapter describes a practicable approach for EA planning, in specific for evaluating the conformance of planning scenarios with the current strategy. Based on common method blocks within current EA planning approaches, tool support is developed by utilizing the proposed analysis framework.

Chapter 8 Chapter 8 comprises the implementation of the analysis framework and the evaluation of the contributions from the previous chapters. The analysis framework with the analysis language is evaluated within different scenarios. The two proposed methods, weak point identification and EA planning, are applied within different case studies. Finally, a discussion of the results is given.

Chapter 9 A summary of this thesis is provided in chapter 9. This includes a short description of the main contributions of this thesis and concludes with an outlook for future work.

2
Foundations

2.1 Enterprise Architecture

According to the ISO/IEC/IEEE 42010:2011 the term architecture is defined as the

> "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [ISO11b].

The term enterprise can be understood as collection of organizations that have common goals [The18, Lan12]. Examples of an enterprise are a whole corporation or only a part of it, a government agency or department but also partnerships or alliances that are working together [The18]. Just like buildings every enterprise has an inherent architecture. This enterprise architecture covers the business and operating model, the organizational structure, business processes, data, applications and the IT infrastructure of the organization as well as the design rules for their future development. [ASML12, Lan12, WF06]. Enterprise Architecture (EA) is an approach for managing and planning the information systems of an enterprise. Tools and methods are provided to support decision-making in this context [JJ05].

The EA provides a multi-layer representation of the organization [WF06]. The different layers depend on each other according to the *align-enable* principle (see figure 2.1). The

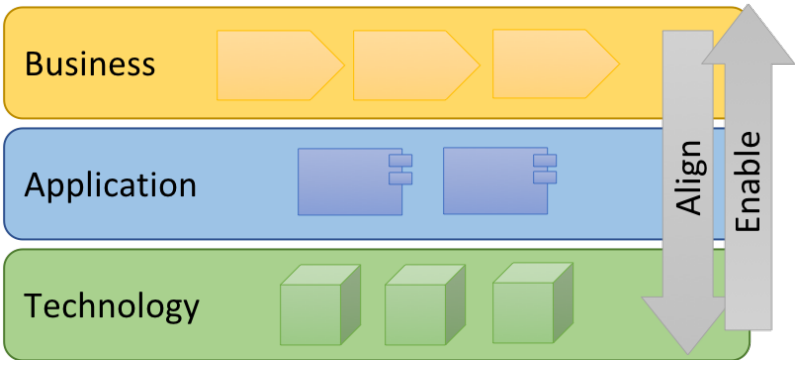


Figure 2.1: Core layers of an enterprise architecture according to [The17].

lower layers enable the functions of the higher ones, whereas the higher ones coordinate the lower ones [Krc05]. There is no common agreement on a specific set of layers [ASML12]. ArchiMate [The17], a modeling language for enterprise architectures proposed by The Open Group, distinguishes three core layers: the business layer, the application layer and the technology layer. In the business layers the processes, services and products that are realized within and provided by the organization are described. The application layer describes the application components and services that are required for the provisioning of the business elements. Finally, the technology layer focuses on the hard- and software components required for processing and for communication between the application components. Additional layers can be used to describe the strategy, physical objects or implementation and migration planning [The17].

In [WF06] the authors identify essential elements of an EA from a business-related perspective. They conclude with five layers: Business and also strategy elements are described in a business and process architecture. The information systems and respective software

artifacts are described in the integration and software architecture. Finally, the technology architecture describes the computing and communication hardware as well as networks.

The enterprise architecture contains only the aggregated artifacts of each layer, e.g. the application systems itself. The details, for example a component diagram describing the software architecture of this system, can be found in specialized architectures for each layer. Only the top-level abstraction is used within in the enterprise architecture. [WF06]

In addition, Enterprise Architecture Management (EAM) describes the respective management discipline which

> "maintains and uses a coherent set of guidelines, architecture principles and governance regimes that provide direction for and practical help with the design and development of an enterprise's architecture in order to achieve its vision and strategy" [ASML12].

Therefore, the discipline uses the documented models of the current and target architecture and supports their future development, their realization and the operation of the business and IT [ASML12, Han13]. Thus, EAM is a mean for incorporating changes throughout the whole organization as well as for driving optimizations of the architecture, especially the alignment of business and IT.

With EAM a redundancy free, homogeneous, integrated and consistent architecture should be developed. This architecture conforms to the enterprise's strategy and goals [Nie06]. Despite improving the quality of the organizational structures and Business-IT-Alignment, EAM provides several benefits: It provides a common knowledge base which is used to improve decision-making [LJJ+06]. It enables enterprises to respond quickly to new demands [LJJ+06] and it reduces the risk and complexity within projects [FB08].

In the following *processes and frameworks* are presented and the three main activities *document*, *analyze* and *plan* are considered in detail.

2.1.1 Processes and frameworks

There are several publications from science and practice that describe processes or frameworks for EAM. An architecture framework describes "conventions, principles and practices for the description of architectures" [ISO11b]. A framework is always developed for a specific domain or community. Examples for well-known frameworks are the Zachmann Framework [Zac87] and TOGAF [The18]. Additionally, there exists framework for the military domain like the Department of Defense Architecture Framework (DoDAF) [Dep10], the Ministry of Defense Architectural Framework (MODAF) [Min12] and the NATO Architecture Framework (NAF) [Arc18]. These three frameworks rely on each other and have very similar concepts [Obj13]. The described concepts are not only applicable for the military domain but also in the business and public service domain [Obj13]. Additionally, the Federal Enterprise Architecture Framework (FEAF) is a framework specific for governmental organizations [Exe12]. Further propositions for frameworks from the industry are the Quasar framework [EHH+08] or the Integrated Architecture Framework (IAF) [vWWH+11].

The Zachmann Framework provides a 6x6-cell matrix that enables the representation of an architecture from various different views. The proposed layers differ between the *Executive Perspective*, the *Business Management Perspective*, the *Architect Perspective*, the

Engineer Perspective, the *Technician Perspective* and the *Enterprise Perspective*. Additionally, for each layer six different dimensions further distinguish between the *What, How, Where, Who, When* and *Why*. The Zachmann Framework provides no details for the implementation process or suggests specific modeling approaches or tools. The concepts of the Zachman Framework are also used in frameworks like FEAF, DoDAF and The Open Group Architecture Framework (TOGAF). [Zac87, Zac08]

TOGAF (The Open Group Architecture Framework) is an industry standard provided by The Open Group [The18]. The framework is the most known one [The19]. Within the standard the Architecture Development Method (ADM) is proposed. Additionally, guidelines and techniques for its application and an Architecture Content Framework, describing a meta model and the relevant artifacts, are presented. Two further parts of the standard deal with tools to categorize and store the results of the architecture activity and they discuss the relevant organizational structure, processes and skills in order to establish and operate an architecture initiative.

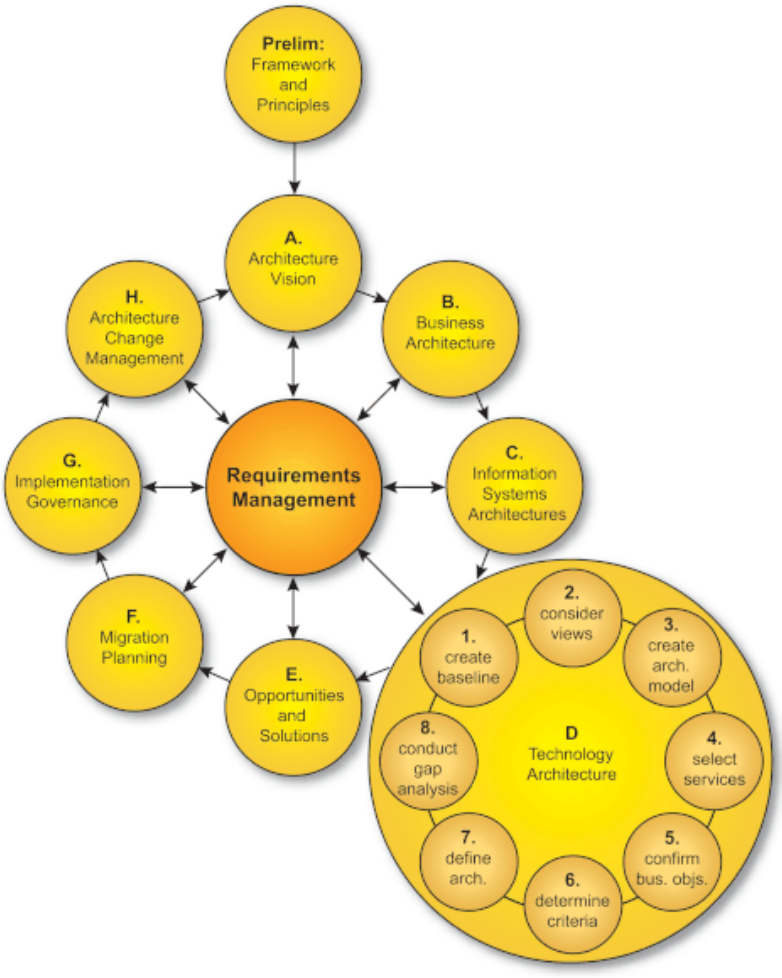


Figure 2.2: Method for the enterprise architecture development (TOGAF ADM) [The18].

The TOGAF ADM provides an iterative process for the definition and realization of an enterprise architecture, aligned with the principles and goals of the enterprise. The process is comprised of nine phases and the ongoing *Requirements Management* activity. Sec-

Table 2.1: Phases within the iterative process of the TOGAF ADM [The18].

Phase	*Description*
Phase A: Architecture Vision	Identification of Stakeholders and requirements, definition of the scope, development of a vision
Phase B: Business Architecture	Development of baseline and target business architecture description, identify gaps and impacts
Phase C: Information Systems Architecture	Development of baseline and target data and application architecture description, identify gaps and impacts
Phase D: Technology Architecture	Development of the baseline and target technology architecture, identify gaps and impacts
Phase E: Opportunities and Solutions	Definition of the architecture road map and transition architectures
Phase F: Migration Planning	Creation of an implementation and migration plan with project and portfolio breakdown
Phase G: Implementation Governance	Ensure conformance of projects with the target architecture
Phase H: Architecture Change Management	Monitoring and managing of change requests and risk management

tion 2.1.1 provides an overview of the phases as well as the detailed steps for phase D, *Technology Architecture*. For each phase the standard specifies the objectives, inputs and relevant steps to create the desired outputs. Beforehand in the *Preliminary Phase* the *Architecture Capability* within the organization is established. This includes for example the definition of the team and the specification of governance structures and architecture principles. The following phases A-H are described in table 2.1.

The outputs of Phase C and D are developed according to the *Business Architecture* and *Architecture Vision* defined in the earlier phases. Thus, the alignment of business and IT can be ensured. In parallel to all phases the *Requirement Management* takes place. Thereby, the requirements are captured, monitored and prioritized. This activity ensures that the relevant requirement specifications are available in the steps.

According to [ASML12] the EAM process integrates three different but interrelated cycles. The *Strategic planning cycle* starts with an analysis of the current situation and its documentation and concludes with a road map and project portfolio to develop the target architecture. Initiating a project, triggers the *Project life cycle* that deals with the set-up, design, implementation and roll-out of a concrete project. Finally, in the *Operation and monitoring cycle* the EA is monitored and changes are collected, assessed and implemented. During monitoring, essential information for the architecture evaluation is collected and provides feedback for the strategic planning.

[Nie06] summarizes the main activities for development and use of EA with the steps *document*, *analyze*, *plan*, *act* and *check* (see figure 2.3). Within the *document* phase the current state of the EA is captured and the respective model is created. In the phase *analyze*, analyses are used to determine weaknesses of the current EA model and to create visualizations. Within the *planning* step, scenarios are developed that represent possible target architectures. These scenarios are further evaluated in terms of risks, costs and their impact on the IT goals. *Act* comprises the implementation of the developed plan in

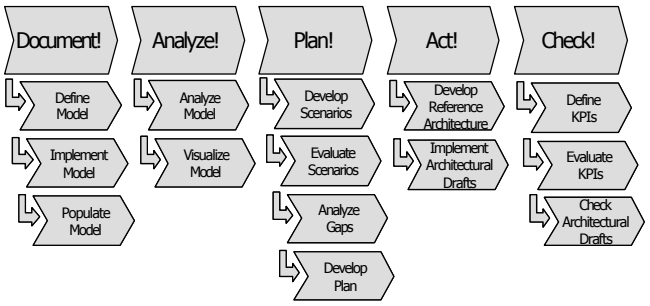


Figure 2.3: EA development processes according to [Nie06].

the previous step. The ongoing *check* activity ensures the progress of the evolution using performance indicators and Key Performance Indicators (KPIs).

In the following three chapters, the steps *document*, *analyze* and *plan* of the EAM cycle proposed by [Nie06] are considered in detail.

2.1.2 EA documentation

The aggregated structures of the organizations are captured in EA models and used for documentation, analysis and planning [AGSW09]. The term enterprise architecture describes the actual architecture of the real-world enterprise, the enterprise architecture model describes its documentation in terms of plans or models [ASML12]. The models are used to document the organization, its components and the relationship between those. Thus, they provide a mean to capture and understand the complex dependencies between the business and the supporting IT infrastructure [Lan12]. During EA documentation the current EA model is created (also named as-is model). This modeling task is preliminary for all ongoing activities especially for describing and understanding the EA [KAV05]. Models are also used to capture a desired target state of the EA (also named to-be model) [ASML12, AKRS08].

To manage the complexity of the typically very large EA models, views are established. A view describes the architecture (or a part of it) according to the concerns of a specific stakeholder [ISO11b]. Views for an EA can be layer-specific but also covering several layers [WF06]. ArchiMate proposes several viewpoints for typical stakeholder concerns. For example, the process architect is interested in the dependencies between business processes as well as their consistency, completeness and the responsibilities. The *Business Process Cooperation Viewpoint* serves these requirements, while focusing on the business processes, their required application systems, their provided services and involved actors. In contrast the *Application Structure Viewpoint* provides relevant information to the application architects regarding the applications with their sub components, their interfaces as well as communication and data dependencies. [The13]. The relevant concepts within this viewpoint are shown in figure 2.4.

Despite viewpoints, the concept of domain architectures is used within EA in order provide an architecture artifact with limited scope [BvSF+10]. Domain architectures can be created for major business processes, information systems or functional areas of an organization [Pul06, PH05]. In [BvSF+10] domain architectures act on an intermediate level between enterprise architectures and solution architectures. They provide a reduced scope and thus enable a stakeholder specific depiction of the architecture.

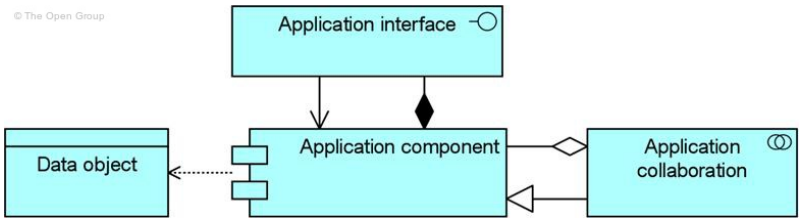


Figure 2.4: Application Structure Viewpoint [The13].

According to [BvSF+10] a domain is defined as an area that is of specific interest for a stakeholder in terms of relevance and/or ownership. It is important that all aspects that are required for the work of this stakeholder are captured by the domain. Domain architectures are not necessarily disjunct from each other. Viewpoints as proposed above can be used as foundation for the creation of domain architectures [BvSF+10]. For example, an application owner is only interested in the structure of his application component. Whereas the viewpoint presents all applications with their structure, within the domain architecture the scope of the viewpoint is set according to the specific application of the owner.

Enterprise architecture models can be informally described with text documents or drawings, often created with Microsoft Visio or Power Point, but also simply drawn on paper [ASML12]. These approaches cannot ensure the consistency between the elements [BW05]. Tabular representations are also used to store the EA information, often realized with Microsoft Excel, but are also still limited regarding view generation and analysis functionality. Specialized EA tools utilize a repository to store the EA artifacts to ensure the consistency [ASML12, BW05]. Additionally, they provide functionality to create reports and visualizations.

There are several relevant sources to create an enterprise architecture model. An important source is the implicit knowledge of the employees e.g. architects and project managers [RHF+13]. This knowledge can be captured with interviews, questionnaires or workshops and the identified data has to be captured manually. Most of the EA tools provide interfaces to import data from other applications via Structured Query Language (SQL) loaders or Comma-separated Values (CSV) importers. Relevant applications are configuration management databases, process management tools and project management tools [RHF+13]. Finally, relevant information can be retrieved from existing Excel files or databases.

The creation of an EA model is a very time and cost consuming task [HBLE14, KAV05, RHF+13]. And once the model is created, the actuality of the data is a major problem, organizations are faced with [RHF+13]. Recent approaches address this challenge while integrating monitoring data [FAB+11], data from network scanners [HBLE14] or data provided by an enterprise service bus [BHS+12] within the model.

Meta models for documentation

An EA model is documented using an organization specific set of concepts and relationships between those. The modeling concepts that are used to describe the models are defined in a meta model [KW07]. It is important to develop the meta model for an EA according to the needs of the stakeholders [KW07]. Current literature provides several different meta models for enterprise architectures (e.g. [The18, Dep10]). Additionally, the tools for EAM

often propose their own meta model (for example iteraplan [ite19] or leanIX [Lea19]). There exists no common standard and typically an organization chooses an existing one as starting point. The meta model has then to be adapted to the special needs of the organization [AKRS08].

In [AKRS08, KW07] a stakeholder-oriented method for meta model development is proposed. Based on identified requirements, several meta model fragments are developed and finally integrated into one meta model. A similar approach is proposed in [BEL+07]. The authors propose several patterns that represent model fragments. Each pattern addresses a specific concern. The final meta model can be created through selecting and composing the relevant patterns for the current challenges.

ArchiMate is a well-known modeling language for enterprise architectures published by The Open Group [The17]. The proposed meta-model of ArchiMate utilizes the concept

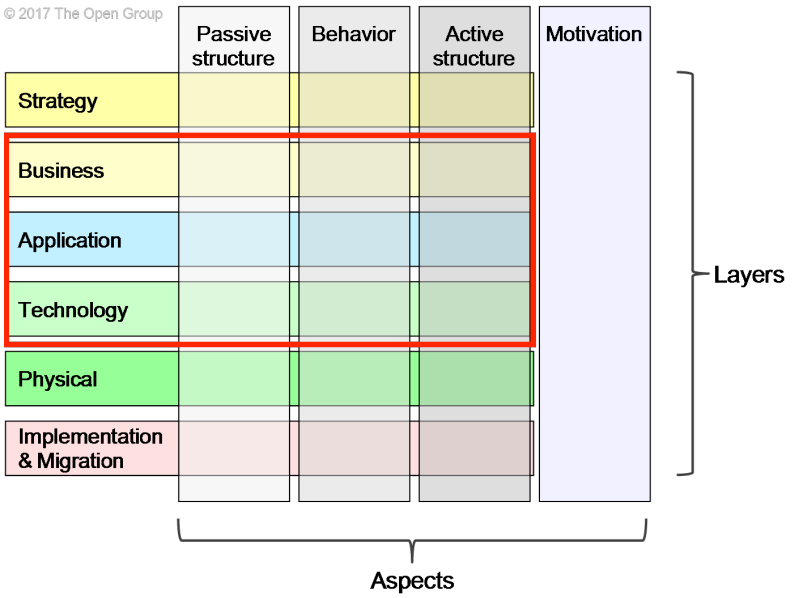


Figure 2.5: Full framework of ArchiMate [The17].

of service-orientation to represent the dependencies between the business, application and technology layer. Despite providing meta models for the different layers and specifying the interrelations between the layers, ArchiMate also proposes several viewpoints serving the different stakeholder needs. Figure 2.5 shows the full framework of ArchiMate. The framework describes the proposed layers in the language as well as the different aspects addressed within the layers.

The core of the framework includes the layers business, application and technology. The *business layer* describes the business services offered to the customers and their realization by actors according to specified business processes. The *application layer* describes the application services supporting the business processes as well as the applications realizing them. Finally, the *technology layer* describes technology services (e.g. services for storage and communication) required by the applications but also the required computer, computer hardware and software systems. The core framework can be further extended to enable modeling of strategical elements, physical elements and implementation and migration concepts.

The detailed meta model of the application layer is shown in figure 2.6. It contains ele-

ments like services, interfaces an application provides and the respective function, process or interaction a service implements. Additionally, the data objects used within the provisioning of the functionality can be described. Events and collaborations are used to model behavioral dependencies between applications, services and functions. Elements of the business layer make use of the described applications. Business functions and services can be realized by application services. Additionally, the elements of the application layer have realization and use dependencies to the elements of the technology layer.

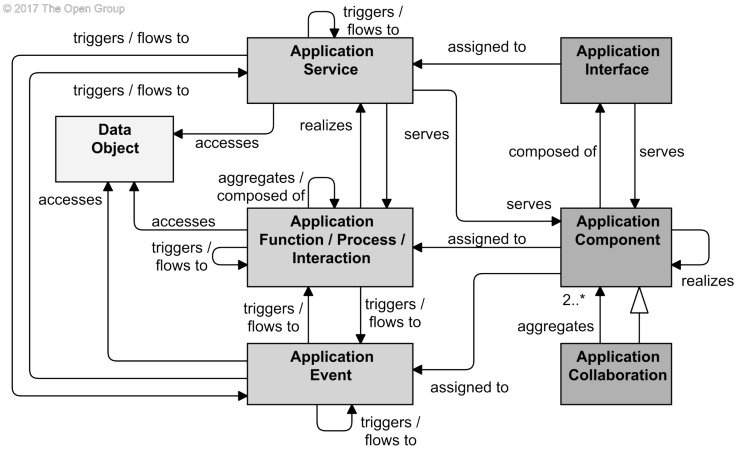


Figure 2.6: Meta model of the application layer of ArchiMate [The17].

Despite the layers ArchiMate distinguishes between four aspects. The *Active Structure Aspects* represent structural elements such as business actors and applications components (presented on the right side of the figures 2.5 and 2.6). Behavioral elements like processes, functions and services are represented within the *Behavior Aspect* (in the middle of the figures). And elements like data or information objects belong the *Passive Structure Aspect* (on the left side). These three core aspects can be extended with a *Motivation Aspect*.

Common concepts for EA

ArchiMate, and also other publications, identify common concepts or foundation concepts for enterprise architecture. The top-level hierarchy used by ArchiMate relies on a *Model* that consists of *Concepts*. These concepts can either be an element or a relationship. Elements are the fundamental components within ArchiMate. They are "used to define and describe the constituent parts of Enterprise Architectures and their unique set of characteristics" [The17]. Elements can be further distinguished between: *Behavior elements*, *structure elements*, *motivation elements* and *composition elements*. Behavior elements are used to describe the internal and external behavior as well as events. Structure elements include the active as well as the passive structure elements.

Additionally, ArchiMate differentiates between four different types of relationships within its hierarchy. The following list provides the types with the assigned relationships:

structural	realization, assignment, aggregation, composition
dependency	influence, access, serving
dynamic	trigger, flow
other	specialization, association

Despite the element and relationship hierarchies, ArchiMate also defines a generic meta

model (see figure 2.7). Within the meta model the main relationships that exist between the behavior and the structure element are depicted. Additionally, every element can have composition, aggregation, and specialization relationships to elements of the same type. [The17]

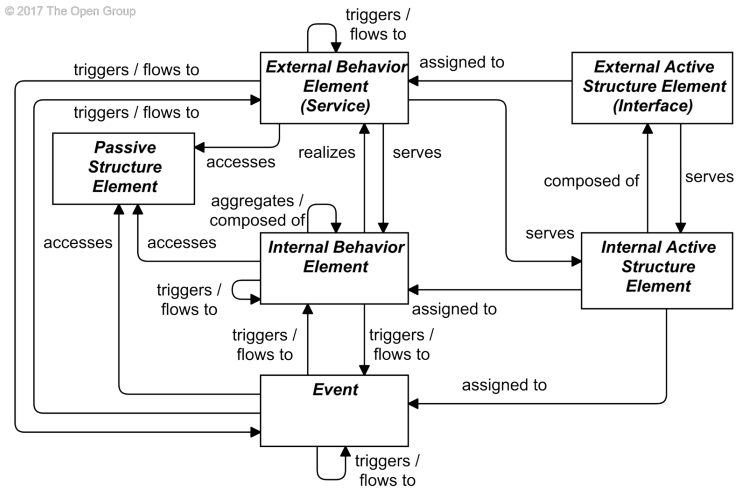


Figure 2.7: Generic meta model provided by ArchiMate [The17].

The DoDAF Meta Model [Dep10] is built according the International Defence Enterprise Architecture Specification (IDEAS) Foundation. Within the IDEAS Foundation top level elements and common patterns for relationships are defined. They are used to formalize the conceptual model which is created on a high level and in non-technical manner. Figure 2.8 presents an overview of the top level of the IDEAS Foundation. Top level elements are: *Thing*, *individual*, *type* and *tuple*. A *thing* is equivalent to the object in object-oriented models. The element *type* is used to provide another installation level for example to model several installed instances of an application system. Common patterns for data constructs that represent relationships are: Composition (whole-part), Super/sub-type, type-instance, before/after and overlap. The first three ones are used to describe relations according to object-oriented models. Before/after is used for time related dependencies. Overlap is used for the exchange of things or for things that occur at overlapping times or places. [Dep10]

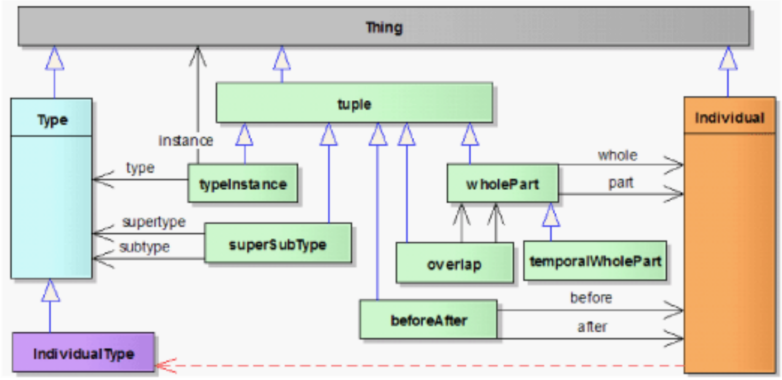


Figure 2.8: Elements and data constructs of the IDEAS Foundation [Dep10].

Finally, the Generalised Enterprise Reference Architecture and Methodology (GERAM) [IFI99] is a meta framework that describes general requirements for frameworks and tools used to maintain and design enterprises, i.e. enterprise engineering. Therefore, the framework describes components that are required in all types on enterprises. Enterprise Engineering tools implement an enterprise engineering methodology and an enterprise modeling language. These tools together with partial enterprise models are used to create enterprise models. The models represent the design of the organization and are used for analysis and to support the business operation. The generic concepts used within these components are human oriented ones, process-oriented ones and technology-oriented ones.

2.1.3 EA analysis

EA models are used within a plethora of different application scenarios. Among those are IT business alignment, project portfolio planning, business process optimization, sourcing decision and IT service management. To utilize the EA models in these contexts, the contained data has to be queried, summarized and interpreted with suitable analysis techniques [BFKW06]. For example, during project portfolio planning, analyses support the decision about approval or rejection of a project proposal. They are used to predict the effects of a decision and evaluate the quality of the future architecture or system [JJSU07]. Another application scenario for analysis techniques is the provisioning of a dashboard with performance indicators to support management decisions [FHK09].

Enterprise architecture models support the architecture process only in a visual and qualitative way [FFJ09]. Analysis techniques provide means to quantify models, predict future behavior and compare different alternatives. They make use of the contained information in order to support the different scenarios. Reconsidering the EA cycle of [Nie06] with the phases *document*, *analyze*, *plan*, *act* and *control*, they are an essential part in the overall EAM process.

As discussed in [LB17] EA analyses increase the understanding of the architecture and provide aggregated information to the management. Through an evaluation of the current and target architecture as well as potential change scenarios, they support decision-making and architecture evolution [SK11]. To cover the various application fields a plethora of different EA analysis approaches can be found in literature. The approaches fulfill various different goals and use a wide selection of techniques to reach them [RLB17]. For the implementation of the different analysis approaches different techniques are used in current research and practice. Existing approaches rely for example on techniques like the Extensible Markup Language (XML) [dBBG+05] or utilize probabilistic techniques like extended influence diagrams [JLNS07a], Bayesian networks [JJSU07] or a probabilistic extension of the Object Constraint Language (OCL) [NBE14]. In [SKR13b] SPARQL was proposed for analyzing an enterprise architecture.

Despite the analysis approaches within literature current EA tools provide analysis support. The analysis support is limited regarding the supported meta models and the technical analysis capabilities [NSV15]. In contrast to our understanding tool vendors often call customization functions for visualizations also an analysis. For example, the tool LeanIX enables *interactive analysis* which means activating and deactivating filter functions within visualizations [Lea19]. The majority of the tools provide support for metrics. This is realized by predefined ones, their creation with a user interface and a few tools also provide a domain specific language for the custom creation of them [HRSM13]. Ad-

ditionally, EA tools provide interfaces to integrate custom analyses or directly query the data in the storage, for example with an SQL interface (e.g. Alfabet [Sof19b]).

Currently, EA tools provide two major approaches to EA analysis: Either they are shipped with a predefined and static meta model (e.g. iteraplan [ite19], leanIX [Lea19]). Upon their meta model these tools typically provide several analysis techniques out of the box. In the other case the meta model of the tool can be customized to the organization needs (e.g. Alfabet [Sof19b]). In this case the analysis functions of a tool typically have also be adapted which is associated with a high effort.

In the following, the existing analysis approaches within literature are presented along five different analysis categories which address different aspects of an enterprise architecture respectively of the architecture process. Figure 2.9 provides an overview of the aspects and the analysis categories.

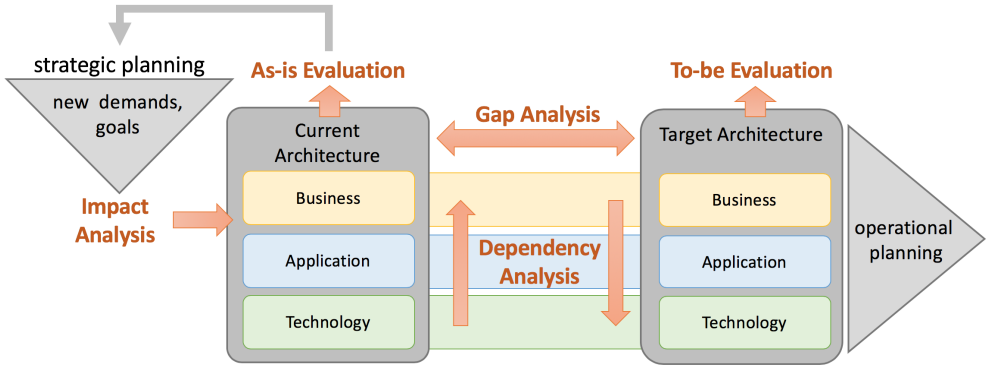


Figure 2.9: Analysis categories in the context of enterprise architectures.

According to [Nie06] the goals, principles and finally projects are defined within strategic planning. As foundation, the information provided by *architecture evaluations* is used. These analysis approaches address quality attributes of the architecture and identify strength and weaknesses. Typical analysis approaches are an analysis of the costs and benefits [Nie06], the performance [JI09], the interoperability [UFBJ10] or the modifiability [LJE10]. Additionally, this category encompasses conformance analysis to existing strategies and goals, typically realized with metrics.

Based on the results of the evaluation of the current architecture within strategic planning, new demands are identified. The effects of those are determined with *impact analyses*. If finally, a desired target architecture is created, the differences between the current and the target architecture are determined with a *gap analysis*. The target architecture itself is also evaluated with respect to the current strategy. Within the subsequent operational planning the proposed changes are implemented within the organization. A special category of analysis are approaches is *dependency analysis*. They focus on the direct and indirect relationships between the architecture elements within the layers and above them.

2.1.3.1 Dependency Analysis

A dependency analysis focuses on the relationships between the high-level business elements and the corresponding applications and infrastructure elements [FFJ09]. Thereby,

especially the indirect relationships between these elements are of interest. The information about them is also relevant within other analyses.

In [KA09] the authors propose a generic approach for a flexible dependency analysis. Thereby, they address current weaknesses regarding the flexibility of existing methods, relying on static meta model structures. Among current approaches there exists no standard evaluation and an adaption of the existing ones requires high effort. [KA09] propose a formal solution for flexible and generic dependency analysis. They define a formalism for a two-dimensional dependency analysis, through determining the transitive closure of the set of relations. Thereby, reflexive relation types are not considered. The derivation of composed relationships for reflexive dependencies is defined according to the semantic of the relationship. The authors describe two different semantics (direct dependency and indirect dependency) as well as a generic approach, using an expansion function for hierarchical refinement relationships. Finally, they describe a formalism for aggregating relationships to reduce the complexity of the models [KA09].

In [Saa10] the author proposes a dependency analysis approach considering time constraints (org. "zeitbezogene Abhängingkeitsanalyse"). Thereby, he considers the life time, the status (current or proposed) as well as the different life cycle phases with their duration. The respective time of modeling is considered using a time stamp. He also introduces the concept of projects to summarize change activities [Saa10]. But the author provides no execution or implementation details.

2.1.3.2 Impact Analysis

The category of impact analysis summarizes the approaches that predict the behavior or effects of changes applied within the architecture model. If a change is made to the model, this typically affects other elements. Even a small change in one element can cause a ripple-effect, thus affecting other parts which are not obvious [Boh02]. Thereby, [Boh02] differentiates direct and indirect impacts: A direct impact (or first level impact) occurs, if the affected objects is directly connected. This can be determined via the connectivity graph using a dependency matrix. An indirect impact (n-level impact) is characterized by a set of relations, representing an acyclic graph, to the affected element. This impact can be determined using a reachability graph. Such a reachability graph indicates potential impacts and typically over-estimates the impact of a change (problem of false-positives). Using a constraint mechanism this effect can be reduced. Structural or semantic information can be used for it [Boh02].

Briand et al. [BLO03] propose a methodology and tool to perform impact analysis in design documents to detect side effects of changes in Unified Modeling Language (UML) models. Since the result set of UML model elements of such an impact analysis is often very large, a way to prioritize the set is necessary. Therefore, the authors propose the use of a coupling measure and predictive statistical model. The impact analysis itself is performed using OCL rules.

Aryani et al. [APH10] propose an approach for change propagation analysis on the software domain level. Their prediction is based on a conceptual coupling measurement for software components. This measurement is dependent from the function of a user interface components as well as the data this function uses. Using this information, a dependency matrix between the user interface components and the respective data is established.

[dBBG+05], [KRRR08] and [vKG03] propose approaches for change impact analysis based

on the different types of relationships and rules for change propagation. In [dBBG+05] a change is removing, extending or modifying an existing architectural element. The impact on another element in the case of a change is dependent on the semantics of the relations between them. The authors define heuristic rules to determine the impact of change for the most important relationships in ArchiMate models, i.e. access, assign, use, realize and trigger. The effect of the change should then be propagated along the graph edges.

Kumar et al. [KRRR08] define general relationships between the elements of an EA to be able to propagate information along them. The element types they consider are business goals, process transactions, service components and infrastructure components and the relationships are *runs on*, *provides*, *executes* and *delivers*. Using rules, they define how attributes of entities are dependent of each other, e.g. a running service reduces the availability of the infrastructure component. If changes occur the attributes are recalculated. For executing the propagation no implementation is described.

In [vKG03] the authors differentiate between three types of relationships to define the traces for later impact analysis: representation, refinement and dependency relationships. To determine the change impact, they utilize a (semi-)automatic analysis of the trace according to three different impact categories. The primary, secondary and tertiary impact category define the impact dependent on the element type and relationship type.

[HNF+09] and [TNJH07] propose Bayesian Belief Networks (BBN) to model causal dependencies and calculate the failure impact respectively change impact within the model. The failure impact analysis [HNF+09] calculates a ranking of architectural components with respect to the criticality for a business process. This approach focuses on the availability of the components and not changes. [TNJH07] propose predictive reasoning, diagnostic reasoning and a combination of both to perform a change impact analysis between architecture elements and decisions. Based on a probability at each root node and a conditional probability table at each non-root node the BBN algorithms calculate the posterior probabilities of the non-root nodes.

2.1.3.3 Gap analysis

Gap analysis is used to compare two different states of an architecture with each other [The18, Han13]. Thereby, elements that are removed, forgotten or new can be identified. Within TOGAF the gap analysis is performed based on matrices, where on one axis the current elements and on the other one the target ones are shown. The cells are then filled out manually, if the elements are available in both or not [The18].

In [DB13] semantic web technologies are proposed to perform the gap analysis. With this technique the set of elements only in the current architecture and those only in the target architecture are determined. The successor relationships between information systems are identified using the business support information.

Within their method for planning enterprise transformation [AG10a] propose a transformation model to store the information about the relationships, predecessor and successor, between elements of the current and the target architecture. The elements that are in the focus of the transformation project are determined using graph comparison. As a result of this comparisons six different relationships types are identified between current and target. This could be a 1:1 correspondence but it is also possible that an element has no or more than one successor and vice versa for predecessors.

2.1.3.4 Architecture evaluation

The category of architecture evaluation summarizes approaches to assess quality attributes of the architecture or the architecture elements. Typical examples are the analyses proposed in [Nie06]: Coverage analysis, analysis of the interfaces, heterogeneity analysis, as well as analysis of complexity, conformity, costs and benefits.

Iacob and Jonkers provide in [JI09] a detailed description for a performance analysis on EA models created with ArchiMate. Based on the average number of uses, the service times, the capacity and the arrival frequency several measures are calculated. The calculation relies first on a top-down workload calculation, followed by a bottom-up calculation of the performance. Performance measures are the processing and response time of services and the utilization of resources. If information about costs are provided, a cost calculation can be performed too. Beforehand the given model has to be transformed in a normalized model in order to ensure the applicability of the algorithms.

In order to quantify and evaluate the current or target architecture, KPIs can be used as measurement indicator [HRSM13]. Matthes et al. [MMSS11] defined 52 KPIs to measure EA management goals, based on a literature study. For each KPI a description, the necessary part of the information model, the addressed organizational goals and a calculation rule is given. For some KPIs also best practices for the measurement frequency, target or planned values and the KPI owner and consumer are given. An example is the *Application continuity plan availability* KPI. This KPI can be used to measure the goals *Improve capability provision* and *Increase disaster tolerance*. The KPI is defined as the number of critical applications where tested IT continuity plan is available divided by the total number of critical applications.

At the Royal Institute of Technology (KTH) several approaches for EA analysis are developed that can be categorized into architecture evaluation. They define EA analysis as the "application of property assessment criteria on enterprise architecture models" [JLNS07a]. Within their approaches they provided several theories for quality attributes of EA models like information security, system quality or maintainability. While utilizing different probabilistic techniques they enable the assessment of these quality attributes.

In [JJ05, JNL07] they propose Architecture Theory Diagrams to decompose system properties like enterprise information security or modifiability into sub-properties with definitional relations. These more concrete sub-properties are used to quantify the abstract quality attribute through a weighted aggregation of the single scores. Therewith they provide a quantified method for the assessment of these properties.

In [JJSU07, LJ08, NSJ+08] an approach for EA analysis with Bayesian networks is proposed. Bayesian networks enable the definition of causal relations between nodes and uncertainties. Using conditional probability distributions, the dependencies between nodes can be specified. Figure 2.10 shows how the property *availability of application* can be defined with a Bayesian network. The network denotes that it is dependent from *availability of server* and *availability of operating system*. In order to adapt this technique to EA, the EA meta model has to be extended with the relevant attributes and attribute relationships (abstract model). The dependencies between the attributes are expressed with conditional probability table (CPT) (see figure 2.11). It is also possible to consider the correctness of the relationships between elements, the value of attributes and the existence of the element. During the analysis the unassigned values in the concrete model are calculated according to the attribute relationships.

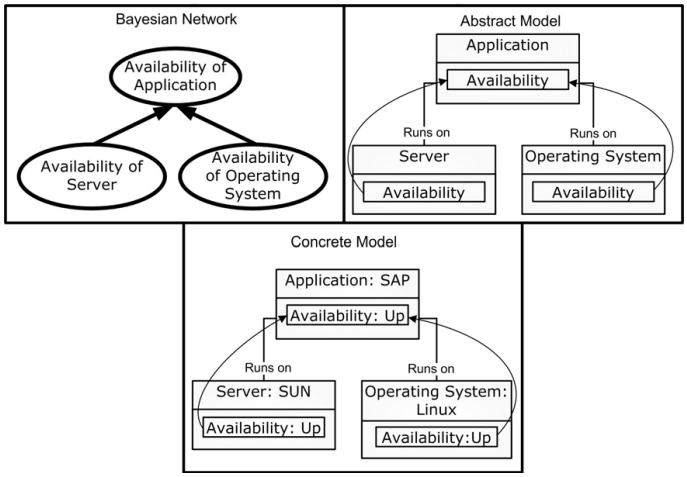


Figure 2.10: Approach for EA analysis with Bayesian networks [NSJ+08].

Availability of O/S		Up		Down	
Availability of HW		Up	Down	Up	Down
Availability of App.	Up	1	0	0	0
	Down	0	1	1	1

Figure 2.11: Conditional probability matric [NSJ+08].

In [LJ08, EFJ+09] the authors use this approach to develop a model for maintainability analysis in order to prioritize and decide for change projects. In [NSJ+08] the Bayesian analysis approach is used for estimating availability, accuracy, confidentially and integrity of two SOA platforms.

In [JLNS07a] the authors propose Extended Influence Diagrams (EID) to calculate the utility of different scenarios and support rational decision-making. Since influence diagrams extend Bayesian networks with decision and utility nodes, it is also possible to represent goals and decision alternatives. Decision nodes are used to represent the choice between different scenarios, the utility node represents the target of the EA analysis. While using the probabilistic inference mechanism of Bayesian networks, they enable the assessment of quality attributes of the EA model and support decision-making between different scenarios. In [NJN07, JLNS07b] this technique is used to assess information system quality based on the ISO 9126. According to this, technical quality is defined through the functionality, the reliability, the usability, the efficiency and the maintainability. Each of these properties is further refined using extended influence diagrams in order to enable a quantification. For example, one factor for maintainability is the system architecture quality. The definition of this property is shown in figure 2.12. Each variable in this diagram like size and age of the system has to be modeled as an attribute in the EA model. The execution of the analysis is performed with the Bayesian network analysis tool GeNIe.

[BUF+11] propose a tool for EA analysis with Probabilistic Relational Models (PRM), a technique related to Bayesian networks. The attributes within the model can be described with a probability distribution using the PRM formalism. This distribution is then used to infer the unknown values. The proposed tool supports the theory modeling for the quality attribute to be analyzed and the application of this theory on a concrete EA model. The EA analysis process is similar to the one proposed in [JJSU07]: First the PRM is defined,

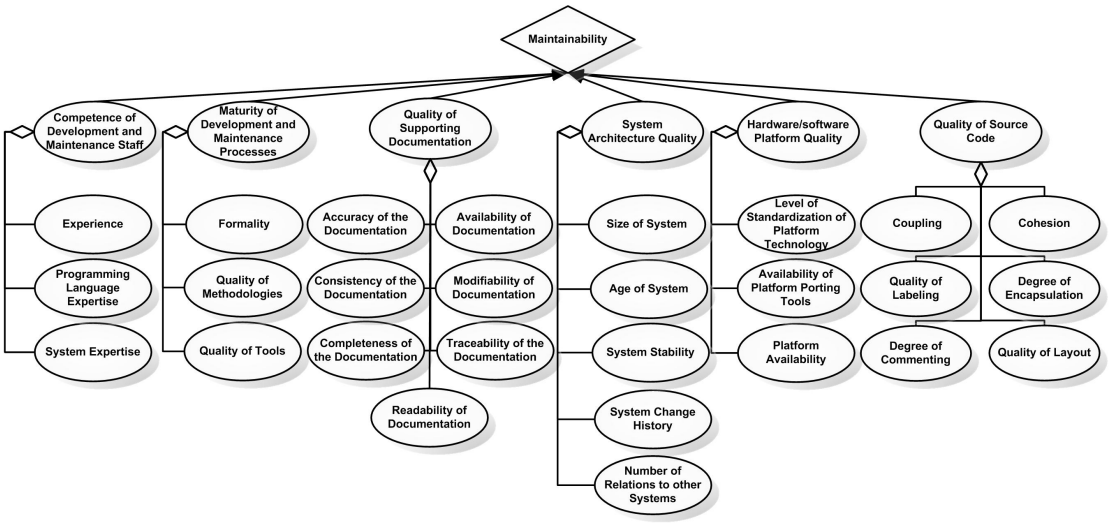


Figure 2.12: Extended influence diagram representing the theory for maintainability analysis [NJN07].

then evidence for the elements is collected and the PRM is instantiated and finally the analysis is executed to calculate the values of the quality attributes. PRMs are also used in [BFH+09] and [LJE10] to formalize the model and calculate the conditional probability of model element for a specific concern. [BFH+09] use the approach for quantitative analysis in combination with the information patterns from [BEL+07]. Therefore, they extend the information models with quality attributes and dependencies between these attributes. In [LJE10] PRMs are applied for enterprise systems modifi-ability analysis, i.e. assessing the cost of making changes to enterprise-wide systems. A major weakness of this approach is the missing query possibility for structural information, thus reasoning within PRMs is limited to object attributes.

To overcome this weakness [UFBJ10, NBE14] propose probabilistic OCL (p-OCL) for EA analysis. p-OCL is an extension of OCL to deal with uncertainty of objects, relations and attributes in the EA models and consider this information within the analyses. p-OCL supports stochastic attributes and the definition of uncertainty regarding the existence of objects and relationships. The probabilistic aspects of p-OCL are implemented in a Monte Carlo fashion: For every iteration the stochastic variables as well as the existence attributes are randomly instantiated according to their distribution. Each p-OCL statement is transformed into an OCL statement. The OCL statements are then evaluated and the result is a weighted mean of all iterations. [UFBJ10] apply p-OCL for interoperability analysis. Thereby, they model the theory behind interoperability with PRM, extended with p-OCL statements. In [NBE14] four formerly developed meta models for analysis approaches are integrated in order enable re-use and trade-off analysis. The meta models for assessing application usage, service availability, service response time and accuracy were re-implemented with p-OCL.

2.1.4 EA planning and decisions making

As discussed in [LB18a] the discipline of Enterprise Architecture Planning (EAP) deals in specific with the development and implementation of the target architectures [AGSW09].

Transformation planning from the current to the target architecture has to be done in alignment with the strategy of the organization [NFT+17]. While implementing the desired target architecture, organizations are faced with new demands [The18]. Following [ASML12] a project can be initiated during strategic planning in order to realize the desired target architecture or be a demand driven project. It is important that those demand-driven projects are realized with respect to the EA strategy. Demands can be driven by new technologies and trends, the need for cost reduction or the integration of standards. A current example is microservice architectures. Sources for business-driven demands are business developments, new innovations and strategy changes. Additionally, ensuring the compliance to legal regulations like Basel III for the financial sector or Merger & Acquisitions lead to changes in the architecture. Such changes do not necessarily conform to the architectural strategy, i.e. the specified principles and goals. The resulting phenomena of a moving target is known challenge within EA planning [Nie06, AGSW09]. It is important to integrate the EA strategy within the projects, i.e. plan and execute them in an EA compliant way [Nie06, FB08, ASML12]. Only if the projects follow the defined strategy, the benefits from the EA initiative and the desired target architecture will be reached.

The frameworks and processes for EAM are aware of this topic. According to [ASML12] necessary change projects have to be evaluated in order to decide about their implementation. It is important that these projects are realized with respect to the EA standards and principles as well as the desired target architecture. To ensure EA compliant projects [ASML12] proposes approval gates during the project to verify the chosen decisions. EA reviews verify the created specifications, design documents or prototype whether they adhere to the EA standards and principle and are aligned with the current and target EA.

Within the TOGAF ADM the first phases deal with the definition of the current and target architecture as well as a respective road map. Used concepts thereby are a gap analysis and the resolving of impacts. In the following phases the realization of the specified target architecture is in the focus (phases E, F, G). Finally, phase H addresses Architecture Change Management. This phase includes an analysis of the performance and gaps as well as to ensure that change requests conform with the EA governance and framework.

Additionally, there exist more detailed and specific approaches that address EAP. Figure 2.13 summarizes the main steps of them.

Spewak et al. [17,18]	Niemann [6]	Pulkkinen et al. [10,19]	Aier et al. [4]	Aier, Saat [20]
A1. Planning initiation A2. Define values and principles A3. Identify business knowledge and current systems and technology A4. Blueprint data, applications and technology architecture A5. Develop implementation and migration plan A6. Define programmatic transition	B1. Define goals B2. Documentation B3. Analysis B4. Planning alternative scenarios B5. Evaluation of alternative scenarios B6. Implementation	C1. Initiation a. Define goals b. Resource and constraints C2. Planning and development: define needed changes in architectural dimensions C3. Ending phase a. Plan, design and evaluate alternative architectures and solutions b. Define long term and short term targets	E1. Define Vision E2. Model As-is Architecture E3. Model (multi-step) alternative to be architectures E4. Analyze and Evaluate Alternative (subsequent) to-be architecture E5. Plan Transformation from As-is to to-be E6. Implement transformation	D1. Strategic EA planning a. Derive goals b. Define long term vision c. Prioritize courses of action D2. Operational EA planning a. Define requirements b. Model variants c. Evaluate and select domain variants d. Consolidate to be models D3. Implementation a. Define change projects b. Launch projects c. Monitor projects

Figure 2.13: Selected planning processes in literature (extended table from [AGSW09]).

One of the first approaches for EAP is the 'Wedding Cake Model' from Spewak and Hill [SH93, ST06]. The current architecture is analyzed according to the values and principles, for example with the use of metrics. Based on these results, the blueprint data, application

and technology architecture are developed. The results of a gap analysis between the current and the target architecture are used to develop an implementation and migration plan.

Within the planning step of the EA development process in [Nie06], the road map from the current architecture to the desired target is defined. Development planning is required in order to integrate new projects with the optimization of the existing application landscape. The proposed process for this step according to [Nie06] is shown in figure 2.13. The first step is the development of different scenarios for the target application environment. Thereby, so called "what-if" analyses support the architect. Each of the scenarios is evaluated with respect to the enterprise and IT goals, the costs and the risk. Also, the gaps between the current and the developed scenario are analyzed. Based on the analysis results the scenario is developed into a target architecture and the required steps to implement them are defined.

In [PH05,Pul06] the authors propose a process model for the management of architectural decisions in EA planning. Thereby, they differentiate between four architectural layers (i.e. business, information, application and infrastructure) and three decision-making levels (enterprise, domain and system level). The authors propose a spiral model for decision-making, where decisions are refined top-down, considering the EA layers and the decision-making levels. A concrete EA project is performed in three steps: In the *Working Phase* the EA planning and development takes place i.e. the required changes in the different architectural dimensions are defined. In the *Ending Phase* alternative architectures and solutions are designed and evaluated. Finally, a development road map is created as well as long- and short-term targets.

The EAP process from Aier et al. [AGSW09] starts with definition of the vision and the creation of the as is-architectural model. In the following, (multi-step) target alternatives are modeled, analyzed and evaluated. Finally, the transformation from the current to the target is planned and implemented.

In [AS11] the authors performed a comparison of EA planning approaches in practice and literature and concluded with a set of common activities during EA planning. The proposed process differentiates the three main steps *Strategic EA Planning*, *Operational EA Planning* and *Implementation*. Similar to the results of [PH05, Pul06] the authors identified three different levels for the activities: The Enterprise Level (mostly strategic planning), the Domain Level (mostly operational planning) and the Project Level (mostly implementation). The definition of requirements for the identified actions as well as the modeling of different target variants in the operational step is performed on domain level. Since the changes do not affect the entire EA the concrete planning can be performed using domain architectures. The established target models are finally integrated in the enterprise model to get a consistent blueprint [AS11]. Evaluation takes place on domain level as well as on the consolidated enterprise level.

Additionally, to EAM methods and EAP processes [NFT+17] and [AG10b] identified several requirements for EAP processes that are also considered while identifying the method blocks. Nowakowski et al. [NFT+17] provide a review of current literature about EAP as well as results of practitioner interviews. Based on them, they extract several requirements for EA planning. They include the ability to analyze and compare the current as well as the target architecture (including qualitative and quantitative metrics), the support for different planning scenarios and transformation paths as well as the availability of up-to-date data. In order to set-up a transformation path, it is necessary to create a

transformation model and to derive transformation projects from the target architecture. Gap analysis is proposed to compare the current and target model. They also point out the relevance of individual life-cycles, the ability to react to unplanned changes as well as the importance of tool support. In comparison with the practitioner interviews [NFT+17] found out that EA planning is primarily carried out by visual comparison of specific models, while the methods proposed by literature are not considered in practice. They propose the development of simpler and more practical approaches especially for the comparison of scenarios.

Aier et al. [AG10b] also work out several requirements for EAP. Among those are the development and evaluation alternatives and the consideration of successor relationships and life cycles in the target models. It must be possible to derive project activities from the developed target model. Overall mean of EAP is the provisioning of information to support change projects. Thereby, the different requirements from the stakeholders have to be considered.

Finally, Foorthuis et al. [FB08] differentiate in their work about EA project compliance between three different layers: Enterprise Architecture (EA), Domain Architecture (DA) and Project Architecture (PA). EAs describe the current and target architecture at the highest level, whereas DAs focus on one specific group of products, services or functions. The focus of PA is the relevant artifacts for one project and describes specific solutions. In [FHBB12] the authors propose an EA compliance model including a compliance testing process. In the center are so called prescriptions which can be principles, models or policy statements. Prescriptions provide norms that have to be applied. Within four different compliance checks the prescriptions are verified. The checks include a correctness check, a justification check, a consistency check and a completeness check. Thereby, the authors propose to start with compliance testing in the early phases and thus to avoid validation of the prescriptions at project level. The full automation of the testing processes is not realistic according to the authors but they propose tool support where it's feasible e.g. operationalizing norms and compliance checks [FB08, FHBB12].

2.2 Data-flow Analysis

As discussed in [LSB14a], Data-flow Analysis (DFA) is a technique based on the principle of information propagation. This principle enables the declarative and recursive specification of analyses. DFA is suitable for static model validation, information extraction, validating modeling guidelines, calculations metrics and supporting model transformations as well as refinements. In [Saa14] several templates for generic analyses are proposed:

- Analysis of reachability and liveness
- Analysis of successors and predecessors
- Determination of strongly connected components
- Flow path analysis
- Analysis of the availability of resources
- Analysis of type information regarding a generalization hierarchy
- Test for cycles
- Context sensitive analysis, e.g. determine predecessors regarding a containment relationship

The technique originates from the area of compiler construction [Kil73]. When translating the source code of a program into machine code, DFA is used to statically derive optimizations based on the structural composition of the program instructions. By examining each basic block in its overall context, it is possible to derive information that holds true for each possible execution of a program. Examples are the calculation of reaching definitions and variable liveness.

For this purpose, a program is usually represented as a control-flow graph. Nodes represent the basic blocks and the edges represent the flow of control. Subsequently, a set of data-flow equations is computed at each node. Within the equations, the results at the immediate predecessor and successor nodes are combined with a confluence operator (set union or intersection). Additionally, the values are modified in order to capture the effects of the instructions made within the basic block of the node. Thus, the provided output relies on the basic block instructions and, recursively, on the results of all predecessor nodes. Consequently, the results are propagated along the edges of the control-flow graph allowing for a context-sensitive evaluation of each equation. Since the presence of loops leads to an infinite number of execution paths (and consequently cyclic DFA equation systems), fixed-point evaluation semantics are employed to approximate the runtime behavior of programs.

In [SB13, Saa14] this analysis technique is adapted to the modeling domain, resulting in a "generic 'programming language' for context-sensitive model analysis". The approach defines a declarative specification language that allows the annotation of data-flow attributes at meta model classes. These attributes can subsequently be instantiated and evaluated for arbitrary models to enable a static approximation of their dynamic behavior. The dependencies between the attributes are implicitly defined within the flow equations. The respective dependency graph for the attributes is constructed on-the-fly during the evaluation. The utilized fix-point computation supports the evaluation of transitive and cyclic declarations.

The attribution consists of a declarative part and the implementation of the actual rules. Therein, the DFA equations (also called propagation rules) are assigned to meta model el-

ements. This can be either done in a textual file or using Java Annotations for propagation rules implemented within Java code. It is possible to extend existing modeling languages with attributes and the respective propagation rules. This enables the application of data-flow analysis without a modification of the modeling language. The propagation rules can be specified with arbitrary languages and are evaluated using a DFA solver. Currently supported are a textual definition in combination with Java or their specification with OCL. The demand-driven and iterative algorithm of the solver supports the dynamic discovery of the attribute dependencies as well as utilizes fix point computation. [SB13, Saa14]

A reference implementation of this approach exists in the form of the Model Analysis Framework (MAF) [SB11] which has been successfully employed to specify and carry out analyses in different domains [Saa14]. The framework utilizes the Eclipse Modeling Framework (EMF) [Ecl18a] for model representation. Within the business process modeling domain, it could be shown that the approach performs well for large models. The analysis results are provided immediately and enable the application of the framework in everyday usage scenarios. Additionally, the approach was applied in the context of model-based development of embedded systems, specifically AUTOSAR (short for AUTomotive Open System ARchitecture) models [KMKB14]. Here, it was used to validate and provide a semi-automatic refinement of automotive software design. The analysis performs well for medium sized models. Within a further project, it was shown that this analysis technique is also suitable for very large system models [Saa14].

Aside from scalability, this technique provides several advantages: Since information can be propagated along model edges, each model element can be evaluated in its overall context. Thus, eliminating the need for static navigational expressions as common in languages such as OCL. The annotation mechanism for the specification of the DFA propagation rules enables the generic applicability without changes to the original modeling language. These are important benefits in the EAM domain, where both the structure of meta models and models is highly dynamic. Secondly, data-flow analysis provides inherent support for the implementation of recursive specifications which iteratively propagate information through a model. Finally, the usage of fixed-point semantics allows the implementation of a correct handling of cyclic paths.

2.2.1 Application: Reachability analysis

As presented in [LSB14a] we illustrate the application of data-flow analysis with the evaluation of a reachability attribute. The assessment of an enterprise architecture is highly dependent on the computation of reachable elements. According to [Boh02], *dependability* refers to directly connected elements while *reachability* additionally regards transitive connections. Consequently, we can directly implement the following definition: *An element is reachable if at least one predecessor element is reachable.* In this context, a predecessor is defined as the source element of an incoming edge.

A simple meta model for control-flow graphs that specifies the class *node* along with two specializations *startnode* and *endnode* as well as a class *edge* is presented in figure 2.14. A Data-flow Analysis can easily determine whether a node is reachable from the *startnode*. For this purpose, we assign a data-flow attribute *is_reachable* of type Boolean to the *node* class (and thereby implicitly to its sub-classes). The annotation of the propagation rules to the nodes is illustrated in figure 2.14 through the notes. The instantiation of this analysis for a specific model attaches an instance of the *is_reachable* attribute to each *node*. The

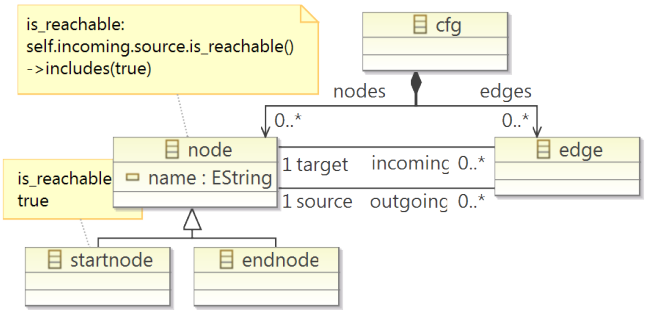


Figure 2.14: Control-flow graph metamodel with annotated analysis [SB13].

DFA solver is then responsible for determining the dependencies between the attribute instances and executing the data-flow rules in a valid order. The full attribution is shown in the following listing:

```
1  attribution reachability
2    attribute assignment is_reachable : Boolean initWith false;
3    extend node with
4      occurenceOf is_reachable calculateWith
5        "return self.incoming.source.is_reachable() -> includes(true)";
6    extend startnode with
7      occurenceOf is_reachable calculateWith true;
```

Listing 2.1: DFA propagation rules for reachability analysis [LSB14c].

As described above, an element e_2 is reachable from another element e_1, if there exists a path between e_1 and e_2. The reachability status is computed by a data-flow attribute *is_reachable* of type *Boolean* which is initialized with the value *false* (line 2). Lines 3 - 5 attach this attribute to all instances of the *node* class. To determine the reachability status of a node, the data-flow rule in line 5 accesses the *is_reachable* values computed at the respective node's predecessors, thereby directly implementing the recursive specification. Finally, for the *startnodes*, the rule is overwritten (lines 6 - 7). Startnodes are by definition, always reachable.

During instantiation, an attribute instance is attached to each node (see figure 2.15). The attribute *is_reachable* is assigned to all nodes, but dependent on the node type (*startnode* or *node*), different occurrences are used. The occurrences represent the data-flow equations for the respective subtype.

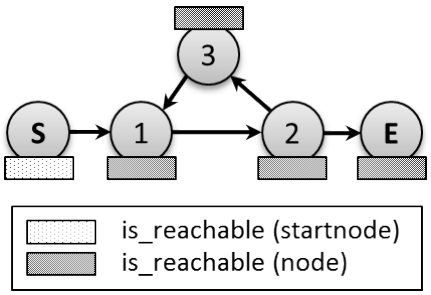


Figure 2.15: Control-flow model with attribute instances [Saa14].

2.2.2 Application: Flow path analysis

Another important application of DFA in the EA context is the flow path analysis. Instead of determining reachability, this analysis computes a list of nodes for each start node. Each list depicts one path within the control-flow graph from the `startnode` to a target node. The propagation rules presented in listing 2.2 extend the meta model in figure 2.14 with a `flowPaths` attribute. This attribute provides a set of paths, represented as lists of elements. For the `startnode` the set contains one path, consisting only of the elements itself (line 4).

```
1  attribution flowpaths
2    attribute assignemnt flowPaths : Set(List(Elements)) initWith {}
3    extend startnode with
4      occurenceOf flowPaths calculateWith "return Set{List{self}}";
5    extend node with
6      occurenceOf flowPaths calculateWith
7        "var result : Set(List(Element)) := Set{}
8         self.incoming -> forEach(edge) {
9          edge.source.flowPaths() -> forEach(path){
10           if(not path -> contains(self)) then
11             result: = result -> add(List{path} -> append(edge) -> append(self));
12           endif;
13         }
14        }
15        return result;"
```

Listing 2.2: DFA propagation rules for flow path analysis [Saa14].

For all other nodes the attribute `flowPaths` is computed according to the rule in lines 7 - 15. First the result value is initialized with an empty set. Then for all incoming edges the current status of the source elements is processed (lines 8 - 9). For each path within the retrieved status which does not contain the current element (line 10), the path is extended with the incoming edge and source element and added to the result set (line 11). After the DFA solver finishes the evaluation of this attribute, for each node the set of paths from the `startnode` is provided.

2.3 Semantic Web Technologies

The term semantic web describes the vision of a web of linked data [W3C15]. Thereby, technologies like the Resource Descripion Framework (RDF) [SR14], the SPARQL Protocol And RDF Query Language (SPARQL) [W3C13] and the Web Ontology Language (OWL) [HKP+12] are used to provide vocabularies and data stores as well as to infer from the data. These standards, developed by the World Wide Web Consortium (W3C), enable the creation of information systems comprising semantic interoperability [SBLH06]. Thereby, ontologies are used for the representation of knowledge. Ontologies represent the knowledge about relationships between different kinds of objects [Stu11].

A vocabulary or ontology is used to define the concepts and relationships required for the description of a specific domain. Thereby, the term ontology often refers to more complex and formal descriptions. The classifications, constraints and rules that can be provided within an ontology, support the integration of heterogeneous data and are the foundation for inference techniques. Inferencing describes the reasoning over data according to the specified rules. Therewith, new relationships can be derived and inconsistencies in the available data are detected. [W3C15]

Despite the original idea of semantic web as a "Web of actionable information — information derived from data through a semantic theory for interpreting the symbols", [SBLH06] the need for sharing semantics about data has emerged in further application scenarios. Well-known application scenarios for semantic web technologies are clinical research, life sciences as well as data integration in enterprises [W3C15, Stu11, SBLH06].

A major benefit of semantic web technologies is the integration of heterogeneous data. Inferencing enables the derivation of indirect dependencies and ensures the consistency of the model. Thus, the quality of the integrated data can be improved. With the expressive query language SPARQL, the stored data can be accessed in an efficient way.

Resource Description Framework RDF

The Resource Descripion Framework (RDF) [SR14, CWL14] provides the basis for the representation and linking of data. The principles of RDF are a triple-based representation of data as well as the usage of Universal Resource Identifiers (URIs) to identify resources [SBLH06]. A URI is used to identify the resources, i.e. anything that needs to be referenced. RDF provides a language in order to represent information about these resources in a machine interpretable way. It enables the specification of simple statements declaring properties and property values of the resources [SR14].

The triples of this graph-based data model consist of a subject, a predicate and an object. Objects of a statement can be resources but also literals. Literals are used to represent string values, numbers or dates. For example, the following statement describes that the resource *BookingCar* has a *criticality* of *7*:

```
<http://example#BookingCar>    <http://eam/criticality>    7.
```

An *RDF graph* is defined as a set of those triples. It can be visualized as a graph. Thereby, nodes denote the subject and object of a triple, arcs denote the predicate [CWL14]. Figure 2.16 presents an RDF graph describing the two individuals *Reservation by phone*, a business service, and *Booking a car*, a business process. The type of the two individuals

can be defined with the `rdf:type` predicate from the RDF built-in vocabulary. Additionally, the graph specifies that the process `realizes` the specified business service and has a `criticality` ranking of 7.

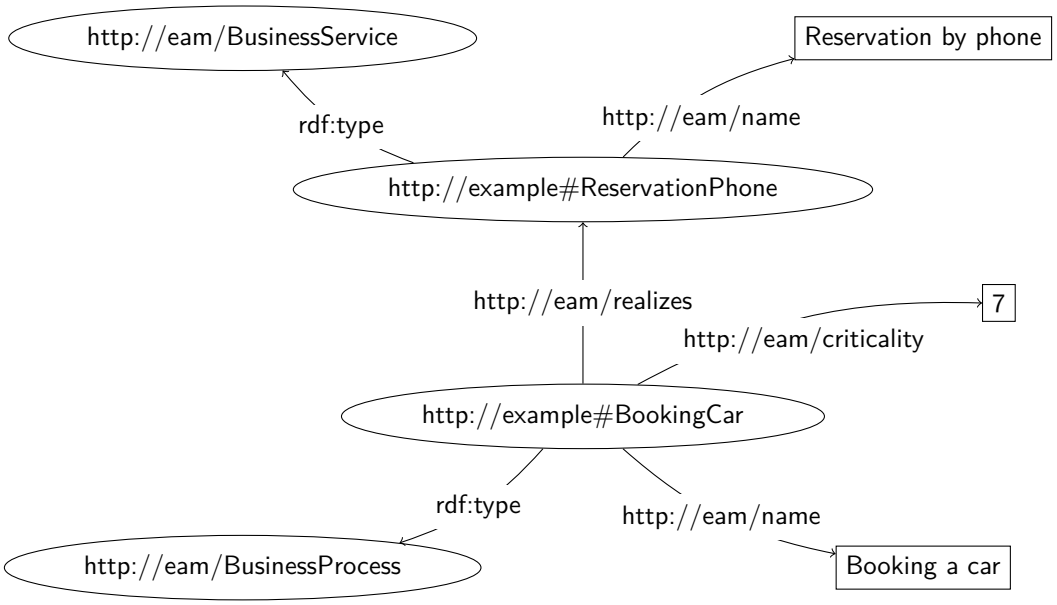


Figure 2.16: RDF graph describing a realization relationship between two elements.

Several RDF graphs can be collected within an *RDF dataset*. Each graph is called a named graph. A named graph is a pair of an International Resource Identifier (IRI) or blank node representing the graph name and an RDF graph. Graph names have to be unique within an RDF dataset [CWL14].

Triple Store

RDF data is stored within triple stores [SBLH06]. These repositories are used to persist and query the triples of an RDF graph. Jena TDB [Fou18] is a triple store for RDF data with the focus on reasoning and querying over the data [SBLH06]. Further triple stores focus on storing large data sets (e.g. 3store [HGR13] or RDF Knowledge Graph from Oracle [Ora19]).

RDF Schema

Resource Descripion Framework Schema (RDFS) [BG14] enables the specification of a vocabulary for RDF data. It can be understood as a semantic extension of RDF which provides a minimal ontology representation language. RDFS is widely adopted within the research community [SBLH06]. With RDFS groups of related resources and the relationships between them can be described. The utilized concepts of classes and properties are similar to the object-oriented data structures. In contrast to object-oriented schemas, where a class is defined according to the properties an instance has, RDFS "describes properties in terms of the classes of resource to which they apply" [BG14]. This is realized with the concepts of domain and range, defining the source and target elements of a property. An advantage of this principle is the expandability of the description.

The following listing presents the used vocabulary within the RDF graph in figure 2.16 using turtle syntax. After the prefix statements in lines 1-4, the two utilized classes are defined. In lines 5-6 the class `BusinessService` and in the following lines 7-8 the `BusinessProcess` class. According to the property-centric approach of RDFS no further properties are declared here. The respective properties `realizes` and `criticality` are defined in the following (lines 9-12 and 13-16). The domain and range value draw the dependency to the respective classes. More specifically the range of the criticality property is the datatype `xsd:integer`, denoting that a number value is expected as object.

```
1   @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3   @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4   @prefix eam: <http://eam/> .

5   eam:BusinessService
6     rdf:type rdfs:Class .

7   eam:BusinessProcess
8     rdf:type rdfs:Class .

9   eam:realizes
10    rdf:type rdf:Property .
11    rdfs:domain eam:BusinessProcess .
12    rdfs:range eam:BusinessService .

13  eam:criticality
14    rdf:type rdf:Property .
15    rdfs:domain eam:BusinessProcess .
16    rdfs:range xsd:integer .
```

Listing 2.3: RDF Schema describing the EAM vocabulary using turtle syntax.

SPARQL

To query and manipulate the stored RDF data, W3C proposed the standardized language SPARQL [W3C13]. SPARQL is a recursive acronym for SPARQL Protocol And RDF Query Language. Listing 2.4 shows an example SPARQL query in order to retrieve all elements, i.e. their URIs, with a criticality more than 3.

```
1 PREFIX  eam: <http://eam/>

2 SELECT ?element
3 WHERE {
4   ?element eam:criticality ?criticalityValue .
5   FILTER (?criticalityValue > 3)
6 }
```

Listing 2.4: Example SPARQL query.

The `WHERE` part of the query is built using so-called basic graph patterns. These patterns are similar to RDF triples, but allow the usage of variables. A query can be composed of several basic graph patterns. If there is a match of the specified graph patterns within the RDF graph, the data is added to results. For `SELECT` queries, the data is bound to the defined variables. `ASK` queries evaluate to true or false, depending on the fulfillment of the specified patterns. A `CONSTRUCT` query returns a new RDF graph which is built according to the patterns specified in the `WHERE` clause. Despite the simple graph patterns, SPARQL queries support group graph patterns and optional patterns. Additionally, operators like filter, union and minus can be used to restrict the results of the patterns. Also, aggregation mechanisms, property paths and nested queries are supported. Named graphs can directly

be addressed within SPARQL queries and allow the restriction of the evaluation of a graph pattern within a specific named graph.

Semantic web and EAM

Within [CHL+13] the use of semantic web technologies is proposed for EAM. This technology stack addresses the challenge of managing the heterogeneity of different data sources integrated within EAM. Thereby, the different data set are formalized in a first step and in the following are linked to each other. The integrated data model can then be used for further analysis and processing.

[OLB15, SKR13a, SKR13b] propose semantic web technologies for the representation of an EA model and present analyses to be performed on this model. The authors showed that this technology stack enables a fast implementation of several existing EA analyses. Implemented analyses include a change impact analysis, a landscape mapping analysis deriving indirect relationships and KPI calculation.

Additionally, there exist several tools leveraging semantic web technologies for EAM. This is *Essential* [Car16], *TopBraid* [Top] and the *Living EAM Platform* [Sof19a]. The three tools utilize the flexibility of semantic web technologies to provide tailored solutions as well as enable the integration of existing data sources.

Part II

Performing Architecture Analysis

3

Capture the Enterprise Architecture Model

In this chapter a modeling approach for enterprise architecture models is presented. Based on the requirements identified within current literature a generic representation schema for those models is developed. The model is extend with further concepts to enable the later execution of analyses. The application of the meta model is finally illustrated with a small model part.

The sections in this chapter present research results already published in [LSB14a] and [OLB15].

3.1 Requirements

As discussed in [LSB14a], to derive meaningful information from a model, an analysis must incorporate knowledge about the semantics of the language constructs. In general, the structure of any modeling language has a significant impact on the way analyses are implemented and executed. This can be a problem in areas such as enterprise architecture modeling, where a large number of competing standards and practices exist [BBJ+11, JNL07]. To circumvent this, we propose requirements for a generic meta model which acts as a universal representation format for EA models, and provides a common foundation for analysis specifications.

Analyses heavily rely on semantic information, since the evaluation of a model element usually depends on its meta model type. The generic meta model therefore not only has to encode the actual model data but also the related meta information, i.e. the classes and associations of the respective meta model. Between both artifacts proper `instanceOf` relationships have to be established. As a consequence, each instance of the generic meta model conforms to a representation of the EA model as well as the EA language itself. This approach has two benefits: On the one hand, the established EA meta models includes only those concepts that are actually used in the EA model. On the other hand, it enables the definition of any analysis over the EA, without adapting the EA meta model. All required adaptions can be covered through an adequate analysis specification.

Main focus of analyses are the dependencies between elements. They are for example used to extract views or aggregate costs. For proper analysis support the model and meta model information should be provided in a way that enables the navigation within the model. Additionally, it is necessary to access properties of the elements like costs, but also properties of the dependencies like the frequency of usage. The type of the properties, for example currency, date or number, has to be declared too.

Beside the requirements to enable analysis execution, the generic meta model must be able to represent the typical characteristics of EA models. An EA model can be seen as a set of things or elements that have specific relationships to each other [BBJ+11, Dep10] . Accordingly, [BBJ+11] proposes the primitives *classes*, *properties* and *binary associations* for an EA information model. Properties can be specified for classes and associations. Additionally, the authors argue for a datatype concept respectively enumerations for further information about the properties.

To ease the specification of generic analyses the meta model should provide additional structure for typical relationship and concept types within EA models. It is important that only the most common concepts are explicitly represented. A large and complex model structure, where only a small amount of the available concepts is used, should be avoided. The meta model should be designed as simple as possible, to enable a broad support of existing approaches. Examples for important and commonly used relationships are ones that indicate composite structures like `wholePart` [Dep10], hierarchies [BBJ+11] or `composition` [The17]. Relating to the overall align-enable principle of an EA, relationships that determine a `usage` or a `provisioning` of some functionality should also be determinable within the generic meta model.

An often-occurred characteristic within EA models is the multiple use of relationship types in different context's (e.g. [The17]). Thereby, a *used by* dependency is utilized to represent that a business process utilizes an IT service. The same relationship type *use* is also applied to represent a communication between an application system and infrastructure services

(see figure 3.2 for illustration). Although the *use* semantic is similar in both cases, they may be treated differently during analysis execution. Therefore, the generic meta model should be able to differentiate between those two relationships. The information that they belong to the same type should be kept.

The frameworks and meta models for EA typically use layers for further structuring. Layers can also be used to provide some kind of packaging mechanisms as demanded by [BBJ+11]. Finally, an important concept used within EA models is `typeInstance` [BBJ+11, Dep10]. It is used to define the relation between an application system and its concrete instances. Typically, large companies have several instances of an application. Each instance may serve a different market or organization unit. The meta model should provide respective concepts to be able to consider this type of relationships inherently.

Table 3.1: Requirements for a generic meta model for the representation of EA models.

	Analysis requirements		*EA information requirements*
R1	Represent actual model data	R7	Represent Elements
R2	Represent meta information	R8	Represent relations between elements
R3	Depict instance relations	R9	Properties for elements and relations
R4	Provide structure for navigation within the model	R10	Multiple use of edge types
R5	Provide access to properties of the elements	R11	Structuring mechanisms
R6	Type information for properties	R12	Multi-level modeling

Table 3.1 summarizes the requirements we described above. On the left-hand side, the necessary requirements to enable analysis execution are summarized. On the right-hand side, requirements addressing the depicted information within the meta model are specified. A generic meta model, fulfilling these requirements, would provide a solid foundation for analysis execution and enables the representation of EA models utilizing different meta models. To keep the meta model as lightweight as possible, we do not want to provide a full modeling language including model constraints. The only purpose of the generic meta model is to provide a representation of EA information for analysis execution.

3.2 Generic Meta Model

The provided Generic Meta Model (GMM) is a universal applicable meta model used for storing the EA information and used for the definition and execution of analysis specifications. For its specification we use the following formalization based on [KA09].

We formalize an EA meta model as tuple:

$$MM = (ET, RT, PT, L, R, M, D) \tag{3.1}$$

such that

ET	is a set *element types*
RT	is a set of *relation types*
PT	is a set of *property types*
L	is a set of *layers*
$R \subseteq ET \times RT \times ET$	is the set of relations that exist two *element types*.
$M : ET \to L$	is a function that returns the *layer* of an *element type*.
$D : PT \to ET \cup RT$	is a function that returns the *type* a *property type* belongs to.

An EA model is described as tuple:

$$A = (E, P, R', I, EA, RA, MM) \tag{3.2}$$

such that

E	is a set of architectural *elements*, being an instance of an *element type*
P	is a set of *properties* having a concrete value
$R' \subseteq E \times RT \times E$	is a set of relations that exist between two *elements*
$I : E \to ET$	is a function that returns the *element type* of an *element*
$EA : P \to PT \times E$	is a function that returns a pair of the *property type* and the *element* a *property* is assigned to
$RA : P \to PT \times R$	is a function that returns a pair of the *property type* and the *relation* a *property* is assigned to

For a model A to be a correct instance of a meta model MM the following conditions have to be fulfilled:

$$\forall e_1, e_2 \in E, rt \in RT : (e_1, rt, e_2) \in R' \implies (I(e_1), rt, I(e_2)) \in R \tag{3.3}$$

$$\forall p \in P, pt \in PT, e \in E \ : \ EA(p) = (pt, e) \implies D(pt) = I(e) \tag{3.4}$$

$$\forall p \in P, pt \in PT, e_1, e_2 \in E, rt \in RT \ : \ RA(p) = (pt, (e_1, rt, e_2)) \implies D(pt) = rt \tag{3.5}$$

Condition 3.3 ensures that for each relation $r' \in R'$ between two elements a corresponding meta relation $r \in R$ exists. The r' must conform to r in terms of the relation type $rt \in RT$ and the assigned element types $I_e(e_1), I_e(e_2) \in ET$ of the elements. To ensure the validity of the properties of an element, the conditions 3.4 and 3.5 are defined. They state that for each property $p \in P$ the specified property type $pt \in PT$ must have an assignment to the respective element type (condition 3.4) or relation type (condition 3.5).

The idea for the GMM is to represent the EA model and its meta information as a stereotyped graph. Thereby, both, the model and the meta model information, are represented in one model. Apart from abstracting from the particular structure of an input model, this approach has the benefit of combining meta model and model data in a single representation. This enables a dynamic creation of the meta model, representing the actual structure of the model.

A condensed version of the GMM specification is depicted in figure 3.1. The relevant elements can be described as follows: The root element `GmmModelContainer` contains a `GmmMetaModel`, a `GmmModel`, each of which acts as a container for the elements of the respective artifact type. Within the GMM additional types are defined to assign common properties like Universally Unique Identifier (UUID) and the name.

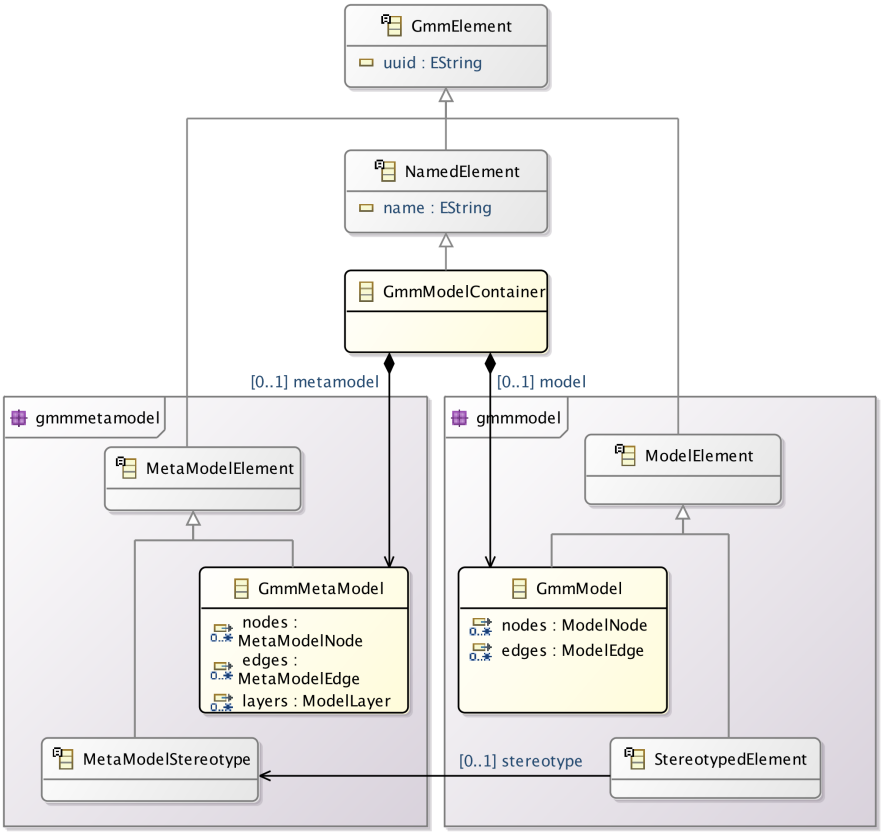


Figure 3.1: Overview of the Generic Meta Model GMM.

Within the `GmmMetaModel` the element types ET, layers L and relation types RT are defined. These are `MetaModelNode`, `MetaModelEdge` and `ModelLayers`. Each element type of the respective EA model is translated into a `MetaModelNode`, each relation type to a `MetaModelEdge`.

The `GmmModel` serves as container for the elements E and relations R. Element instances from the EA model are converted into `ModelNode` and relations between them into `ModelEdge`. The `ModelNodes` as well as the `ModelEdges` are connected to their respective meta model elements. For this purpose, each node, edge and also property type defined in the `EAMetaModel` specializes the abstract `MetaModelStereotype` class. On the other hand, each instance of a node, edge or property in the `EAModel` is a `StereotypedElement` which

references a stereotype from `EAMetaModel`.

The details of the `GmmMetaModel` and `GmmModel` including the structure for properties and stereotyping are described in the following two subsections. For illustration purposes we introduce the enterprise architecture of a fictitious car rental company, CarRental, as running example. The part of this model, relevant for booking a car, is shown in figure 3.2. The full EA model is provided in the appendix in figure A.1. The model corresponds to the provided demo EA model shipped with the modeling tool [MID19].

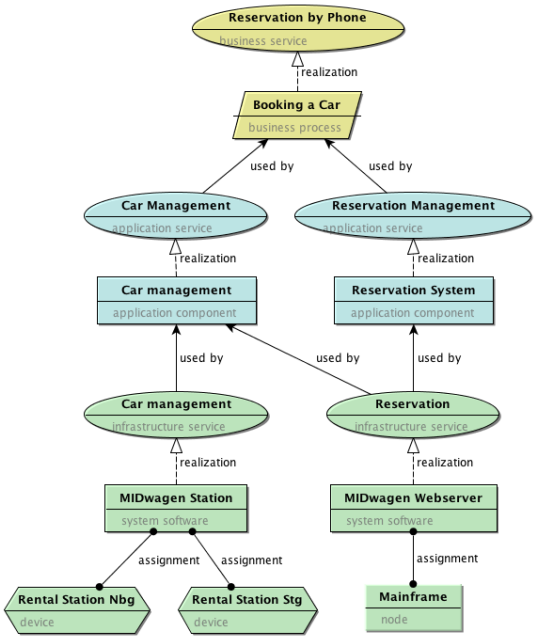


Figure 3.2: Part of the enterprise architecture of the CarRental company.

Within this architecture part, the IT support for the business service *Reservation by Phone* is depicted. The realizing business process utilizes two IT services provide the required information. One is implemented by the *Car Management* application, providing relevant data about the available cars. The other one is implemented by the *Reservation System* which manages the already transacted reservations. The application systems itself utilize additional infrastructure services provided by a defined system software, running on a specific device. The used elements are arranged and colorized according to the business layer, the application layer and the infrastructure layer (from top to bottom).

3.2.1 GMM meta model

The detailed structure of the GMM Meta Model is provided in figure 3.3. A `GmmMetaModel` consists of a set of `MetaModelNodes`, a set of `MetaModelEdges` and a set of `ModelLayers`. In order to map the structure of an existing EA meta model to the GMM format, the defined element types in ET have to be translated to `MetaModelNodes` while their associated properties $pt \in PT$ with $D_pt(pt) = et$ become `MetaModelProperties`. The `properties` relation captures the assignment of pt to et from the function D. For each `MetaModelProperty` also its type is stored. Examples for types are date, number or string. According to the function M each `MetaModelNode` has a connection the `ModelLayer` it

belongs to.

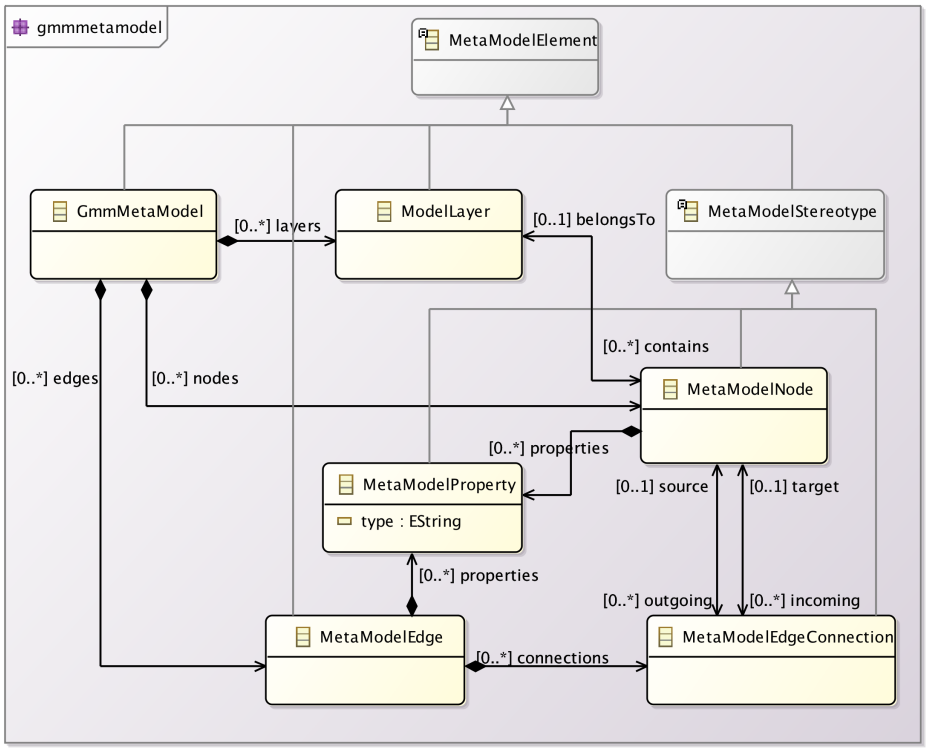


Figure 3.3: Details of the GMM meta model package.

Relation types are depicted with two concepts: `MetaModelEdge` and `MetaModelEdgeConnection`. This distinction is motivated by the fact that the specifications of various EA standards use the same relation type between different source and target types. The interpretation of them within an analysis may differ in some cases. Additionally, this may conflict with restrictions imposed by the current state of modeling technology. In the UML for example, it is not possible for a class to declare two incoming or outgoing associations with the same name. This usually requires to implement workarounds such as assigning a unique numeric index to each association. The downside to this approach is the loss of semantic information as associations of the same type but with different indices are treated as separate entities by modeling tools. The definition of a `MetaModelEdge` type that can be shared by multiple `MetaModelEdgeConnections` solves this problem. The `MetaModelEdgeConnection` defines the source and target meta model node of a relation type according to the set R. The `MetaModelEdge` aggregates different relation types with the same name, for example *used by* relations. Also, the property types $pt \in PT$ of a relation type are specified at the `MetaModelEdge` level.

To represent the instantiation of model elements with meta model elements, the `MetaModelNode`, the `MetaModelEdgeConnection` and the `MetaModelProperty` inherit from the supertype `MetaModelStereotype`.

A simplified textual representation of the meta model from the example in figure 3.2 is shown in the following. The syntax follows the representation of EMF models within the EMF editor. Properties of the elements as well as their UUID are omitted due to visibility reasons. `MetaModelLayers`, `MetaModelNodes` and `MetaModelEdges` are demonstrated only

with their names. The MetaModelEdge connection is depicted using the assigned source and target node and an arrow indicating the direction.

```
1  GmmMetaModel metaModel:
2    - layers = {business, application, infrastructure}
3    - nodes = {business service, business process, application service, application
         component, infrastructure service, system software, device}
4    - edges = {
5        realization:
6          - connections = {
7              [business process -> business service],
8              [application component -> application service],
9              [system software -> infrastructure service]
10           }
11       used by:
12          - connections = {
13             [application service -> business process],
14             [infrastructure service -> application component]
15           }
16       assignment:
17          - connections = {
18             [device -> system software],
19             [node -> systemsoftware]
20           }
21       }
```

Listing 3.1: Meta model for the RentalCar example.

First in line 2, the three layers are defined, followed by the meta model nodes representing the element types used within the example. Finally, in line 4 the specification of the edges begins. The MetaModelEdge *realization* is used within three different contexts in the EA model. This is represented with the three different MetaModelEdgeConnections. The first one describes the realization between *business processes* and *business services*, the second one between *application component* and *application service* and the last one between *system software* and *infrastructure service*. The definitions of the relation types *used by* and *assignment* are done accordingly.

3.2.2 GMM model

As mentioned above, the GmmModel is used to specify the instances of the type declarations in the GmmMetaModel. Figure 3.4 provides an overview of the proposed structure.

For each element $e \in E$ a ModelNode along with the respective ModelProperties $p \in P$ with $EA_p(p) = (pt, e)$. Each property possesses a *value* field that accommodates the property's data. The given property type pt within the function D' is represented with the stereotype association to the respective property type $pt \in PT$ in the meta model. The element type et of the element e provided by the function I_e is also depicted with a stereotype association between e and et.

The relations $(e1, e2, rt) \in R'$ between the elements are defined with the concept ModelEdge. For each ModelEdge the respective source and target element $e1$ and $e2$ is defined. The utilized relation type rt is specified using a stereotype association to the relation type $rt \in RT$ from the meta model. Additionally, ModelProperties describing properties of the model edges are specified in the same manner as for model nodes.

To enable the associations to meta model elements, all three types, the ModelNode, the ModelEdge and the ModelProperty inherit from the supertype StereotypedElement. Each StereotypedElement has a property MetaModelStereotype which provides the in-

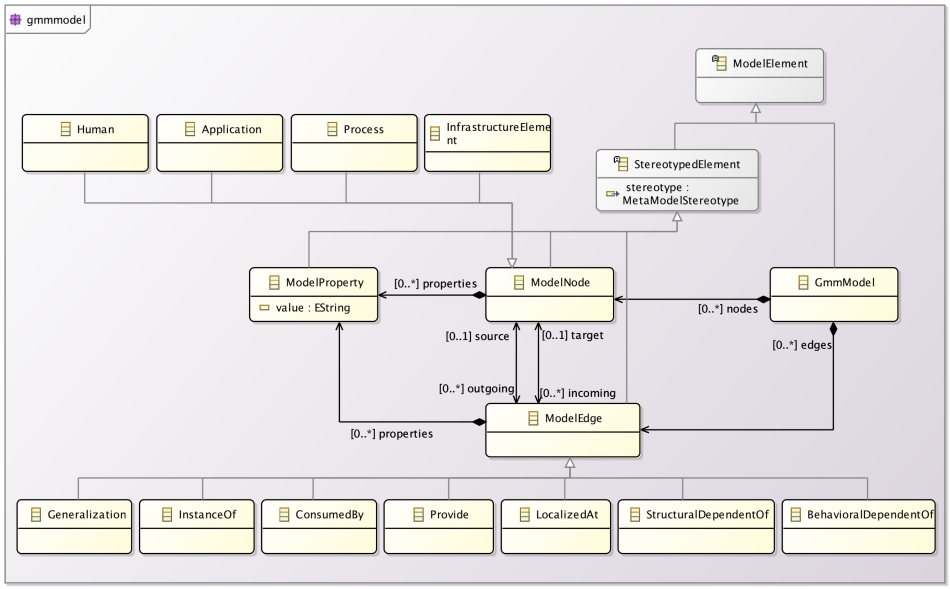


Figure 3.4: Details of the GMM model package.

stantiated meta model element. This instantiation mechanisms ensures that the proposed conditions at the beginning of section 3.2 are held.

Relationship categorization

At this point the GMM enables the generic representation of arbitrary EA models. The `ModelEdges` with their source and target specification provide a navigation structure between the `ModelNodes`. The stereotype mechanism extends model elements with their respective element types, relations types and property types. However, the GMM does not provide any EA specific information, as demanded within the requirements *R6* and *R12*, at this point.

Therefore, we employ a class categorization approach similar to [vKG03]. The authors propose three general types of relationships for their semiautomatic impact analysis on traces. Since the proposed classes (representation, refinement and dependency) are not sufficient for EA models, we analyzed current literature for common respectively foundational concepts used within the EA domain. This includes the ArchiMate Generic Model (ArchiMate GM) [The17], the DoDAF Meta Model (DM2) conceptual data model of DoDAF [Dep10] and the requirements for an EA meta model proposed in [BBJ+11].

Overall, we were able to identify seven classes of relevant EA relationship types: *Located at* denotes the allocation to some location or organization unit. Any kind of providing functionality, information or behavior is captured with the type *provide*, while the *consumed by* class denotes the consumption of those elements. *Structural dependent on* relationships define the structure or organization of entities within a single layer. The *behavioral dependent on* class summarizes relationships which declare dependencies between the behavior of elements in a single layer which are neither of the type provide nor consume. The *instance of* class is used to support multi-level modeling within the EA. Finally, the *generalization* relationship provides a category for relationships depicting inheritance hierarchies.

Table 3.2 lists all categories along with corresponding examples from the ArchiMate GM,

Table 3.2: Categorization of EA relationship types.

relationship class	*ArchiMate GM* [The17]	*DoDAF DM2* [Dep10]	*Req.* [BBJ+11]
`located at`	assigned to	is-at	
`provide`	realizes, accesses	produces, realized-by performed-by	
`consumed by`	used by, accessed by	consumed by	
`structural dependent on`	aggregated by, composed by	whole-part	hierarchies
`behavioral dependent on`	triggered by, flow from	before-after	
`instance of`		type-instance	type-item
`generalization`	specialization	super-sub-type	generalization

the DoDAF DM2 conceptual data model and requirements identified in [BBJ+11]. Note that the mapping in table 3.2 is a suggestion based on our interpretation and can be adapted if an organization assigns different semantics to these relations.

Within the GMM these seven types are modeled as sub types of the generic `ModelEdge`. According to a given mapping table, the relations of an EA model are instantiated with one of these types. If no mapping is provided, `ModelEdge` is used as default type. The relation categorization enables more expressiveness during analysis specification, while keeping independent from the utilized EA meta model. Since the categorization is optional, the universality of the proposed GMM remains.

Element categorization

We propose a similar categorization approach for the elements of an EA model. Thereby, we utilize the general concepts proposed within the meta framework GERAM [IFI99] which are human-oriented concepts, process-oriented concepts and technology-oriented concepts. In addition, we further distinguish the technology-oriented concepts and propose the two categories *Application* and *Infrastructure Component* instead. Human-oriented concepts are represented through the category *Human* and process-oriented concepts through the category *Process*. Element types representing humans or the role of humans, processes, applications and infrastructure components can be found in most of the EA frameworks (e.g. [The17], [The18], [WF06], [Dep10]). The proposed element categories represent the key concepts used within the core layers, business, application and technology, defined in ArchiMate [The17].

3.3 Example Application

To illustrate the use of the GMM we select the two model elements *Reservation by phone* and *Booking a car* with their *realization* relationship from the example provided in figure 3.2. The representation of this relation and the two elements using the GMM is illustrated in figure 3.5. The left-hand side shows the type definitions within the *GmmMetaModel*, while the right-hand side contains the model data.

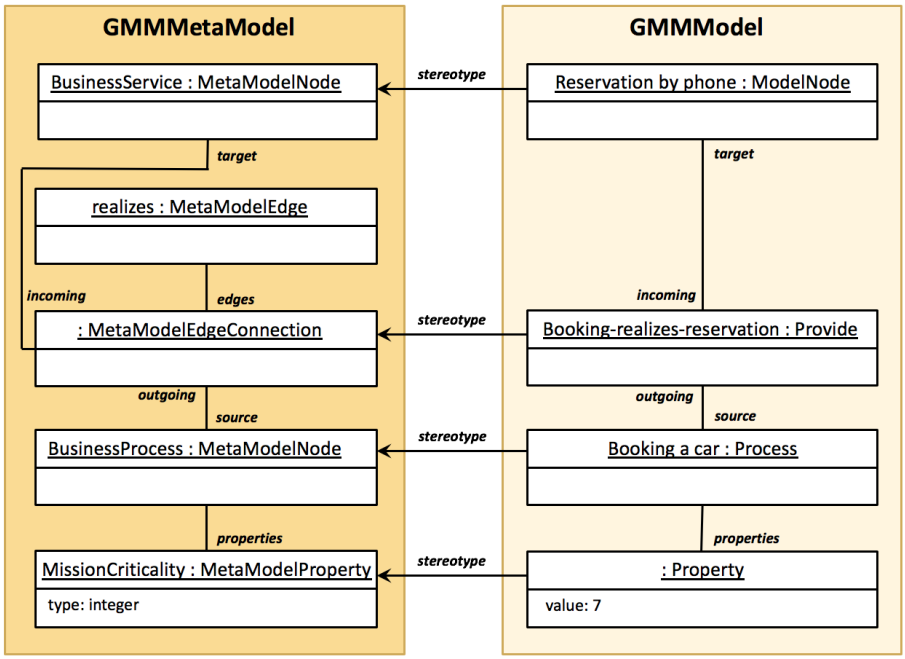


Figure 3.5: Example for the instantiation of the generic meta model.

For the two elements a respective `ModelNode` is created. Thereby, for the business process *Booking a car* the specialization `Process` is used. For a business service like *Reservation by phone* no specialization is provided within the GMM and the generic `ModelNode` is used. The types of the two elements, i.e. *BusinessProcess* and *BusinessService*, are modeled as `MetaModelNode`. With the `stereotype` relationship their instantiation according to the function I_e is provided.

For the business process *Booking a car* the mission criticality is provided as a `Property` of the element. The `stereotype` relation depicts the respective meta model element, i.e. the `MetaModelProperty`. The `MetaModelProperty` provides the name (*mission criticality*) and the type (*integer*) of the property

The concrete relationship *Booking-realizes-reservation* is represented within the GMM as `ModelEdge`. Thereby, the specialization `provide` is used, since it covers the semantics of the *realization* class. The relation type is depicted with a `MetaModelEdgeConnection` which belongs to the `MetaModelEdge` *realization*. The `MetaModelEdgeConnection` specifies the source and the target meta model nodes of this specific relation type. Source is in this case the *business process* and target the *business service*. A *stereotype* association defines the type usage between the specific model edge and the edge connection.

3.4 Converting Architectural Data to the GMM

In the previous sections we presented the GMM as a representation format for enterprise architecture models. The data used to create such a model can be extracted from different sources. Main source is the utilized EA tool within an organization. These tools provide detailed data about business, application and infrastructure elements on an enterprise level. Interfaces enable the export of the manually captured data which enables their further processing. If the EA maturity is still low, the relevant EA data is typically provided within Excel or CSV files. Additionally, more detailed information can be extracted from Configuration Management Databases for the infrastructure layer or Application Performance Monitoring (APM) tools for the application layer. Due to the universality of the GMM, it is possible to represent all these different sources.

The proposed conversion procedure is depicted in figure 3.6. In order to map a data source to the GMM, a corresponding adapter has to be created once. This adapter captures the exported data from the source and converts it into the GMM representation. Thereby, an optional mapping of the element and relation types to the different element and relation classes can be performed. The assignment of classes to types has to be done manually in advance and is then provided during the actual transformation. Once the adapter and the mapping are defined the architectural data can be converted, no further adaption is required by the user.

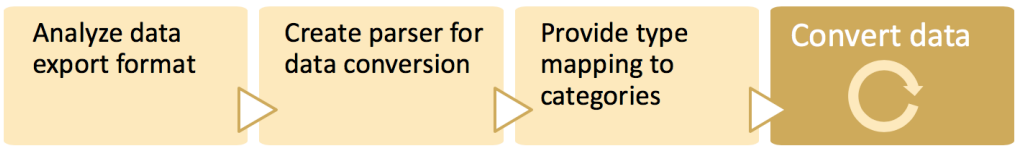


Figure 3.6: Translating an EA model into the GMM.

Within this thesis, we provide predefined adapters for the EA modeling tool Innovator [MID19], the EA modeling Tool Archi [Bea19] and for CSV files. To ensure the up-to-dateness and to provide more details within the application layer, we also consider communication data provided by logs or monitoring tools. This enables the creation of an architectural model, representing the actual behavior of the system. We do not have to estimate performance measurements like the number of messages or the message size. Sources for communication data are the used middleware, APM tools or log data.

APM tools like Dynatrace [Dyn19a] or Instana [Ins19] monitor the registered application servers and provide performance measurements for the applications including incoming and outgoing connections. The captured data can be exported and converted into the GMM, as it is done for EA tools. If such a tool is not available, existing log messages can be used to reconstruct the communication dependencies between the applications. The log data provides information about the source and target of a communication dependency as well as its duration and data volume.

Middleware software like a message queue or an enterprise service bus also provide interfaces to access communication data and thus to extract dependencies between application systems. Within this thesis we provide solutions to extract the data from the message queue solution RabbitMQ [Piv07] and the enterprise service bus Orchestra [sof18].

These possibilities of capturing the communication data typically result in a flat map of application systems and their communication dependencies. This model is enriched with

actual performance data regarding response times and data volumes. After establishing the architectural model, a review has to ensure the quality of the model. The recovered architectural model contains only those services that send or receive a message during the considered time span. Further logical aggregation or assignment to business functions have to be done manually. As alternative, merging these data with data from EA tools provides a holistic and up-to-date model.

When implementing a dynamic approach for data retrieval, it is important to set an adequate time span for data retrieval, since only the communicating services are captured. Additionally, the data retrieval procedure must be performed in a way that it has no significant impact on the productive system. Especially the performance must be ensured, when integrating such an approach. To deal with the heterogeneity in current application landscapes, the data retrieval must also be independent from the used technologies and programming languages. Additionally, the necessary extensions have to be performed without changing the source code. The last point is important to ensure the practicability and acceptance.

3.5 Related Work

As described in section 2.1.2 there exist several proposals for EA meta models in current literature but no standard. Organizations often use the proposed meta models only as starting point and adapt them to their individual needs [AKRS08]. This requires a generic representation format of these models in order to enable reusable analysis functions. Providing just another EA meta model with specific element and relation types would not result in a practicable approach since it does not consider the individual needs of the organizations.

To cope with the variability of meta models during analysis, Jonkers and Iacob [JI09] propose a differentiation between design space and analysis space. The design space is expressed by using languages like UML, business process modeling languages or architectural description languages. The analysis space is expressed by using special-purpose languages which enable the later analysis. Their analysis approach includes a model transformation, followed by the execution of the analysis, and finally a reverse transformation back to the design space (including the analysis result). Thereby, each analysis is built upon its own meta model and thus for each analysis a new model transformation is required.

Kurpjuweit et al. [KA09] provide a formalism for EA meta models and EA models in order to realize a generic and applicable dependency analysis. Within their formalism they do not consider properties, neither for elements nor for relations. Properties are important for later quantitative analysis. Additionally, they do not explicitly support relationships to enable multi-level modeling and packaging. In contrast to our proposal, they explicitly model the inverse of a relationship. We created a relation type only for one direction. The reification of a relation as a class makes both directions accessible, the incoming and outgoing one.

The ArchiMate GM as well as the proposed element hierarchy [The17] and the DM2 conceptual model as well as IDEAS Foundation of DoDAF [Dep10] propose a small set of common concepts that can be used as foundation to describe an EA model. Both rely on one foundational unit to describe an EA model which is represented with the *ModelNode* within the GMM. But regarding the relationship and element classification both approaches are not sufficient for EA model representation for analysis tasks. The generic elements proposed within DoDAF DM2 are already too detailed to form a common foundation for representing different EA models. [Dep10] provides a more abstract solution, but still lacks a generic navigation mechanism. The proposed basic relationships within the IDEAS Foundation, i.e. wholePart, superSubType, typeInstance, and beforeAfter, can be mapped straightforward for the proposed relationship classes within the GMM. Additionally, the GMM provides classes for provisioning, consuming and location.

The relation super types within ArchiMate [The17], *structural*, *dependency*, *dynamic* and *other*, were also not detailed enough. The align-enable principle is not supported since realization and composition relationships are aggregated in one category. Additionally, the important *type-instance* relationship is missing. The differentiation only between structure and behavior element is also not detailed enough. The more detailed approaches within the Generic Metamodel of ArchiMate [The17] has a strong focus on service-orientation. Services with their interface are the mean to provide and use functionality.

Meta meta languages like the domain independent Meta Object Facility (MOF) [Obj16, Obj17] or the domain specific Multi-prespective Enterprise Modeling (MEMO) modeling language [Fra11, Fra14] provide no sufficient foundation for the representation of EA

models for later analysis purposes. Broadly speaking, these meta meta models enable the specification of classes, properties and relationships. These concepts allow the representation of EA models as well as their meta models, but the structure is not sufficient for analysis definition. The approaches provide no EA specific constructs that enable a generic definition of complex analysis. The expressiveness of analyses relying only on the semantics of classes, properties and relationships is not sufficient.

Buckl et al. propose in [BBJ+11] several requirements for a meta-language for EA information modeling. The proposed primitives of an EA model are classes, properties and binary associations. All of them are supported by the GMM. The demanded generalization and hierarchy modeling, multi-level modeling and packaging relationships are addressed with the specializations of `ModelEdge`. To enable properties for relations, they are explicitly modeled for the `ModelEdge` and `MetaModelEdge` classes. Additionally, the datatype for the properties should be provided. Further requirements addressing the specification of constraints or the explication of dependencies between properties are not considered within the GMM. We argue that these issues are part of the analysis and not of the architectural representation. Supporting intentional semantics, i.e. provisioning different names for a type or an element, can be depicted with the generic property annotation mechanism.

3.6 Conclusion

In this section we propose the Generic Meta Model (GMM) to enable the representation of an arbitrary EA model for later analysis purposes. The GMM provides a technical foundation to abstract from the specifics of the different EA meta models used in organizations, while providing a uniform model structure to enable reusable analysis functions. The basic idea of the GMM is to represent the EA as a stereotyped graph. Thereby, the GMM incorporates the model data (i.e. elements with their relations and properties) as well as the meta model data (i.e. the element relation, and property types). Therewith, the GMM can be applied to architectural models that are defined in an object-oriented manner, i.e. models which distinguish between type definitions and instances.

To provide additional semantics about the elements, the GMM provides an optional categorizations mechanism for elements and relations. Therefore, we identify common concepts used within EA models. For relations we identified the seven classes: *provides*, *consumed by*, *located at*, *structural dependent on*, *behavioral dependent on*, *instance of* and *generalization*. The identified classes for elements are *Human*, *Process*, *Application* and *Infrastructure Element*. Each element or relation can be assigned to one of them by using one of the specializations of `ModelEdge`. We decided against the implementation within the meta model or with stereotype to better support performant out-of-the-box interpretations during analysis tasks.

To convert existing data into the GMM format, a respective parser including an optional mapping of the element and relation types to categories has to be created once. In the following, the data can be imported without further adaptions. If the EA meta model does not change the adapter can also be used to update the existing model. In this case, new elements are added and property values are updated but no information will be deleted. Main source for data is the provided exports from EA tools or information within CSV files. Additionally, to provide further details about the applications and their dependencies, also application monitoring tools and communication log data can be considered. In this case the frequency and data volume does not have to be estimated, and the created model represents the current state. Ideally, this data is combined with the static data extracted from EA tools.

4

Enterprise Architecture Analysis Definition

Within this chapter we consider current analysis approaches within EAM and derive common requirements for analysis activities. Based on these results we develop the Architecture Analysis Language (Arla). Arla is a declarative language for analysis specification that provides a uniform interface to the different analysis activities within EAM. Therefore, different analysis classes are proposed that can be adapted to the individual concerns of the stakeholders.

The foundational analysis of EA approaches utilizes results retrieved within the bachelor thesis [Rau13] and the master thesis [Rau15]. Both were supervised by the author of this thesis. The results provided in [Rau15] were additionally published in [RLB16, RLB17]. The idea and concepts of Arla were previously published in [LB17].

4.1 EA Analysis Approaches

In [Rau13] we considered 105 analysis approaches for EAM which we roughly clustered into 40 analysis types. The types are created based on the scope of the respective analyses. The goals and execution methods of the analysis approaches assigned to one type can differ. Examples of analysis types are *Analysis of Service Response Time* (e.g. [NBE14]), *Change Impact Analysis* (e.g. [SKR13a]) and *Performance and Workload Analysis* (e.g. [Lan12]). Figure 4.1 provides an overview of the 40 analysis types. The color indicates their degree of development. *Blue* indicates a high level of maturity. *Red* indicates a low maturity, i.e. there is not enough information or detail given to apply the approaches of this analysis type.

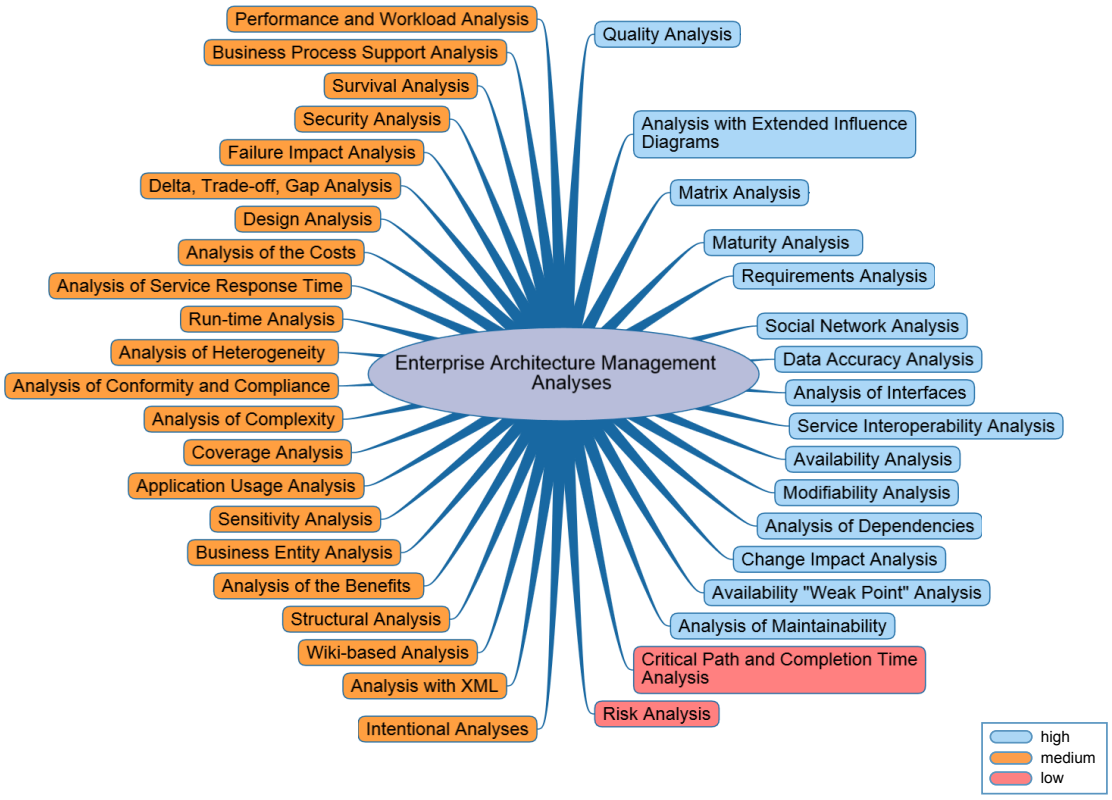


Figure 4.1: Enterprise architecture analysis types with their degree of development [Rau13].

14 of these analysis types reference quantitative analysis approaches, whereas eight types answer functional questions, e.g. the effects of changes. Another 14 types cannot be categorized solely in one of those dimensions. For three types no assignment was possible.

Based on the initial typing we propose a two-dimensional categorization of EA analyses in order to derive typical characteristics requirements for analysis execution. Each analysis approach is assigned to one technical dimension and at least one functional dimension. A technical dimension summarizes approaches utilizing a similar method with similar steps regardless of the addressed goal and subject. In contrast a functional dimension summarizes approaches addressing the same objectives and goals. Since an approach can fulfill different goals, there are several approaches with more than one functional dimension. For

example, the cost analysis of [Nie06] is assigned to the technical dimension *KPI* and the functional dimension *Financial*. The quality analysis of [NSJ+08] is assigned to the technical dimension *Bayesian Networks* and to the functional dimensions *System, Attribute, Quality* and *Data*. In total we identified 10 functional dimensions and 17 technical dimensions. The functional dimensions are briefly presented in the following:

Quality	Quality of different aspects is analyzed (e.g. usability).
Attribute	Analysis of specific attributes like availability.
Dependencies	Focus are the relationships between the elements.
Effects	Capture the effects of changes.
Business objects	Addressing all kind of business objects (e.g. operations, artifacts).
Design	Addressing the overall design of the enterprise architecture.
Requirements	Validate the requirements for goal fulfillment.
System	Object under analysis are (parts of) systems.
Financial	Metrics for cost benefits, risks and weak points.
Data	Analysis of the quality and accuracy of data.

The technical dimensions relying on probabilistic technologies are further summarized within the *Probabilistic dimensions*. Also dimensions that utilize quantifying techniques for answering the targeting questions are summarized. Additionally, the dimensions *Matrices* and *Design* are mentioned together, since both utilize matrices during analysis execution. The dimension AHP is short for the technique Analytic Hierarchy Process. Therewith, the following overview of the technical dimensions can be provided:

Probabilistic dim.	Includes *Bayesian Networks*, *EID* and *PRM*.
Quantitative dim.	Includes *Metrics and KPIs*, *Business entities*, *Time evaluation* and *Weak points*.
Matrices and *Design*	Utilize matrices during analysis execution.
Comparisons	Compares scenarios, processes, attributes and dependencies.
Views	Creation different perspectives to analyze aspects in detail.
AHP	Define theories about attributes and quality aspects of EAs.
Tree	Utilize trees to analyze dependencies and quality features.
Social network	Analyze questionnaires and email data to build network graphs.
Lifecycle	Determine dependencies according to different lifecycle phases.
Ontology	Assess the ontological representation of an EA.
Structural	Observe obstacles of different architecture versions.

The assignment of the dimensions to the analysis approaches is described in detail in the master thesis [Rau15] and the publications [RLB16, RLB17] building on it. A summary of the assignment is provided in table 4.1. This provides an overview of the functional dimensions, and which technical dimension are used for their realization. Additionally, the number of analysis approaches within each dimension is presented. Thereby, each publication is interpreted as one analysis approach. An analysis approach can be assigned to several functional dimensions. Thus, approaches may be counted twice or three times. The large number of probabilistic approaches is due to the strong publication effort from one research group on this topic. The publications, used within the counting in [Rau15], extend and rely on each other.

A general requirement that occurs across all dimensions, is the availability of one or more concrete EA models which are subject to the analysis. Additionally, the utilized meta model has to be given as well as the goal of the analysis. Identified goal types within the approaches are percentage, probability, number, matrix, dependency, object, effect,

Table 4.1: Dependencies between functional and technical categories [Rau15].

Technical \ *Functional*	*Quality (31)*	*Attribute (28)*	*Dependencies (19)*	*Effects (17)*	*Business objects (16)*	*Design (12)*	*Requirements (10)*	*System (8)*	*Financial (7)*	*Data (7)*	*Other (4)*
Probabilistic dimensions (29)	x	x	x	x				x		x	
Quantitative dimensions (17)	x	x			x				x		
Matrices (incl. Design) (14)	x		x	x		x			x		x
Comparisons (9)		x	x	x	x	x					
Views (6)				x	x		x				x
AHP (5)	x	x									
Tree (4)	x	x	x		x						
Social network (4)					x	x					
Lifecycle (2)			x				x				
Ontology (2)			x	x	x	x					
Structural (1)						x					
Other (3)	x										x

scenario and Boolean. Despite these input parameters the obvious precondition must be hold that the relevant data for analysis is available within the EA model. To be able to execute the analysis, some approaches require additional information besides the EA model. These are for example chance and utility nodes and their causal relations for analysis utilizing extended influence diagrams. The final evaluation is merely done using metrics and scales, and about 30% of the analysis approaches utilize probabilistic techniques. A further important technique is the definition of views on the architecture. Additionally, the functional category *Dependencies* is often used.

The approaches within the probabilistic dimensions require additional information and dependency structures as well as probability distributions for attributes. Gaining this information provides additional effort and the specifics of these approaches makes it hard to provide a usable analysis definition language to the architect. Therefore, we focus in the following on the analysis approaches assigned to the remaining technical dimensions.

Especially we concentrate on *quantitative* dimensions and **views**, as they belong to the more important dimensions. In specific, we implement a **performance analysis** to enable the evaluation of the run time behavior of a system. Additionally, we integrate an **impact analysis**, a **dependency analysis** and a **gap analysis** for the comparison of two models. The analysis language should also enable the **composition** of analysis approaches to support more expressive analyses. This enables the calculation of metrics only for a specific part of the architecture which is defined with another analysis specification.

The result of an analysis evaluation can either be calculated for a specific set of model elements or be a single result for the whole architecture. For example, infrastructure costs can be calculated for each infrastructure element but also a total value for the whole infrastructure. Result types of an analysis evaluation can be

- a numeric value (e.g. percentage, probability, or number),
- a single model element,
- a set of model elements representing a part of the architecture,
- an attribute value (e.g. effect type or Boolean), or
- a dependency, i.e. a path between two elements.

4.2 Language Overview

The *Architecture Analysis Language Arla* is a textual, domain specific language (DSL) for analysis definition. The language provides a uniform interface to different analysis approaches. Arla provides a declarative approach for analysis specification and abstracts from technical details. The architect defines only "what" he is interested in, how this information is retrieved, is generated from the Arla analysis definition. This execution process, utilized for the evaluation of the language, is described in section 5.5.

For the language development we used Xtext [Ecl18b], a framework that comprises a powerful language for the description of textual languages. The grammar language provided by Xtext enables the description of the concrete syntax of a textual DSL as well as its mapping to the semantic model. The model as well as a parser, linker, type checker and compiler are generated by the framework. The DSL was developed according to the meta model development process for abstract syntax development from [BCW12]. This incremental and iterative process consists of three phases: The Modeling Domain Analysis phase, elaborating the purpose and content for the language, the Modeling Language Design phase, defining the meta model, and the Modeling Language Validation phase, verifying the correctness and integrity. The results of the first phase, a description of the different analysis types and their characteristics, were already presented in the previous section. The results of the second phase are presented in the following, and the validation of the language is presented in section 8.

Arla supports the specification of **specific analysis definitions** for a concrete EA model, but also of generic **template definitions** using placeholder variables. In order to apply a template on a concrete EA model, the declared variables have to be mapped to existing stereotypes. A re-definition of the analysis is not necessary.

Another concept to ease the re-use of analyses and support template definition is node and edge categorization as proposed within the GMM in section 3.2.2. The proposed classes are used to abstract from EA specific stereotypes during analysis definition. We identified seven relation classes: `provide`, `consumed by`, `located at`, `structural dependent of`, `behavioral dependent of`, `generalization` and `instance of`. Additionally, we identified four element classes that are used in most EA meta models: `Human`, `Process`, `Application` and `InfrastructureElement`. If the categories are insufficient for an analysis definition, it is always possible to use the concrete stereotypes.

An analysis or template definition is composed of a general part and an analysis specific part. The general part comprises attributes like the name, the analysis style, the result type and a description. Within the analysis specific part, the configuration information necessary for analysis execution is provided. The language is designed in a modular way, i.e. complex analyses are defined through the composition of simpler ones. The language supports the following customizable analysis classes:

Metric Calculation	Calculates a metric for each element or for the whole architecture
Scope Analysis	Defines a part of the model, i.e. a view
Path Analysis	Calculates available dependencies between two elements according to provided restrictions
Impact Analysis	Calculates the effects of an event to the architecture
Gap Analysis	Compares two models and determines the differences

```
1 Template ApplicationStructureViewpoint {                          Header
2 "The Application Structure viewpoint shows the structure of one or more
      applications or components. This viewpoint is useful in designing or
      understanding the main structure of applications or components and the
      associated data."
3 as Aggregate Modelelementset                                      Body
4 defined with set definition:
5   nodeClass: Application              OR
6   nodeType: "Application interface"   OR      Configuration
7   nodeType: "Data object"             OR
8   nodeType: "Application collaboration"
9 }
```

Figure 4.2: Arla template for a scope definition.

Performance Analysis	Calculates several performance metrics
Composed Analysis	Enables the composition of analyses
Adapted Analysis	Provides the relevant mapping for executing a predefined template

Additionally, two further analysis classes are provided to enable the execution of native SPARQL queries or individual data-flow analyses. The classes can be used if a desired analysis cannot be captured with one of the seven previous ones.

The structure of analysis and template definitions is the same. The specification of a template definition is exemplary shown in figure 4.2. For its illustration the definition of the *Application Structure Viewpoint* from ArchiMate [The17] is used. This analysis can be realized using the *Scope Analysis* class. Within the template definition the conditions for executing the scope analysis are defined. A template definition, respectively analysis definition, is composed of three parts. Within the `Header` (lines 1, 2) the name and description of the analysis are defined. The header of a template starts with the signal word `Template`, an analysis definition with the signal word `Analysis`. Within the `Body` (lines 3 - 8) the details of the analysis are provided. First, in line 3 the analysis style and the result type are defined. `Aggregate` states that this analysis definition will provide one result, and this result is of the type `Modelelementset`. All elements within the `Modelelementset` are part of the resulting view. Afterwards the constraints for analysis execution are defined within the `Configuration` part (line 4-8). The configuration is specific for each analysis class.

In the example, the view should contain the elements having the stereotype *application interface*, *application component*, *application collaboration* or *data object*. This can be formulated with a `set definition`, where these concepts can be listed using the `OR` operator. Thereby, the class annotation `Application` is used for *applications components* and `nodeType` references are used for the other element types. The three `nodeType` statements provide only placeholder variables. In order to execute the analysis on a specific EA model, these variables have to be mapped to concrete stereotypes of the model. For understandability reasons it is recommended to give meaningful names to those variables, although every term is possible. Within specific analyses instead of variables the concrete stereotypes of the EA model are used.

Figure 4.3 provides an overview of the meta model utilized for the analysis language. Arla consists of three packages. `Arla Core` (in the middle and at the bottom, green) defines the parts that are used by both analyses, the specific analysis definitions and the template

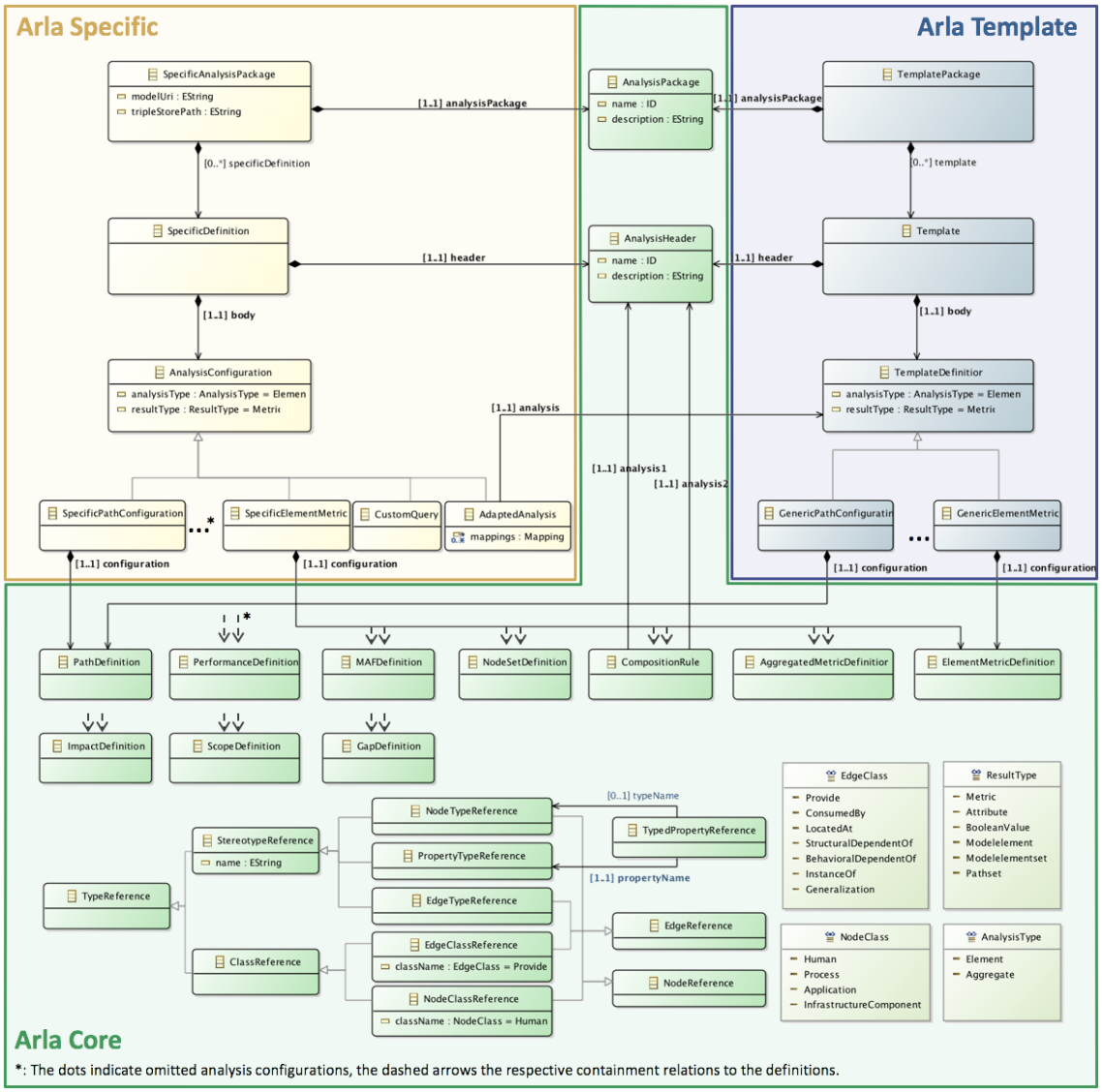


Figure 4.3: Simplified Arla Overview.

definitions. `Arla Specific` (at the left side, yellow) defines the constructs for specific analysis definitions, whereas `Arla Template` (on the right side, blue) defines the constructs for template definitions. The structures of Arla Specific and Arla Generic are very similar to each other. The definitions are provided within a `SpecificAnalysisPackage` respectively `TemplatePackage`. Each definition is composed of a `Header`, defined in the core package, and a `Body`. The body provides the configuration for the analysis and is individual for each analysis class. Omitted analysis configurations within both packages are indicated with "..." and dashed incoming relations. The commonalities for the configuration of each analysis class is provided in a respective `Definition` in Arla Core. The details of the analysis definitions in `Arla Core`, are also omitted here and presented in section 4.3.

4.2.1 Arla Core

Within `Arla Core` the commonalities between template and specific analysis definitions are depicted. In specific, this is the `AnalysisHeader` of a definition. The header is used for the specification of the name and a description. The definitions are described with an analysis specification file, the `AnalysisPackage`. The package declares the name and the description for the file. The core package also defines enumeration types like `AnalysisStyle`, `ResultType`, `NodeClass` and `EdgeClass` and a generic construct for referencing element and relation types. The `AnalysisStyle` defines whether the analysis provides one result for the whole architecture (*Aggregate*) or a result for each element (*Element*). The `ResultType` defines the kind of result. Possible results are *Metric, Attribute, Boolean, Modelelement, Modelelementset* and *Pathset*. The result type and the analysis style are attributes of all analyses. But since they can have predefined values for some analysis classes, they are not specified within the `AnalysisHeader`. Additionally, for each analysis class the respective definition used for its configuration is depicted. This includes calculation rules for metrics, definition of a node set, rules for analysis composition, the definition of a scope, the configuration of paths, the configuration of an impact and the definition of a performance analysis. The details of the concrete definition classes are presented in 4.3.

Element, relation and property types are referenced with a `TypeReference`. A `TypeReference` can either a `StereotypeReference` or a `ClassReference`. The `ClassReference`, used for referencing node or edge classes, has a value field for the respective class enumeration item. The `NodeClass` and `EdgeClass` enumeration represent the categories proposed within the GMM in section 3.2.2. Using the categories enables the definition of generic analysis templates without further adaption to a concrete EA model. The categories can be used within both definitions, the specific analyses and the templates.

If the categories are not sufficient, it is also possible to use the `StereotypeReference` instead. This concept defines a concrete stereotype in case of a specific analysis and a stereotype variable in case of a template. `StereotypeReference` can be used to reference meta model nodes (`NodeTypeReference`), meta model edges (`EdgeTypeReference`) and meta model properties (`PropertyTypeReference`). For referencing the property of a specific node type a dedicated `TypedPropertyReference` is provided.

To enable the substitutable use of type and class references within analysis and template definitions, the `NodeTypeReference` and the `NodeClassReference` inherit from the interface `NodeReference` and accordingly for edge references from a `EdgeReference`. Additionally, the stereotype references, i.e. the `NodeTypeReference`, the `EdgeTypeReference` and the `PropertyTypeReference` are overwritten in `Arla Template` and `Arla Specific` to enable the adapted of templates within specific analysis definitions.

4.2.2 Arla Template

Source element of `Arla Template` is the `TemplatePackage`. The package consists of the generic `AnalysisPackage`, defining the name and providing a description, as well as of an arbitrary number of `Templates`. The respective grammar excerpts are presented in listing 4.1.

```
1 TemplatePackage:
2   analysisPackage = AnalysisPackage
3   (genericDefinition += Template)*
```

```
4   ;
5  Template:
6    'Template' header = AnalysisHeader '{'
7    'as' body = TemplateConfiguration
8  '}'
9  ;
10 TemplateConfiguration:
11   GenericImpactConfiguration | GenericEdgeConfiguration |
         GenericPathConfiguration |
12   GenericNodeSetConfiguration | GenericElementMetric | GenericAggregatedMetric |
13   GenericPerformanceConfiguration | GenericCompositionConfiguration |
14   GenericDFAConfiguration
15 ;
```

Listing 4.1: Grammar excerpt for `TemplatePackage`, `Template` and `TemplateConfiguration`.

A `Template` consists of an `AnalysisHeader` which defines the name and description for the template. Further details are described within the body of the template, the `TemplateConfiguration`. For each analysis class a specific configuration is provided. For metric calculation and scope analysis two different configuration possibilities are provided. Metric calculation differs between the configuration of element metrics and aggregated metrics addressing the whole architecture. The scope analysis can be configured with a `GenericEdgeConfiguration` or with a `GenericNodeSetConfiguration`. Gap analysis as well as custom SPARQL queries are not supported for template definition.

Listings 4.2 and 4.3 show exemplary two different template configurations. Within the first listing, the grammar excerpt for the configuration of a scope analysis using an edge definition is provided. Since the calculated result can only be one set of model elements, the values for `analysisStyle` and `resultType` are predefined. The actual analysis configuration is provided within the `EdgeDefinition`. Therein, the conditions required for analysis execution are defined. The `EdgeDefinition` is depicted within `Arla Core` and can be used for templates as well as specific analysis definitions. The details of this definition are introduced in section 4.3.1.

```
1  GenericEdgeConfiguration:
2    analysisStyle=Aggregate
3    resultType=Modelelementset
4    'defined with scope definition:' configuration = EdgeDefinition
5  ;
```

Listing 4.2: Grammar excerpt for the configuration of a scope analysis template.

Listing 4.3 shows the configuration for a composition of two analyses. In contrast to the previous one, the values for `analysisStyle` and `resultType` are not predefined. They cannot be specified at design time and have to be provided by the user when defining the analysis. As in the listing above, the actual configuration is provided with an element from `Arla Core`. This `CompositionDefinition` is presented in detail in section 4.3.9.

```
1  GenericCompositionConfiguration:
2    analysisStyle=AnalysisStyle
3    resultType=ResultType
4    'defined with composition rule:' configuration=CompositionDefinition
5  ;
```

Listing 4.3: Grammar excerpt for the configuration of an analysis composition template.

4.2.3 Arla Specific

The root element of `Arla Specific` is the `SpecificAnalysisPackage`. The respective grammar excerpt is presented in listing 4.4, line 1 - 6. Within the package the concrete EA model which is subject to the analyses, has to be provided. The EA model is identified with the path of the triple store as well as the relevant URIs. The generic `AnalysisPackage` from the core provides the name and the description. Finally, an arbitrary number of specific analysis definitions (`SpecificDefinition`) can be added.

```
1  SpecificAnalysisPackage:
2    'Model' modelUri = STRING
3    'TripleStore' tripleStorePath = STRING
4    analysisPackage = AnalysisPackage
5    (specificDefinition += SpecificDefinition)*
6  ;
7  SpecificDefinition:
8    'Analysis' header = AnalysisHeader '{'
9    'as' body = AnalysisConfiguration
10 '}'
11 ;
12 AnalysisConfiguration:
13   SpecificImpactConfiguration |   SpecificEdgeConfiguration |
14   SpecificPathConfiguration | SpecificElementMetric | SpecificAggregatedMetric |
15   SpecificNodeSetConfiguration | SpecificPerformanceConfiguration  |
16   SpecificCompositionConfiguration | SpecificDFAConfiguration |
17   AdaptedAnalysis  | CustomQuery | GapConfiguration
18 ;
```

Listing 4.4: Grammar excerpt for the definition of a specific analysis package.

The `SpecificDefinition` (lines 7 - 11) and the referenced `AnalysisConfiguration` (lines 12 - 18), are constructed in the same way as the `Template` and the `TemplateConfiguration`. In addition to the configurations provided for templates there exist three more configuration types:

- the `AdaptedAnalysis`, used to customize a template,
- the `CustomQuery`, providing an interface for directly entering SPARQL queries, and
- the `GapConfiguration`, used to compare two models with each other.

Concrete `AnalysisConfigurations` are constructed in the same way as `TemplateConfigurations`. Exemplary, the configuration for a specific scope analysis is shown in listing 4.5.

```
1  SpecificEdgeConfiguration:
2    analysisStyle=Aggregate
3    resultType=Modelelementset
4    'defined with scope definition:' configuration = EdgeDefinition
5  ;
```

Listing 4.5: Grammar excerpt for the configuration of a specific scope analysis.

First, the details about the expected result are provided. The constraints for determining the desired part of the architecture are provided within the `EdgeDefinition`. The `EdgeDefinition` is defined in `Arla Core` and described in the subsequent section.

To enable a unique identification of the stereotypes, the generic `StereotypeReference` provided in `Arla Core` has to be extended with an identifier field. The `NodeTypeReference`, the `EdgeTypeReference` and the `PropertyReference` are extended with an additional `id` field. Therefore, these concepts from the core package have to be overwritten in the specific package (see listing 4.6).

```
@Override
StereotypeReference:
  NodeTypeReference | EdgeTypeReference | PropertyReference
;
@Override
NodeTypeReference:
  'node:' name = STRING '['id = STRING ']'
;
@Override
EdgeTypeReference:
  'edge:' name = STRING '['id = STRING ']'
;
@Override
PropertyReference:
  'property:' name = STRING '['id = STRING ']'
;
```

Listing 4.6: Grammar excerpt for for overriding the stereotype references in `Arla Specific`.

4.3 Analysis Classes within the Language

In this subsection the nine analysis classes of Arla are presented in detail. The classes are summarized in table 4.2.

Table 4.2: Analysis classes and their configuration definitions within Arla.

Analysis class	*Analysis/template definition*	*Configuration definition*
Scope analysis	`SpecificEdgeConfiguration` `GenericEdgeConfiguration`	`EdgeDefinition`
	`SpecificNodeSetConfiguration` `GenericNodeSetConfiguration`	`NodeSetDefinition`
Impact analysis	`SpecificImpactConfiguration` `GenericImpactConfiguration`	`ImpactDefinition`
Path analysis	`SpecificPathConfiguration` `GenericPathConfiguration`	`PathDefinition`
Metric	`SpecificElementMetric` `GenericElementMetric`	`ElementMetricDefinition`
	`SpecificAggregatedMetric` `GenericAggregatedMetric`	`AggregateMetricDefinition`
Performance analysis	`SpecificPerformanceConfiguration` `GenericPerformanceConfiguration`	`PerformanceDefinition`
Gap analysis	`GapConfiguration`	`GapDefinition`
Adapted analysis	`AdaptedAnalysis`	*mappings*
Custom analysis	`SpecificDFAConfiguration` `GenericDFAConfiguration`	`DFADefinition`
	`CustomQuery`	*query string*
Composed analysis	`SpecificCompositionConfiguration` `GenericCompositionConfiguration`	`CompositionDefinition`

Within the second column, for each class the respective configurations for specific analysis definition and template definition are provided. In the third column, the utilized configuration definition from the base package is provided. In the following, details for different analysis classes are introduced and the language part relevant for their configuration is depicted. This is done with example definitions and their execution on the RentalCar model.

4.3.1 Scope analysis

Within Arla the scope analysis is used to generate different views of an architecture or to define a domain architecture, i.e. partial EA models. Partial EA models representing the relevant views for decision-making are an important asset within EA [RGA07]. They support the senior IT managers in structuring and filtering the large amount of information provided with EA models. Domain architectures focus on one part of the architecture that is relevant for a specific stakeholder, i.e. the context of a business process for the business process owner [BvSF+10]. It is important that the domain architecture has a clear scope and that the related elements are included. Both generating views and generating domain architectures are based on the concept of viewpoints. A viewpoint addresses a specific concern of stakeholder which also determines the scope. Viewpoints can be used to either concentrate on the details of an aspect or the coherence's between elements.

Arla provides two different definition types for the configuration of viewpoints and to

generate the partial EA models. The NodeSetDefinition and the EdgeDefinition. The first one enables the definition of a viewpoint using constraints which an element has to fulfill to be part of the resulting set of model elements. For example, this can be the condition of having a specific node type. In contrast an EdgeDefinition defines a viewpoint for a concrete architecture element with constraints about the incoming and outgoing relations. For example, this enables the definition of a realization viewpoint through including all elements related with a direct or indirect *realization* relation.

Listing 4.8 provides an example for a scope analysis using the NodeSetDefinition. In listing 4.7 the grammar for the NodeSetDefinition is presented.

As mentioned above, a NodeSetDefinition is a composition of conditions an element must fulfill to be part of the result. These conditions can make statements about node types, properties and relations (line 9). It is also possible to create nested conditions. The conditions can be composed with an AND or an OR operator (lines 4 - 7) which are interpreted as logical operators. I.e. either an element must fulfill all conditions that are composed with AND or it must fulfill one of the conditions composed with OR.

```
1  NodeSetDefinition:
2    Composition
3  ;
4  Composition returns NodeSetDefinition:
5    Condition (({AndComposition.left=current} 'AND' |
6    {OrComposition.left=current} 'OR') right=Condition)*
7  ;
8  Condition returns NodeSetDefinition:
9    '('NodeSetDefinition')' | NodeReference | PropertyCondition | RelationCondition
         | NotCondition
10 ;
11 PropertyCondition:
12   'having property' propertyName = PropertyReference
13   ('with value' propertyValue = ValueExpression)?
14 ;
15 ValueExpression:
16   {StringValue} value = STRING | '('{NumericValue} op=Operator value = INT')'
17 ;
18 enum Operator:
19   EQUAL = '=' | LESS = '<' | GREATER = '>' | LESSorEQUAL = '<=' |
         GREATERorEQUAL = '>='
20 ;
21 RelationCondition:
22   'having relation to' '(' nodeSet = NodeSetDefinition ')'
23 ;
24 NotCondition:
25 'NOT' '(' nodeSet = NodeSetDefinition ')'
26 ;
```

Listing 4.7: Grammar excerpt for the NodeSetDefinition.

The simplest condition is the NodeReference. This enables the specification of the node type an element must have to be included in the result. Using the PropertyCondition (lines 11 - 14) the presence of a property with a specific property type for an element can be depicted. Optionally, a concrete value can be provided for this property. This can be either a specific String value or a numeric expression (line 15 - 17). The operator used within the numeric expression enables a statement whether the element must exactly have the given numeric value or if it has to be *less*, *less or equal*, *greater* or *greater or equal* than this value (lines 18 - 20).

The RelationCondition (lines 21 - 23) enables the definition of another NodeSetDefinition. All elements that have at least one relation to one of the elements of this nested nodeSet are included in the result. The elements of the nested nodeSet are not part of the results. This condition is important to allow statements like '*all business units that have a relation to a business process with criticality greater than 3*'. The respective NodeSetDefinition is presented in listing 4.8.

Finally, the NotCondition can be used to negate a node set definition (lines 24 - 26). The result consists in this case of all elements that do not meet the provided condition. An example for its application is the specification of all elements being not of a specific type.

```
1 Template ANodeSetExample {
2   "All business units that have a relation to a business process with criticality
        greater than 3"
3   as Aggregate Modelelementset
4   defined with set definition:
5     nodeType:"business unit" AND
6     having relation to
7       (nodeType:"business process" AND
8        having property propertyType:"criticality" with value (> 3))
9 }
```

Listing 4.8: Example template for a NodeSetCondition.

The second alternative for the configuration of model parts is the EdgeDefinition. The respective part of the grammar is presented in listing 4.10. This configuration type calculates a set of model elements with respect to a selected start element. According to the provided constraint for the relation types, additional elements will be included in the resulting model element set. Constraint types for relations are single, transitive and none. Assigning a single constraint to a relation type or relation class means that it should be considered once to determine the model part, i.e. the target element is within the result set, but none of the further related elements. In contrast transitive means that the target element is in the result set and if further relations met one of the conditions, those indirect related elements are also included.

```
1 Template AEdgeDefinitionExample {
2   "Determining the use by and realize context
        for an element."
3   as Aggregate Modelelementset
4   defined with scope definition: {
5     ModelEdge   in: none   out: none
6     ConsumedBy  in: none   out: single
7     Provide     in: none   out: transitive
8   }
9 }
```

Listing 4.9: Example template for an EdgeDefinition.

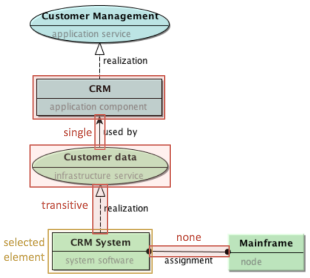


Figure 4.4: Result for the scope definition.

Listing 4.9 provides an example template for the scope configuration with an edge definition. This definition determines the *use* and *realize context* for an element, i.e. the direct and indirectly connected elements that are provided or consumed by the start element. Therefore, the single constraint is assigned to the relation class ConsumedBy and the transitive constraint is assigned to the class Provide. All other relations should not be considered for determining the resulting model part. The meaning of the constraints is illustrated with the example provided in figure 4.4. In the figure, the *CRM System* is the selected start element for the scope analysis. Since the *realization* stereotype belongs

to the category Provide the relation should be considered in a transitive way, i.e. the target element *Customer Data* is in the result. The further relations of this element, will be considered too. The outgoing relation *used by* belongs to the category ConsumedBy and is therefore considered in a single manner. That means, the target element *CRM* is added to the result set, but further relations of this element are not assessed. Finally, the *assignment* relation from the *CRM System* is not considered, since it is evaluated according to the constraint given for ModelEdge.

The part of the Arla grammar defining the EdgeDefinition is presented in listing 4.10. The constraints can either be annotated at concrete stereotypes using the ScopeDefinitionByStereotype or at relation classes using the ScopeDefinitionByClasses. The ScopeDefinitionByClasses requires a mandatory specification of constraints for ModelEdges (lines 5 - 6). This determines the default execution procedure. Further constraint assignments to relation classes are optional (line 7). In all cases, the constraints must be defined for incoming (in) and outgoing (out) relations.

```
1  EdgeDefinition:
2    EdgeDefinitionByStereotype | EdgeDefinitionByClasses
3  ;
4  EdgeDefinitionByClasses: '{'
5    'ModelEdge' 'in:' defaultEdgeIncoming=ScopeValue
6                'out:' defaultEdgeOutgoing=ScopeValue
7    (edgeConstraints+=EdgeConstraint)*
8  '}'
9  ;
10 EdgeConstraint:
11   class=EdgeClass 'in:' edgeIncoming=ConstraintValue 'out:' edgeOutgoing=
         ConstraintValue
12 ;
13 enum ConstraintValue:
14   None | Single | Transitive
15 ;
16 EdgeDefinitionByClasses: '{'
17   ('TransitiveIn' '(' transitiveInStereotypes+=EdgeTypeReference
18       ( ',' transitiveInStereotypes+=EdgeTypeReference)* ')')?
19   ('TransitiveOut' '(' transitiveOutStereotypes+=EdgeTypeReference
20       ( ',' transitiveOutStereotypes+=EdgeTypeReference)* ')')?
21   ('SingleIn' '(' singleInStereotypes+=EdgeTypeReference
22       ( ',' singleInStereotypes+=EdgeTypeReference)* ')')?
23   ('SingleOut' '(' singleOutStereotypes+=EdgeTypeReference
24       ( ',' singleOutStereotypes+=EdgeTypeReference)* ')')?
25   ('None' '(' noneSteorotypes+=EdgeTypeReference
26       ( ',' noneSteorotypes+=EdgeTypeReference)* ')')?
27 '}';
```

Listing 4.10: Grammar excerpt for the EdgeDefinition.

The second alternative, the EdgeDefinitionByClasses, enables the assignment of the constraints to concrete stereotypes or stereotype variables. For this case five different lists indicating the constraint type and the direction of the relation are defined (lines 17 - 26). By default, a stereotype is interpreted with no consideration for both directions.

4.3.2 Impact analysis

The impact analysis simulates the effects of certain events, like changes or failures, through propagating impact values along the relations of the EA model. Therewith the potential effects, direct and indirect ones, can be assessed. Thereby, three different types of impact

values are supported: `high`, `medium`, and `low`. Within respective rules the semantics for propagating those impact values through the model are defined.

Due to the lack of detailed information in enterprise architecture models, an accurate assessment of the impact is not possible. We decided against the use of probabilistic models because of the inherent uncertainty when defining the thresholds (although, if desired, the technique could be extended to compute probabilities for each effect). Instead, we approximate impacts through a best case/worst case analysis. Similar to the practices in software analysis, the worst case represents the maximal set of affected elements, whereas the best case conforms to the minimal set. The actual impact (which must be determined by a domain expert) typically lies somewhere in between. Additionally, customized impact analyses are supported, where the propagation semantics can be individually defined.

The propagation semantics are captured with impact rules that define the effects according the direction and type of the relation. For example, a propagation rule defines that a service has a `high` impact type, if the hosting application also has a *high* impact value. The impact status for each element is calculated with respect to the impact status of the related elements.

The impact analysis can be applied for change impact but also for availability analysis. In the first case the impact values are interpreted as the change status of the elements. In [dBBG+05] three different change types are proposed: *extend*, *modify* and *delete*. *Extension* refer to cases where new issues are added but the initial functionality remains the same. Consequently, an extension has no effects on depending elements and can be mapped to the impact value `low`. By contrast, a *modification* also affects the functionality or the structure and therefore it cannot be guaranteed that initially provided functionality will still be available or that the behavior remains unchanged. This change status is mapped to the impact value `medium`. Finally, *deletion* indicates that an element will be removed from the EA. This is represented with the `high` impact type.

Within availability analysis, the effects of the breakdown of an element are simulated. The `strong` impact type denotes a severe effect of the breakdown to this element. It cannot be ensured that an element with a `high` impact type is available any longer. Whereas, a `low` impact type denotes only a weak effect. This element will be still available, possibly with small restrictions. It also possible to determine the effects of other events through the definition of custom propagation semantics.

Arla provides three different types for the configuration of an impact analysis (see listing 4.11). The static mode (`StaticImpactConfiguration`) approximates the effects using predefined propagation rules in two cases. The worst case represents the maximal set of affected elements, whereas the best case represents the minimal set.

```
1 ImpactDefinition:
2 ImpactDefinitionByStereotype | ImpactDefinitionByEdgeClasses |
3 StaticImpactDefinition
4 ;
5 StaticImpactDefinition: {StaticImpactDefinition}
6 'static' mode = ('worst' | 'best') 'case'
7 ;
```

Listing 4.11: Grammar excerpt for the `ImpactDefinition`.

In the dynamic mode, the propagation semantics can be defined by the user. Therefore, either the relation types (`ImpactDefinitionByStereotype`) or the relation classes (`ImpactDefinitionByEdgeClasses`) are mapped to effect types. An effect type defines

propagation rules for each impact type. In Arla three different effect types are supported:

Table 4.3: Description of the effect types.

effect type	*description*
`weak_effect`	A high impact type causes a medium one, and a medium value a low one.
`strong_effect`	A high impact type causes a high one, and a medium type a medium one.
`no_effect`	No impact value is propagated.

A *low* impact type is propagated in no cases. The effect types have to be assigned to each direction of a relation, i.e. one effect type for incoming and one for outgoing relations. If, for example, an application component realizes a service, then the application component has a strong impact on the service, while the service only has a weak impact on the application component. Therefore, an outgoing realization would be assigned to the `strong_effect` and an incoming realization to the `weak_effect`.

The assignment of the effect types to relations is done similar to the assignment of the scope conditions in the section before. Listing 4.11 provides the grammar for the assignment to stereotypes. For each effect type and each relation direction the list of stereotypes which should be interpreted according to this type, has to be defined. If a stereotype is not assigned to any of these lists, it is interpreted according to the `no_effect` type.

```
1  ImpactDefinitionByStereotype : {ImpactDefinitionByStereotpe}
2    ('WeakEffect In' '(' incomingWeakStereotypes +=EdgeTypeReference
3        ( ',' incomingWeakStereotypes+=EdgeTypeReference)* ')')?
4    ('StrongEffect In' '(' incomingStrongStereotypes +=EdgeTypeReference
5        ( ',' incomingStrongStereotypes+=EdgeTypeReference)* ')')?
6    ('WeakEffect Out' '(' outgoingWeakStereotypes +=EdgeTypeReference
7        ( ',' outgoingWeakStereotypes+=EdgeTypeReference)* ')')?
8    ('StrongEffect Out' '(' outgoingStrongStereotypes +=EdgeTypeReference
9        ( ',' outgoingStrongStereotypes+=EdgeTypeReference)* ')')?
10 ;
```

Listing 4.12: Grammar excerpt for the `ImpactDefinitionByStereotype`.

Listing 4.13 provides the relevant part of the Arla grammar for an assignment of the effect types to the relation classes. Here as well, a default execution rule has to be provided for the generic type `ModelEdge` (lines 2 - 3). Typically, this is `no_effect` for the incoming as well as the outgoing direction. The default rule can be extended with individual propagation rules for relation classes (line 4 and line 8).

```
1  ImpactDefinitionByEdgeClasses:'{'
2    'ModelEdge'   'in:' defaultEdgeIncoming=ChangePropagation
3                  'out:' defaultEdgeOutgoing=ChangePropagation
4    (impactDefintion+=ImpactDefinition)*
5  '}'
6  ;

7  ImpactDefinition:
8    class=EdgeClass 'in:' edgeIncoming=EffectType 'out:' edgeOutgoing=EffectType
9  ;

10 enum EffectType:
11   no_effect | strong_effect | weak_effect
12 ;
```

Listing 4.13: Grammar excerpt for the `ImpactDefinitionByEdgeClasses`.

4.3.3 Path analysis

The path analysis assesses the direct and indirect dependencies between elements of the EA model. We define a path as a dependency chain from a concrete source element to a target element. A path contains of at least one relation and may contain several intermediate elements. This analysis class enables answering questions like '*What are the supporting application components for this business element?*'. In figure 4.5 all possible paths, with maximum length four, from the source element *Return* to elements with type *application components* are depicted.

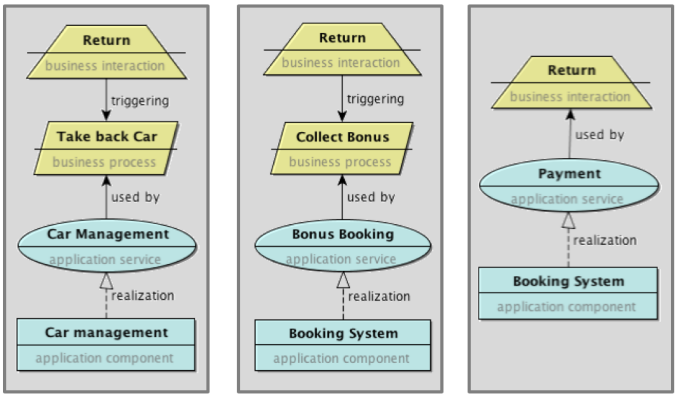


Figure 4.5: Paths from *Return* to *application components*.

The path analysis not only determines direct dependencies, also indirect ones are considered and possible intermediate elements must not be known during analysis definition. The respective analysis configuration used to determine the paths within the figure is provided in listing 4.14.

```
1 Template APathAnalysis{
2   "Determines all connected application for a selected element"
3   as Aggregate Pathset
4   defined with path definition: {
5     TargetStereotypes (nodeType:"application")
6     use AllPath
7   }
8 }
```

Listing 4.14: Example template for the configuration of a path analysis.

Result of a path analysis is a set of paths from the selected source elements to the specified target elements. The maximum length of the paths has to be provided when triggering the analysis execution. The source and target elements can be provided either within the analysis configuration or when triggering analysis execution. Listing 4.15, lines 3 and 4, provide the respective part of the Arla grammar.

Despite the element type restrictions, it is also possible to restrict the considered relation types. This can either be done with references to the stereotypes which should be considered (lines 10 - 12) or with referenced to the respective edge classes (lines 15 - 16). In the first case it is further possible to choose between the `AllPath` or the `ShortestPath` mode (line 12). Class restrictions are always determined within the `AllPath` mode.

The `AllPath` mode determines all possible dependencies chains between the source and target element. The `ShortestPath` mode returns only the shortest path between a source and a target element.

```
1  PathDefinition: {PathDefinition}
2    'defined with path definition:' '{'
3      ('SourceStereotypes' '('source+=NodeTypeReference (',' source+=
           NodeTypeReference)*')')?
4      ('TargetStereotypes' '('target+=NodeTypeReference (',' target+=
           NodeTypeReference)*')')?
5      edgeCondition = PathEdgeCondition
6    '}'
7  ;
8  PathEdgeCondition: PathByStereotypes | PathByClasses ;
9  PathByStereotypes:
10   ('Incoming' '(' in+=EdgeClassReference (',' in+=EdgeClassReference)*')')?
11   ('Outgoing' '('out+=EdgeClassReference (','out+=EdgeClassReference)*')')?
12   'use' mode= ('ShortestPath' | 'AllPath')
13 ;
14 PathByClasses: {PathByClasses}
15   ('Incoming' '(' in+=EdgeTypeReference (',' in+=EdgeTypeReference)*')')?
16   ('Outgoing' '('out+=EdgeTypeReference (','out+=EdgeTypeReference)*')')?
17 ;
```

Listing 4.15: Grammar excerpt for the definition of a path configuration.

Selecting the option `ShortestPath` within the example in figure 4.5, determines only the first and the third path. The second path has the same target element as the third one, but since it is longer it will be discarded during analysis execution.

Replacing the `use AllPath` statement with the following class condition, enables the determination of realization paths:

```
1  Incoming (edgeClass:Provide, edgeClass:ConsumedBy)
```

The condition states that only incoming relations assigned to the classes `Provide` or `ConsumedBy` should be considered. Thus, only the third path will be returned as result. The two other paths are ignored in this case, since they contain a *triggering* relation.

The source and target elements for the path analysis are either selected at run time, through the selection of concrete elements within the EA model, or they are specified within analysis definition. The elements of the provided stereotypes extend the respective user selection of source and target elements. In the example above, no source stereotype is provided, i.e. only the selected model elements are used as starting point for the paths. The target elements of the paths should be of the type `application`. This means that only those paths are within the result whose target element has the type `application`.

The result of the path analysis is often used to create a dependency matrix or a support map. These are specific visualization types to focus on the dependencies between EA model elements. Exemplary, a support map is shown in figure 4.6.

	Reservation by Phone	Car Pick up	Car Return	Payment
User Management	User Management			
CRM	Customer Management			
Car management	Car Management Reservation Management	Car Management		Bonus Booking
Booking System				Payment Bonus Booking
Reservation System	Reservation Management			

Figure 4.6: Example support map.

The support map provides a visualization of the dependencies between application components (rows), application services (center) and business services (columns). The application services are located within the support map according to the existence of a path to the respective application component and business services. In contrast a dependency matrix visualizes available paths between two types of elements and thus, a cell is filled with 'x' if there exists a path from the element of the row to the element of the column.

4.3.4 Metrics

The calculation of metrics is important for the evaluation of an EA and several existing analysis approaches utilize a quantification within their procedure. Metrics can be used to quantify goals and determine their achievement as well as to quantify benefits, risks and costs. In listing 4.16 an example metric using an Arla Template is provided. This metric calculates the average costs for all applications. Therefore the `costs`, provided at elements with type `application`, are summed up (line 5) and divided through the total number of elements with type `application` (line 6).

```
1 Template AMetricTemplate{
2   "Calculate the average costs for an application"
3   as Aggregate Metric
4   defined with calculation rule averageCosts:
5   (SUM (nodeType:"application".propertyType:"cost")) /
6   (COUNT(nodeType:"application"));
7 }
```

Listing 4.16: Example metric template to calculate the average application costs.

Within Arla, metrics are calculated as an aggregated value for the whole architecture (lines 5 - 8 in listing 4.17). But it is also possible to calculate metrics for a specific set of elements (lines 1 - 4). Thereby, the analysis definition is extended with a set of stereotypes or element classes identifying the relevant elements (line 3).

```
1 ElementMetricDefinition:
2   'defined with calculation rule:' calculationRule = CalcExpression ';'
3   'for types' '('(nodeTypes+=NodeReference (',' nodeTypes+=NodeReference)*)?')'
4 ;
5 AggregateMetricDefinition:
6   'defined with calculation rule' (name=ID)? ':'
7       calculationRule=CalcExpression ';'
8 ;
```

Listing 4.17: Grammar excerpts for metric definition.

The `CalcExpression` enables the specification of the calculation rule. The respective grammar is presented in listing 4.18.

```
1 CalcExpression:
2   Addition | Sum | Mult | NodeCount | EdgeCount
3 ;
4 Addition returns CalcExpression:
5   Multiplication (({Plus.left=current} '+' |
6   {Minus.left=current} '-') right=Multiplication)*
7 ;
8 Multiplication returns CalcExpression:
9   PrimaryExpression (({Multi.left=current} '*' |
10  {Div.left=current} '/') right=PrimaryExpression)*
11 ;
```

```
12 Sum:
13   'SUM' value=CalcExpression ('of connected' nodeSet = NodeSetDefinition)?
14 ;
15 Mult:
16   'MULT' value=CalcExpression ('of connected' nodeSet = NodeSetDefinition)?
17 ;
18 NodeCount :
19   'COUNT' '(' value=NodeSetDefinition ')'
20 ;
21 EdgeCount:
22   'COUNT' '(' value=EdgeCondition ('for' nodeType=NodeReference)? ')'
23 ;
24 EdgeCondition:
25   {UndirectedEdgeCount} 'connected' (edgetype = EdgeReference 'to')? value =
         NodeSetCondition |
26   {OutgoingEdgeCount} 'outgoing' value = EdgeReference |
27   {IncomingEdgeCount} 'incoming' value = EdgeReference
28 ;
29 PrimaryExpression returns CalcExpression:
30   '(' CalcExpression ')' |
31   {AggregatedMetricReference} value=AggregatedMetricReference |
32   {TypedPropertyReference} value=TypedPropertyReference |
33   {Number} value = Number
34 ;
35 AggregatedMetricReference:
36   metric=[AggregateMetricDefinition]
37 ;
38 Number :
39   INT | Float
40 ;
41 Float returns ecore::EFloat:
42   INT'.'INT
43 ;
```

Listing 4.18: Grammar excerpt for the definition of a calculation rule.

A `CalcExpression` supports the common mathematical arithmetic operations for addition, subtraction, multiplication and division (lines 4 - 10). For aggregating values Arla provides the following four operations:

`Sum`	to summarize all values defined by a calculation rule (lines 12 - 13)
`Mult`	to multiply all values defined by a calculation rule (lines 15 - 16)
`NodeCount`	the number of elements according to `NodeSetDefinition` (lines 18 - 19)
`EdgeCount`	the number edges according to a `EdgeCondition` (lines 21 - 22)

Within a `Sum` or `Mult` statement, the considered elements for value aggregation can be further restricted with a `NodeSetDefinition`. If such a definition is provided, only elements within the node set that have a relation to the current context element are considered. Within the `NodeCount` the number of elements within the referenced node set is determined. The `EdgeCondition` enables the specification of constraints over relations. There are three different types of edge conditions provided within Arla. The first one counts all relations to an element provided within the node set condition, independently from the direction (line 25). Optionally, a restriction to a certain edge type or edge class can be made. The other two ones are directed conditions (lines 26 - 27). They provide the number of incoming, respectively outgoing, relations of a specific type.

Finally, the atomic values, i.e. `PrimaryExpression`, of a calculation rule are provided in lines 29 - 43. The `TypedPropertyReference` depicts the values of the provided property type. The `AggregatedMetricReference` references the result of another aggregated metric and a `Number` is used to include a static numeric value of type `int` or `float`.

4.3.5 Performance analysis

A special element metric is the performance analysis proposed in [JI09]. The authors propose the calculation of four different performance indicators: workload, processing time, response time and utilization. Based on a top-down propagation, first the workload of certain architecture elements is determined. This is followed by a bottom up propagation of the performance measures. Thereby, the response time is approximated based on the utilization of the resources and the processing time of requested elements.

The response time provides the time span from initiating a request until receiving an answer. This measurement is relevant for the user or customer perspective. From a service or product point of view, the processing time, i.e. the time for actual handling of a request, is important. Finally, for the resource view the utilization is a relevant measurement. It provides the percentage of the operational time, where this resource is working. If the utility is too high, this may be an indicator for bottlenecks.

For each of the four measures the authors propose formulas for their calculation. The calculation relies on the following set of properties:

- `weight` for any relation
- `service time` for behavior elements
- `capacity` for any resource, e.g. actor, application component, device and node
- `arrival frequency` for business level elements

The weight can be used to depict an average number of accesses or usages for a relation. The default value is one, i.e. one incoming request triggers one outgoing call. The service time of a behavior elements provides the internal time for the realization of the provided functionality. The capacity provides the maximum number of requests that can be handled by this resource.

Figure 4.7 provides the result of the performance analysis within the running example. For visibility reasons, only an excerpt of the model is shown. Within the attribute section of the elements, the retrieved results for the performance measures are provided. The properties of the EA model like arrival frequency and service time are visualized in extra boxes.

The workload is calculated for every model element. Initial value is the *arrival frequency* of the top-level business service. The value is propagated to the process and further to the application layer. At each element the workload is defined as the sum of all workloads from incoming relations. Since the *Car Management* service is also used from other business processes, the workload value is higher than the provided 120. The workload is further propagated to the infrastructure layer until reaching the devices and the *Mainframe* node. These elements have a *capacity* attribute which is used to determine their utilization. The utilization is used to approximate the response time for elements having an *service time* attribute. For the approximation a queuing model is used. Finally, the results of the response time and processing time can be propagated bottom up until the business layer.

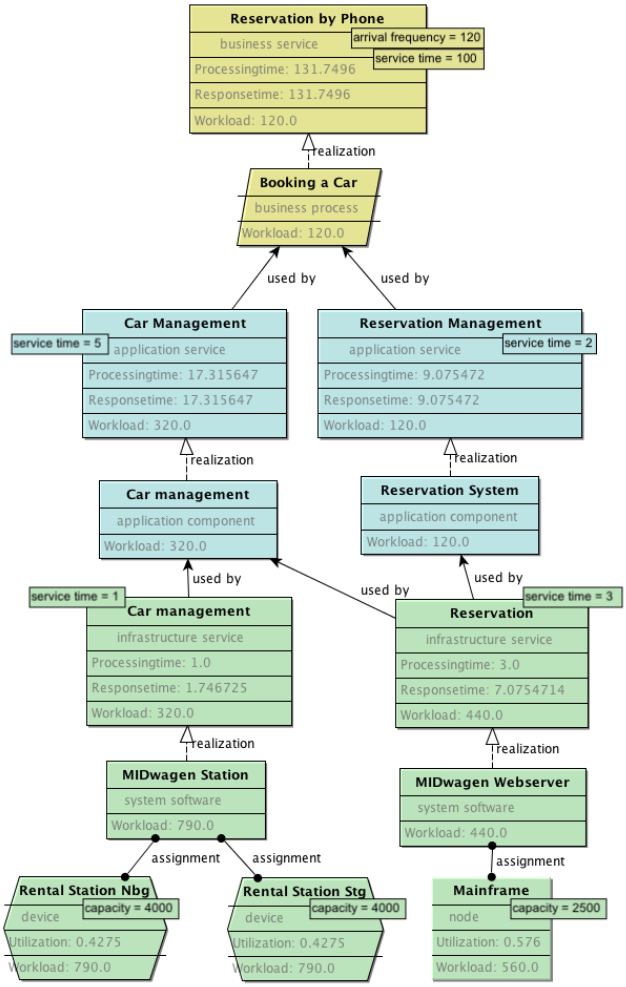


Figure 4.7: Result for the performance analysis.

The recursive definition of the performance measures cannot be captured with the previously presented metric. Hence, a new analysis class for performance analysis is provided within Arla. The respective grammar excerpt is presented in listing 4.19.

```
1  PerformanceDefinition:
2    'defined with performance calculation' '{'
3      'target result:' target=PerformanceResult
4      'map' 'frequency' 'to' frequencyProperty=PropertyReference
5      'map' 'weight' 'to' weightProperty=PropertyReference
6      'map' 'capacity' 'to' capacityProperty=PropertyReference
7      'map' 'serviceTime' 'to' serviceTimeProperty=PropertyReference
8    '}'
9  ;

10 enum PerformanceResult:
11   workload | responsetime | processingtime | utilization
12 ;
```

Listing 4.19: Grammar excerpt for the definition of a performance configuration.

To enable the execution of this analysis the utilized properties in calculation rules of [JI09] have to be mapped to actual property stereotypes of the EA model (lines 4 - 7). In

contrast to [JI09], we identify the elements for the calculation of the measures not with their concrete stereotype. They are identified by having the respective property, i.e. the utility is calculated for all elements having an *capacity* attribute. The relations referenced within the calculation rules of [JI09] are replaced with the edge classes. Thus, no further mapping to concrete stereotypes is required. Since Arla supports only the calculation of one result per element, the targeting attribute for the performance analysis has to be specified (line 3 and lines 11, 12).

4.3.6 Gap analysis

The purpose of the gap analysis in the EA context is the determination of differences, respectively gaps, between two different architectural models [The18, DB14]. Thereby both models are described using the same schema and elements in the two models can be identified using names or IDs. Its goal is to identify newly added elements and elements that no longer exist. The gap analysis is mainly used to compare the current architecture with a desired target architecture or a planning scenario. A planning scenario describes also a desired target architecture, but is does not comprise the whole architecture. The gap analysis can also be used to compare different target alternatives with each other. Figure 4.8 provides an example planning scenario for the CarRental company. Through the implementation of a *Return Machine* and the respective application components, a 24h car return is realized.

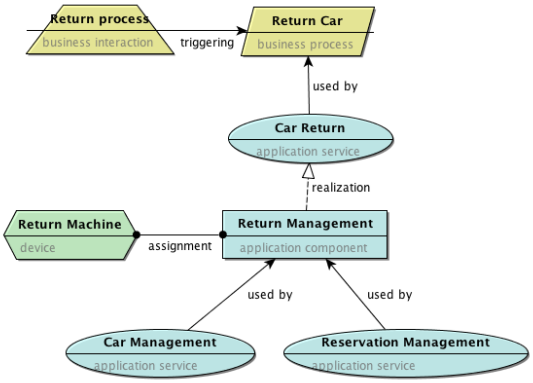


Figure 4.8: Planning scenario 24h car return for the CarRental example.

To express the differences between the models, the gap analysis retrieves a status for each element. `Unaffected` elements occur only in the current model, whereas `new` elements occur only in the target model. Elements identified within both architectures are assigned with the status `affected`. Elements with the assigned attribute *unaffected* are potential deletion candidates, since it cannot be concluded automatically whether the absence in the scenario was with purpose or not. Heuristics can be used to provide suggestions for deleted elements. We further refine the set of *deletion candidates* to those that have at least one relationship to an element affected by the planning scenario. Finally, it is the task of the architect to decide about the planning status of the element. In figure 4.9 the result of a gap analysis between the current architecture of the car rental company and the proposed planning scenario for the 24h car return is provided.

The grammar excerpt for the respective analysis configuration in Arla is presented in

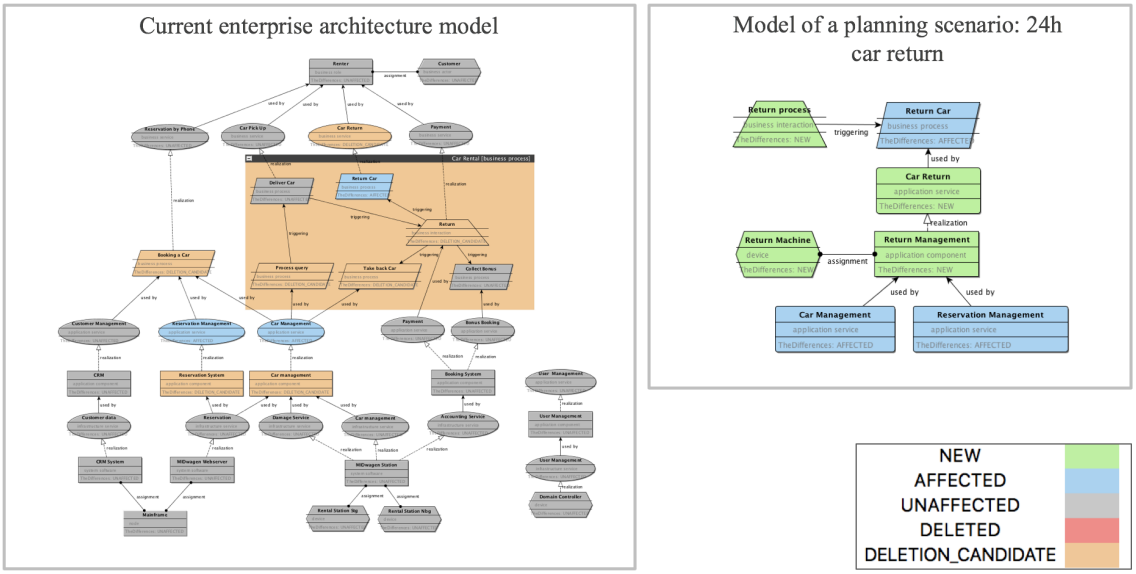


Figure 4.9: Result for a gap analysis using the `Differences` option.

listing 4.20. For performing a gap analysis, the current and the target model have to be provided. If no current model is given, the actual EA model is used. The gap analysis can be performed with two different options (line 5 and lines 7 - 8). With the `Difference` option, an attribute is calculated for each element according to the description above. An example result with this option is presented in figure 4.9.

```
1  GapDefinition:
2    'defined with gap configuration:'
3    ('current model' base=STRING)?
4    'target model' target=STRING
5    'calculate' option = GapOption
6  ;
7  enum GapOption:
8    SuccessorCandidates | Differences
9  ;
```

Listing 4.20: Grammar excerpt for the definition of a gap analysis.

The `SuccessorCandidate` option determines a set of potential predecessors for each `new` element within the target model. The respective result for the example is depicted in figure 4.10. Only for the *Return machine* no predecessors are provided, since there exists no element in the current architecture with the same stereotype *device*.

Affected (current)	Successor (target)
Return Car	Return Car
Reservation Management	Reservation Management
Car Management	Car Management

Figure 4.10: Result for a gap analysis using the `SuccessorCandidates` option.

4.3.7 Adapted analysis

Re-use of analysis definitions is provided with the analysis class `AdaptedAnalysis`. This class is used to execute predefined templates on a specific EA model and is therefore only provided in `Arla Specific`. Listing 4.21 illustrates the usage of an adapted analysis with a customization of the template *ANodeSetExample* (provided earlier in listing 4.8).

```
1 Analysis AnAdaptedAnalysis {
2   "Description"
3   as Aggregate Modelelementset
4   adapt MyAnalysisPackage.ANodeSetExample {
5     map "business process" to node:"business process" ["7f2e37ca-2baf"]
6     map "business unit"    to node: "business role" ["4f48110f-ffb4"]
7     map "criticality"      to property:"mission criticality" ["e1608cdb-4e0e"]
8   }
9 }
```

Listing 4.21: Example analysis definition for an adapted analysis.

Each stereotype reference used within the template has to be mapped to a concrete stereotype of the EA model. In this case, these are the two node type references `business process` (line 5) and `business unit` (line 6) as well as the property reference `criticality` (line 7). The grammar excerpt for an `AdapatedAnalysis` is presented in listing 4.22.

```
1  AdaptedAnalysis:
2    analysisStyle=AnalysisStyle
3    resultType=ResultType
4    'adapt' analysis= [genericArla::Template|QualifiedName] '{'
5      (mappings += Mapping)*
6    '}'
7  ;
8  Mapping:
9    'map' reference = [genericArla::StereotypeReference|STRING]
10   'to' typeName = StereotypeReference
11 ;
```

Listing 4.22: Grammar excerpt for the configuration of an adapted analysis.

As for all analyses, the analysis type (i.e. if it provides an aggregated result or results for each element) and the result type must be specified. This is followed by the qualified name of the template that should be adapted. The qualified name consists of the name of the `TemplatePackage` and the name specified within the `AnalysisHeader` of the template. Finally, the mappings between the stereotype references used within the template and the concrete stereotype references of the EA model are defined.

4.3.8 Custom analysis

If an analysis cannot be captured with one of the proposed classes above, Arla provides an interface for the execution of custom analyses. These custom analysis configurations directly access the underlying execution environment. Two types of custom analyses are supported within Arla. The `SpecificDFAConfiguration`, respectively `DFAConfigurationTemplate`, and the `CustomQuery`. The `SpecificDFAConfiguration` and the `DFAConfigurationTemplate` use a `DFADefinition` (listing 4.23) for analysis configuration.

```
1 DFADefinition:
2   'defined with DFA configuration:'
3     'Configuration path' configuration=STRING
```

```
4       'Strategy' strategy=STRING
5   ;
```

Listing 4.23: Grammar excerpt for the definition of a DFA configuration.

The `DFADefinition` provides the `path` to a DFA analysis configuration as well as the `strategy` that determines the triggered analysis from the configuration.

Additionally, the `CustomQuery` (listing 4.24) enables the specification of a SPARQL query to analyze the EA model according to individual needs. This analysis can only be specified as a specific analysis definition.

```
1  CustomQuery:
2    analysisStyle=AnalysisStyle
3    resultType=ResultType
4    'defined with SPARQL Query' configuration=STRING
5  ;
```

Listing 4.24: Grammar excerpt for the definition of a custom query.

In order to use these two analysis classes, the user has to be familiar with the respective technologies. There is no further support within Arla to create a respective DFA configuration or a SPARQL query. Nevertheless, they are important analysis classes to address individual needs that are not covered by the previous analysis classes.

4.3.9 Composed analysis

To enable the definition of more complex analyses, Arla supports the composition of analysis definitions and templates. This enables the combination of different analysis results or to execute an analysis based on the result of another one. Additionally, analysis composition is an important mean to deal with incomplete models. Before executing a specific analysis, the model can be restricted to the relevant part of the architecture that is sufficiently described. For example, if performance parameters are not available for all servers, those servers can be excluded.

Arla provides four different possibilities for the `CompositionDefinition` which are shown in listing 4.25. The `ApplyRule` and the `ApplyEachRule` define a successive execution of the two referenced analysis definitions. The `CombineRule` and the `SpecificCombineRule` merge the results of two independently executed definitions. An analysis is referenced using the `name` provided within the `AnalysisHeader`.

```
1  CompositionDefinition:
2  ApplyRule | ApplyEachRule | CombineRule | SpecificCombineRule
3  ;

4  ApplyRule:
5  'apply' analysis2=[AnalysisHeader|ID] 'on' analysis1=[AnalysisHeader|ID]
6  ;

7  ApplyEachRule:
8  'apply' analysis2=[AnalysisHeader|ID] 'onEach' analysis1=[AnalysisHeader|ID]
9  ;

10 CombineRule:
11 'combine' analysis1=[AnalysisHeader|ID] 'and' analysis2=[AnalysisHeader|ID]
12 ;

13 SpecificCombineRule:
14 'combine' analysis1=[AnalysisHeader|ID] 'and' analysis2=[AnalysisHeader|ID]
       'with' 'operation'
15 rule=SetOperation
```

```
16  ;
17  enum SetOperation:
18  INTERSECTION | UNION | DIFF
19  ;
```

Listing 4.25: Grammar excerpt for the definition of an analysis composition.

The procedure for composing two analysis definitions using the `ApplyRule` and the `ApplyEachRule` is illustrated in figure 4.11. The result R1 of the first analysis (A1) is thereby used within the second analysis (A2). The final result R is the result of the second analysis. Whereas in the `ApplyRule` the result itself is used within the second analysis, in the `ApplyEachRule` the result R1 is used as input parameter for the second analysis. In this case R1 must be interpretable as a set of elements. These elements are the *selected* elements used for the execution of (A2).

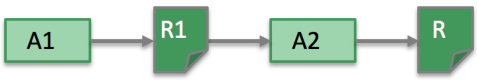


Figure 4.11: Successive analysis execution.

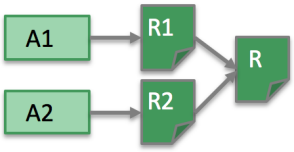


Figure 4.12: Analysis result merging.

Listing 4.26 provides an example template for an analysis composition using the `ApplyEachRule`. Thereby, the `ANodeSetExample` determines all business processes with a high criticality (see listing 4.8). The result of this analysis is then used within metric calculation. The referenced metric `AElementMetricTemplate` is calculated for all elements within the node set result.

```
1  Template AAnalysisCompositionWithApplyEachRule {
2    "Calculate a metric for business critical processes"
3    as Element Metric
4    defined with composition rule:
5    apply AElementMetricTemplate onEach ANodeSetExample
6  }
```

Listing 4.26: Example template for an analysis composition using an `ApplyEachRule`.

The `CombineRule` and the `SpecificCombineRule` combine the results of the referenced analysis definitions. This process is illustrated in figure 4.12. First the analyses are both executed, retrieving result R1 from the first analysis and R2 from the second one. Afterwards both results are merged with each other in order to create the final result R. When using the `CombineRule`, R1 and R2 are simply merged with each other. That means, model element sets and path sets are combined into one model element set respectively path set. Calculated attributes are merged into one result list.

The `SpecificCombineRule` can only be applied to results referencing elements, like a model element set or a path. The two sets of elements provided by R1 and R2 are then merged with each other according to the specified `SetOperation` (line 15 and lines 17 - 18). An `INTERSECTION` returns only those elements that appear in both results. The `UNION` operator returns all elements of both results and the `DIFF` operator returns the elements of R1 without the elements of R2.

4.4 Conclusion and Related Work

In this section we presented a architecture analysis language (Arla) for the definition of EA analyses. The language Arla provides a universal interface to EA analyses. Thereby, it abstracts from the technical details and provides a declarative way of specifying and customizing EA analyses. Arla enables the specification of different analysis classes:

- The *Scope Analysis* enables the specification of constraints to generate different views of the architecture model.
- Within the *Impact Analysis* propagation rules can be customized in order to estimate the effects of an event.
- Using *Metrics* it is possible to quantify quality attributes of the architecture or to evaluate the goal fulfillment. Arla supports the typical mathematical operations as well as aggregating mechanisms like the addition of property values from a set of elements.
- The *Performance Analysis* facilitates the generic application and customization of the analysis approach proposed in [JI09]. The analysis calculates different performance indicators for the EA elements through top-down and bottom-up propagation of workloads and execution times.
- Dependencies within an EA are analyzed using the *Path Analysis* class. The analysis determines dependency chains between a source and a target element. Additionally, specific constraints can be defined that must be met by the resulting path.
- Finally, the *Gap Analysis* enables the comparison of two different EA models.

Additionally, Arla supports the *composition* of these classes to support more complex analyses. If an analysis cannot be captured with these classes, Arla provides an interface for the *custom definition* of SPARQL queries or the execution of custom DFA analysis configurations.

To address the heterogeneity of meta models within the EA domain and to support reuse of analyses, Arla provides a *Template* mechanism. Despite the definition of specific analysis definitions for a concrete EA model, templates can be defined independently from a meta model. Thereby, the proposed relation and element classes in section 3.2 can be used. Additionally, stereotype variables can be utilized for further references that are not captured by these classes. Within the *Adapted Analysis* the customization of templates for a concrete EA is made. Thereby, only the variables have to be mapped to concrete stereotypes.

Arla overcomes current weaknesses like the isolation of existing approaches. Existing approaches have merely a limited scope and address only one goal. We could not identify a language for the specification of EA analyses that covers different analysis types. Since a unified approach for EA analyses is missing, analysis composition is also sparely considered in current research.

In [BMS09] Buckl et al. provide a categorization of seven analysis approaches according to five dimensions. These are the body of the analysis, i.e. if the analysis addresses the structure, behavior statistics or the dynamic behavior, the time reference (ex-post or ex-ante), the analysis technique (expert-based, rule-based or indicator-based), the analysis concern (functional or non-functional) and the self-referantiality. They identify the necessity within further work to establish approaches that cover multiple characteristics, for example through a combination of existing ones.

Arla covers 62% of the identified EA analysis types in [Rau13]. Another 20% can be captured with an individual definition of a SPARQL query or a custom DFA analysis configuration. Only 18% of the types cannot be captured at all (see section 8.2). Regarding the coverage of the technical dimensions, Arla is especially weak regarding probabilistic techniques like Bayesian networks, PRMs or EIDs. The majority of the approaches cannot be captured with the language, some of them can partially be realized with Arla. All other technical dimensions are covered or at least partially covered with Arla. The coverage of the functional dimensions with Arla is very high. Only the dimension *System* is not covered. All approaches assigned to this dimension are also assigned to one of the probabilistic dimensions.

[JUB+13] and [NSV14] provide frameworks for EA analysis dealing with different types of analyses. [JUB+13] propose probabilistic OCL to describe properties of an architecture and to determine them. [NSV14] focuses on the visualization of EA models. Through the definition of a composition chain an individual view can be created using selectors and decorators.

As discussed in [LSB14a], another related research area are general purpose querying languages like SQL [ISO11a] for databases and SPARQL [W3C13] for ontologies (proposed for EA in [SKR13b]). Both have the weakness that they are highly dependent on the underlying structure (database schema, respective t-box for ontologies). Only the abstraction layer introduced with Arla and the GMM allows a metamodel independent specification of analyses. Also the constraint language OCL can be used for EA model analysis. It allows the annotation of constraints at meta model elements and their evaluation for models. Here as well, the OCL statements rely on the meta model. Changes in the meta model lead to extensive adjustments to the statements. Analyses like the impact analysis or the performance analysis, where the result depends on the value at the neighboring nodes, are difficult to implement with OCL [Saa14]. SPARQL, without further reasoning support, and SQL do not support this kind of analyses either. All three approaches lack the possibility to reuse given definitions by referencing them.

[HRSM13] evaluated 13 different EAM tools according to their metric support. About half of the tools provide a user interface for metric definition, and another half of the tools are shipped with predefined metrics. Only three tools (ABACUS, EA solutions and iteraplan) provide a domain specific language for the definition of metrics. Despite metrics, current tools support analysis definition either through providing predefined ones or through referencing to SQL or a similar interface. Customization possibilities are often limited to the way the data is visualized. Especially complex analyses have to be integrated by the vendor. More than 60% of the participants of a survey regarding EAM tools have the need for a user friendly analysis frontend which is specific for roles or users [HHKR16].

5
Architecture Analysis Framework

In the previous chapter we proposed the analysis definition language Arla for specification of different analyses. The supported analysis classes are scope analysis, impact analysis, metric calculation, performance analysis, path analysis and gap analysis. Additionally, there is the need for the individual customization of these analyses according to the current stakeholder needs. The proposed language enables the custom specification of analysis definitions as well as to define generic analysis templates. These templates are independent from a specific EA model and support the reuse of analyses definitions.

The goal of this chapter is the provisioning of an execution framework that is able to interpret these analysis and template definitions and determines the results.

Parts of the presented results were previously published in [LSB14c, LSB14b, LB17]. The details about impact analysis execution were previously published in [LSB14b, LSB14a]. The execution procedure for the gap analysis was previously published in [LB18a].

5.1 Design Goals

EA analysis activities are characterized by the high dynamic regarding the underlying meta model structures as well as regarding the demands from the stakeholders. To cope with the challenges during EA analysis activities the *Architecture Analysis Framework A2F* should consider the following design goals.

Generic applicability For representing the EA model, a large variety of different meta models and tools is used. There exists no standard to rely on. Nevertheless, due to the large and complex models tool support is required to process and evaluate them. It is important that such an analysis support makes use of the existing EA models. To ensure a generic applicability, the framework must be able to process and evaluate EA models independently from the utilized meta model and tool.

Universal interface Different analysis types are proposed within literature to process and evaluate EA models. An integrated approach, enabling the execution of different analysis types is missing. The A2F should provide the stakeholders with a universal interface to the EA analysis activities. Through supporting different analysis types, a large variety of the goals during analysis activities can be covered. Such an integrated approach must also allow the composition of analyses to enable a more in-depth consideration of the architecture.

Custom analyses Not only the meta models but also the requirements for analysis activities vary. Depending on the stakeholder and the EA maturity, different information demands arise. Providing only predefined views and evaluation mechanisms does not cope with the existing dynamic. Stakeholders want to be able to specify custom analyses, respectively adapt predefined ones in order to fulfill their needs.

Declarative analysis definition The framework should enable a declarative specification of the supported analyses. The stakeholders do not want to deal with the technical details and challenges. Additionally, defining an analysis should not require being an expert of the data model which is used for persistence. Therefore, analysis configuration deals with the specification of parameters and constraints declaring the 'What?'. The concrete execution routine is generated automatically. Thus, the framework abstracts from the technical details of analysis execution and provides a more abstract interface for analysis specification to the stakeholders.

Re-use of analysis definitions Despite the individual needs for analysis activities, there are also recurring questions. Examples are the views proposed within ArchiMate [The17] which address typical information needs from different stakeholders. The predefined templates enable a fast and easy execution of the respective analysis. With ongoing maturity of the EAM, these predefined templates will be further adapted to serve the individual information needs. The short feedback cycles increase the acceptance of an EAM initiative within the organization, especially at the beginning. A poor acceptance is a common problem during EAM introduction.

Robustness regarding large models EA models are very large and complex models. The analysis framework must be able to deal with this issue. This means that stakeholders performing an analysis do not want to have overnight-evaluations. The decision-making processes within EAM are interactive ones. The stakeholder performs an analysis, and based on the result, new demands arise which trigger new analyses. This interactive nature requires immediate feedback from the analysis framework.

Robustness towards incomplete models EA models describe the business of an organization, the dependencies to the applications as well as their IT infrastructure. Although, it is the goal to provide a holistic model of the enterprise, the created models typically are not complete and often not up to date. The analysis framework must be able to deal with such incomplete models. It should provide mechanisms that are robust regarding missing information as well as provide means to overcome them.

Round-trip-engineering Finally, the analysis framework should not be a dead end for the data. It must be possible to return the analysis results back to the model or provide them to other demanders. Although a generic application and representation of the EA data is required, it is important that none of the specific information is dropped. This is important to enable the integration within current tooling in the EAM context. The available data can be extracted for analysis and, if new information is generated, it can be returned back into the modeling tool.

5.2 Overview

The Architecture Analysis Framework A2F consists of three main components, considering the aspects analysis definition, analysis execution and model storage. Figure 5.1 provides an overview of the structure.

Analysis definition is done with the proposed DSL Arla (see chapter 4). Arla enables the declarative definition of nine different analysis classes. Thereby, it supports their configuration for a specific EA model as well as the configuration of generic applicable templates. Templates enable a meta model independent definition of EA analyses. The templates, respectively specific analyses, can be configured according to the needs of the stakeholders. Thus, enabling a customized analysis execution.

The *analysis execution* component contains the logic to convert Arla into evaluable constructs, i.e. it deals with the '*How*' of analysis execution. The interpreter parses the analysis definitions and generates the relevant artifacts for execution. Structural requests are transformed into SPARQL [W3C13] queries. Behavioral requests and recursive definitions are evaluated using DFA [SB13]. The interpreter generates the required configuration for the DFA execution. The result processing component provides a uniform schema for the final result. It also converts the specific result types of SPARQL queries and the DFA into the common schema.

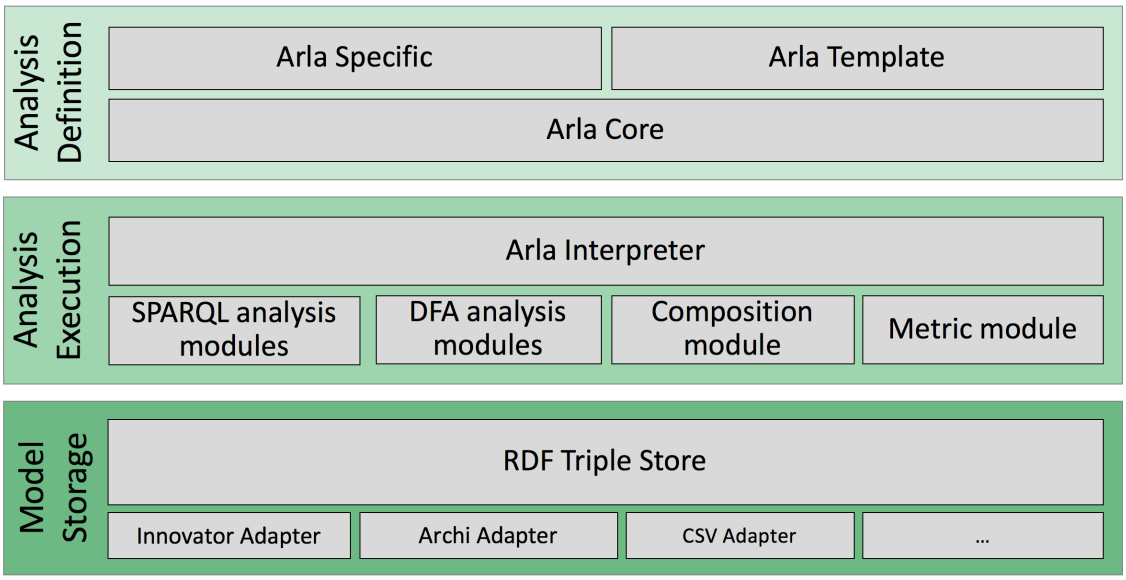


Figure 5.1: A2F Overview.

The *model storage* is realized with an RDF Triple Store. Thereby, the generic meta model GMM, presented in chapter 3 is utilized. For data import, several adapters are provided to convert the EA data into the RDF format. These are an adapter for the modeling tool Innovator [MID19], an adapter for CSV files and an adapter for the open source modeling tool Archi [Bea19]. The respective meta model is created dynamically during import.

The model storage, the analysis definition and the analysis execution are described in detail in the following sections.

5.3 Model Storage

The model storage component within the Architecture Analysis Framework (A2F) deals with the persistence of the EA data as well as their access for analysis purposes. Figure 5.2 provides a conceptual overview of the required sub components as well as the dependencies between them. The flow from the import files at the bottom to the return values for analysis execution at the top illustrates the storage approach.

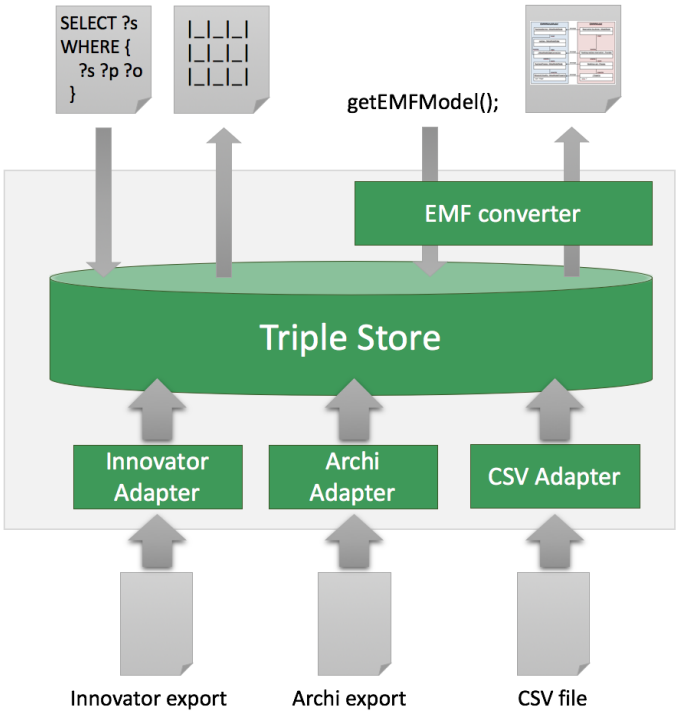


Figure 5.2: Conceptual overview of the model storage component within A2F.

The export files provided by EA tools, like Innovator or Archi, but also generic CSV files are processed by respective *Adapters*. Those adapters parse the files and load the contained EA data into the triple store. During analysis execution the triple store is accessed to provide the relevant data. SPARQL queries can directly be answered by the triple store. For executing analyses based on the DFA technique, the EA data must be transformed into an EMF model. The *EMF converter* translates the RDF triples within the triple store into an EMF model and provides the generated model for DFA execution.

5.3.1 Data representation within the triple store

The captured EA model data is stored within the triple store utilizing the proposed generic meta model in section 3. Semantic web technologies are already applied within the EAM domain to store and analyze the data within in research [CHL+13, OLB15, SKR13a, SKR13b] and practice [Top, Sof19a, Car16]. The property-centric data representation within RDFS provides a flexible and extensible storage technique. The principle of URIs for identification as well as the inferencing mechanisms support the integration of heterogeneous data, a common issue when creating EA models. We decided for RDFS and not OWL for data representation, since the expressiveness is sufficient to represent the EA

model and the query and inferencing engines for solely RDFS data are more performant.

GMM vocabulary

To enable the application of semantic web technologies the proposed meta model GMM has to be transformed into an RDF graph. Due to the parallelism between RDFS and object-oriented modeling this can be done straightforwardly. Each class of the GMM is translated to an `rdf:Class`. The class hierarchy can be captured using `rdfs:subClassOf` statements. Each relation between the classes is translated into a `rdf:Property`. The source and target element of a relationship are depicted with the `rdfs:domain` and `rdfs:range` statement. Class properties like name or UUID are also represented with an `rdf:Property`. In this case the range of the property is not another class but an `rdf:Literal`, representing the type of the property value. The resulting RDF graph is illustrated in listing 5.1 and the tables 5.1 and 5.2.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix gmm:   <http://www.smds.de/autoanalyze/eam/gmm/gmm#> .
4  @prefix meta: <http://www.smds.de/autoanalyze/eam/gmm/gmmmetamodel#> .
5  @prefix model: <http://www.smds.de/autoanalyze/eam/gmm/gmmmodel#>.

6  meta:MetaModelNode
7    a                rdf:Class ;
8    rdfs:subClassOf  meta:MetaModelStereotype ;
9    rdfs:subClassOf  meta:NamedElement .

10 meta:properties
11   a                rdf:Property ;
12   rdfs:domain      meta:MetaModelNode ;
13   rdfs:domain      meta:MetaModelEdge ;
14   rdfs:range       meta:MetaModelProperty .
```

Listing 5.1: RDF representation of the GMM.

Listing 5.1 shows the representation of the class `MetaModelNode` and its `properties` relation within the RDF graph using the turtle syntax. Lines 1 - 5 declare the prefixes used within the graph. The different packages used within the GMM meta model in section 3.2 are represented as different namespaces within the RDF graph. Following there is one namespace `gmm` for the generic elements like `NamedElement`. Additionally, there is one namespace `meta` for the concepts describing the meta model part and one namespace `model` for elements describing the model part.

The class `MetaModelNode` is described in lines 6 - 9. Since it inherits from the classes `MetaModelStereotype` and `NamedElement` the respective subclass statements are added. Properties and relations of this class are declared in `Property` statements due to the property-centric approach. For example, in lines 10 - 14 the relation `properties` is defined, having `MetaModelNode` as domain, and `MetaModelProperty` as range. Since `MetaModelEdges` can also have properties, the respective domain statement is added too (line 13).

In table 5.1 and table 5.2 an overview of the RDF graph describing the GMM vocabulary is provided. A row in table 5.1 corresponds to one class declaration according to listing 5.1, lines 6 - 9. The first column depicts the URI of the class. For each URI in table 5.1 a respective class statement (`rdf:type rdf:Class`) exists within the graph. The second column describes the available `rdfs:subClassOf` statements. For each class provided within this column, a sub class statement according to lines 8 and 9 in listing 5.1 is added to the graph.

A row within table 5.2 corresponds to one property declaration according to listing 5.1,

Class URI	rdfs:subClassOf
gmm:GmmElement	
gmm:NamedElement	gmm:GmmElement
gmm:GmmModelContainer	gmm:NamedElement
meta:MetaModelElement	gmm:GmmElement
meta:GmmMetaModel	meta:MetaModelElement
meta:MetaModelNode	meta:MetaModelStereotype, gmm:NamedElement
meta:MetaModelProperty	meta:MetaModelStereotype, gmm:NamedElement
meta:MetaModelEdge	meta:MetaModelElement, gmm:NamedElement
meta:MetaModelEdgeConnection	meta:MetaModelStereotype
meta:ModelLayer	meta:MetaModelElement, gmm:NamedElement
meta:MetaModelStereotype	meta:MetaModelElement
model:ModelElement	gmm:GmmElement
model:GmmModel	model:ModelElement
model:StereotypedElement	model:ModelElement
model:ModelNode	model:StereotypedElement, model:NamedElement
model:ModelEdge	model:StereotypedElement, model:NamedElement
model:ModelProperty	model:StereotypedElement
model:Provide model:ConsumedBy model:LocalizedAt model:StructuralDependentOf model:BehavioralDependentOf model:InstanceOf model:Generalization	model:ModelEdge
model:Human model:Process model:Application model:InfrastructureElement	model:ModelNode

Table 5.1: Class statements for the GMM vocabulary.

lines 10 - 14. For each URI in table 5.2 a corresponding property statement (`rdf:type rdf:Property`) exists within the graph. The second and third column in table 5.2 describe the range and domain of the declared property. For each URI specified within the domain respective range column the corresponding statements are defined in the RDF graph.

EA data representation

The proposed RDF graph is used as vocabulary to store the EA model within the triple store. Elements and relations as well as the meta model elements and meta model relations are represented as resources of the RDF graph. Each resource is associated with its class using the `rdf:type` statement. The properties of elements and relations are also represented as resource. The dependencies between them are realized with an `rdf:Property` statement. Names and the values of the properties are described with literals.

Listing 5.2 shows an excerpt of the RDF graph describing the example model in figure 3.5. It describes the required part of the EA meta model and the EA model, to express that the element *Booking a car* has the *criticality* value *7*. The necessary meta model part comprises the meta model node *business process* (lines 5 - 8) with the associated property *criticality* (line 8). The property itself is described in lines 9 - 12 with its name and type.

Property URI	rdfs:domain	rdfs:range
gmm:uuid	gmm:GmmElement	xsd:string
gmm:name	gmm:NamedElement	xsd:string
gmm:model	gmm:GmmModelContainer	model:GmmModel
gmm:metamodel	gmm:GmmModelContainer	meta:GmmMetaModel
meta:elements	meta:GmmMetaModel	meta:MetaModelNode
meta:edges	meta:GmmMetaModel	meta:MetaModelEdge
meta:layers	meta:GmmMetaModel	meta:MetaModelLayer
meta:incoming	meta:MetaModelNode	meta:MetaModelEdgeConnection
meta:outgoing	meta:MetaModelNode	meta:MetaModelEdgeConnection
meta:properties	meta:MetaModelNode, meta:MetaModelEdge	meta:MetaModelProperty
meta:belongsTo	meta:MetaModelNode	meta:MetaModelLayer
meta:type	meta:MetaModelProperty	xsd:string
meta:connections	meta:MetaModelEdge	meta:MetaModelEdgeConnection
meta:source	meta:MetaModelEdgeConnection	meta:MetaModelNode
meta:target	meta:MetaModelEdgeConnection	meta:MetaModelNode
meta:contains	meta:MetaModelLayer	meta:MetaModelNode
model:elements	model:GmmModel	model:ModelNode
model:edges	model:GmmModel	model:ModelEdge
model:stereotype	model:StereotypedElement	meta:MetaModelStereotype
model:incoming	model:ModelNode	model:ModelEdge
model:outgoing	model:ModelNode	model:ModelEdge
model:properties	model:ModelNode, model:ModelEdge	model:ModelProperty
model:value	meta:ModelProperty	xsd:string
model:source	model:ModelEdge	model:ModelNode
model:target	model:ModelEdge	model:ModelNode

Table 5.2: Property statements for the GMM vocabulary.

In the following the model element *Booking a car* (lines 13 - 17) is described. For this element its type `Process` (line 14), its name (line 15), the stereotype (line 16) as well as the respective criticality property (line 17) is provided. For the property an own resource is defined (lines 18 - 21), where the respective meta model property is referenced (line 20) as well as the value of the property is provided (line 21). The statements declaring the UUID of the elements are omitted to visibility reasons.

```
1  @prefix meta: <http://www.smds.de/autoanalyze/eam/gmm/gmmmetamodel#> .
2  @prefix gmm:   <http://www.smds.de/autoanalyze/eam/gmm/gmm#> .
3  @prefix model: <http://www.smds.de/autoanalyze/eam/gmm/gmmmodel#> .
4  @prefix rentalCar: <http://rentalcarmodel#> .

5  rentalCar:7f2e37ca-2baf-ad18-ea02-2c6328e06ba2
6     a                meta:MetaModelNode ;
7     gmm:name         "business process" ;
8     meta:properties  rentalCar:e1608cdb-4e0e-aad3-fa8b-399259131d0c .

9  rentalCar:e1608cdb-4e0e-aad3-fa8b-399259131d0c
10    a                meta:MetaModelProperty ;
11    gmm:name         "mission criticality" ;
12    meta:type        "T_Integer" .

13 rentalCar:e15d6610-585e-4504-4507-b8dec5469705
14    a                  model:Process ;
15    gmm:name           "Booking a Car" ;
16    model:stereotype   rentalCar:7f2e37ca-2baf-ad18-ea02-2c6328e06ba2 ;
```

```
17    model:properties  rentalCar:e1608cdb-585e-4504-4507-b8dec5469705 .

18  rentalCar:e1608cdb-585e-4504-4507-b8dec5469705
19    a                 model:ModelProperty ;
20    model:stereotype  rentalCar:e1608cdb-4e0e-aad3-fa8b-399259131d0c ;
21    model:value       "7" .
```

Listing 5.2: RDF representation of an EA model.

For each data source of an EA model a named graph is created within the triple store. Thus, the whole EA model can be referenced as a set of named graphs. Thereby, it is important to use the same URI for identical elements within different named graphs. If a new URI has to be created for an element, a preceding search for the UUID or the name within the triple store is recommended. This ensures that there do not exist two URIs for the same architectural element.

Named graphs are also used to store different versions of a data source, i.e. the status of the applications at different point of times, or to persist analysis results. Using named graphs allows the extension of the stored data without changing or extending the original graphs. Combining different models can be done using the union-operator on the dataset. This is illustrated in listing 5.3. Therein, a new property for a business process representing the analysis result *process costs* is defined. Listing 5.3 depicts the named graph which declares a further meta property *process costs* (lines 7-10) with the type integer. In lines 5 - 6 the meta property is assigned to the meta model node *business process*. For the process *Booking a car* also a concrete value of this property is assigned (lines 11-12). The property with its value and a reference to the respective meta property is described in lines 13-16. The original graph as presented in listing 5.2 remains unaffected.

```
1  @prefix meta: <http://www.smds.de/autoanalyze/eam/gmm/gmmmetamodel#> .
2  @prefix gmm:   <http://www.smds.de/autoanalyze/eam/gmm/gmm#> .
3  @prefix model: <http://www.smds.de/autoanalyze/eam/gmm/gmmmodel#> .
4  @prefix rentalCar: <http://rentalcarmodel#> .

5  rentalCar:7f2e37ca-2baf-ad18-ea02-2c6328e06ba2 # MetaModelNode "business process"
6    meta:properties  rentalCar:111-222-333 .

7  rentalCar:111-222-333
8    a               meta:MetaModelProperty ;
9    gmm:name        "process costs" ;
10   meta:type       "T_Integer" .

11 rentalCar:e15d6610-585e-4504-4507-b8dec5469705  # ModelNode "Booking a Car"
12   model:properties  rentalCar:444-555-666 .

13 rentalCar:444-555-666
14   a                 model:ModelProperty ;
15   model:stereotype  rentalCar:111-222-333 ;
16   model:value       "100.000" .
```

Listing 5.3: RDF graph extending the EA model with analysis results.

5.3.2 Accessing the data for analysis purposes

The purpose of the triple store is to provide the EA model data for analysis execution. Analyses within the A2F are either performed with SPARQL or with DFA. SPARQL is used for structural requests like *'Retrieve all processes with criticality over 5'*. DFA is used for behavior approximation like a calculation of expected response times. DFA is also used to consider the indirect dependencies and assess ripple effects.

SPARQL queries can directly be answered by the triple store. To be able to apply the DFA technique, the EA model data has to be transformed into an EMF model. The EMF model must be an instance of the meta model utilized for the definition of the propagation rules. The analysis rules specified with the DFA technique are built upon the GMM presented in section 3.2. Following a converter is required that takes one or more named graph IRIs and returns the corresponding EMF model.

For the model conversion, the *EMF Converter* translates in a first step the stored meta model. All layers, meta model nodes and meta model edges retrieved within the model union of the specified named graphs are translated into the respective `GmmMetaModelElements`. The specified meta properties for meta model nodes and edges as well as the concrete edge connections are converted too. In the following the identified model elements and model edges along with their properties are translated into the respective `GmmModelElements`. Thereby a possible mapping of the types to the element and relation classes is considered.

5.4 Analysis Definition

Analysis definition is done with the text editor provided by the Xtext framework [Ecl18b]. Based on the grammar presented in chapter 4 the respective language infrastructure can be generated. This includes a parser, a linker, a type checker, compiler and editor. Important for analysis definition is especially the textual editor. Among others, it provides support for syntax and semantic coloring, error checking and auto-completion.

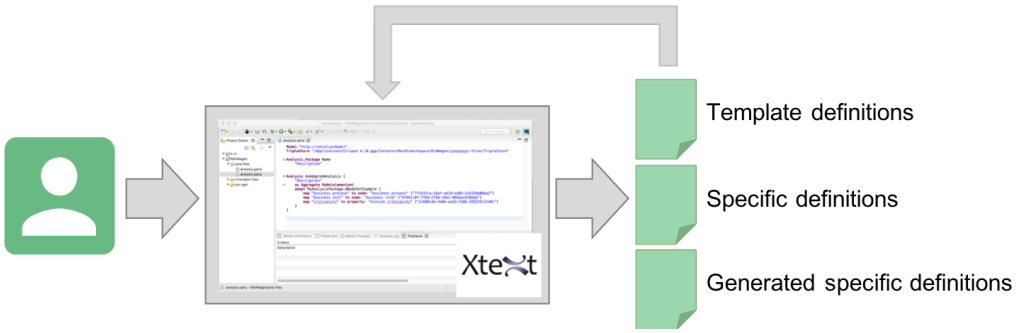


Figure 5.3: Overview of analysis definition.

As illustrated in figure 5.3, the user works with the textual editor to create specific analysis and template definitions. For their creation he can use predefined templates. Additionally, the language editor resolves the defined `AdaptedAnalysis` definition. With a model-to-text transformation an additional file is generated which contains the resolved specific definitions according to the provided mappings for a template. The two resulting files with specific definitions, the manually created one as well as the generated one, are the outcome of analysis definition and further processed within analysis execution.

5.4.1 Analysis definition support

To enable the specification of valid analysis definitions and to ease the use of the provided DSL, the presented grammar in chapter 4 is extended with validation expressions as well as proposals for auto completion. Auto completion proposals are generated by the Xtext framework for all references and enumerations within the DSL, for example the node and edge classes. Additional proposals are provided for node, edge and property references. Therefore, it is necessary that a valid URI, referencing the EA model, and a valid triple store path are provided. Further user support is provided, when specifying an adapted analysis (see figure 5.4). The list of proposed templates that can be used is limited to those that realize the specified analysis style and result type. To support the mapping, the references within the templates are proposed, for whom no mapping is specified yet.

Figure 5.4: Customized mapping proposals for an adapted analysis.

A missing mapping within an adapted analysis will lead to an error message. In figure 5.5 the reference `business proces` cannot be found due to a spelling mistake. Due to this mistake, not all references used in the specified templates are mapped to a concrete stereotype. Further validation is done for the specified result type within an adapted analysis and an analysis composition.

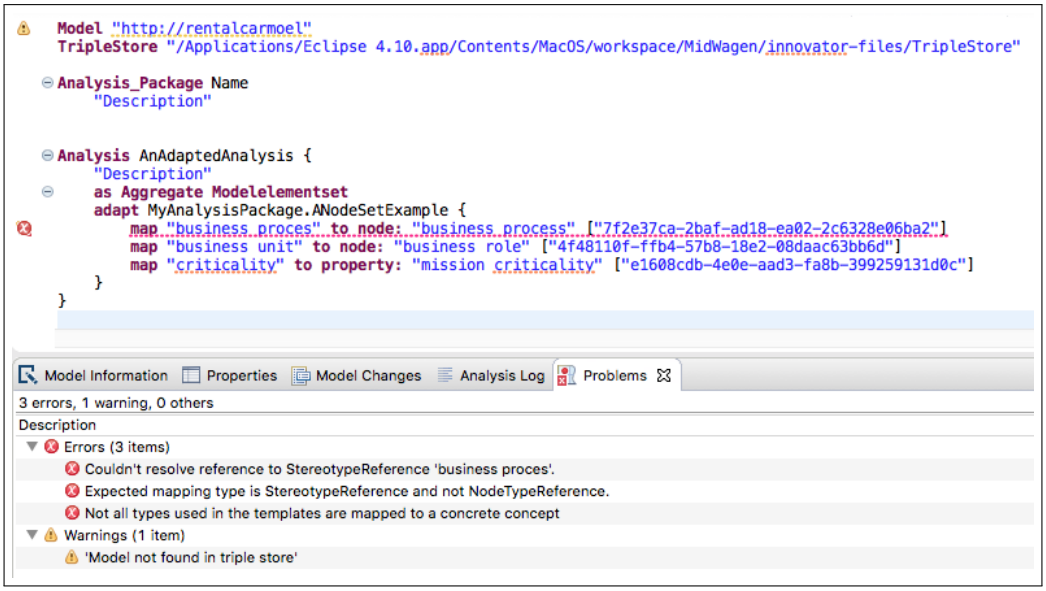


Figure 5.5: Validation of a specific analysis definition.

If the specified model URI within a specific analysis package cannot be found, the language editor provides a warning. In this case no auto completion for node, edge and property stereotypes is possible. A warning is also provided, if it is not possible to establish a connection to the triple store at the specified path.

5.4.2 Utilization of templates

In order to enable the execution of `AdaptedAnalysis` definitions, the language editor performs a model-to-text transformation. During this transformation, for each adapted analysis definition the respective specific definition according to the template is generated. Within the generation the variables of the templates are replaced with the stereotypes provided in the mapping configuration. For executing the transformation, the *Generator* provided by the Xtext framework is extended. Result of this transformation is a second specific analysis definition file, containing an analysis definition for each `AdaptedAnalysis`. The variables within the templates are resolved according to the mappings in the respective adapted analysis definitions.

Listing 5.4 presents an analysis definition for a scope analysis, generated by the model-to-text transformation. The respective adapted analysis configuration is provided in listing 4.21 and the referenced template in listing 4.8.

```
1  Analysis AnAdaptedAnalysis_generated {
2    "Generated analysis definition. Description"
3    as Aggregate Modelelementset
4    defined with set definition:
5      (node:"business role" ["4f48110f-ffb4"]                          AND
6       having relation to ((node:"business process" ["7f2e37ca-2baf"]    AND
```

```
7       having property property:"mission criticality" ["e1608cdb-4e0e"]
8           with value (> 3))))
9  }
```

Listing 5.4: Generated analysis definition for the adapted analysis in listing 4.21.

The name of the original adapted analysis is extended with a `generated` flag, and the description is extended with the information that this is a generated analysis definition. In the following the result type and analysis style are defined according values of the adapted analysis. The subsequent set definition follows the configuration provided within the referenced template. Only the used variables for *business process*, *business unit* and *criticality* are replaced with concrete stereotypes from the current EA model.

The transformation procedure is presented in detail in the following listings using a scope analysis with a node set definition as example. The transformation of the other analysis approaches is done similarly. Listing 5.5 presents the Xtend templates for the transformation of the preceding general information about the EA model, the name and the description of the analysis package. Thereby, the name is extended with the suffix `adaptedAnalysis` (line 4) and the description is extended with a sentence to inform about the generated nature of this package and the original analysis package (line 5). Finally, all analysis definitions which are specified using an `AdaptedAnalysis` are processed (line 6 and lines 8 - 18).

```
1  static def generateSpecificArlaFile(SpecificAnalysisPackage analysis)'''
2    Model "«analysis.modelUri»"
3    TripleStore "«analysis.tripleStorePath»"
4    Analysis_Package «analysis.analysisPackage.name»_adaptedAnalyses
5    "Generated specific analysis for analysis package «analysis.analysisPackage.name»"

6    «analysis.specificDefinition.filter(d | d.body instanceof AdaptedAnalysis).forEach[createDefinition]»
7  '''

8  static def createDefinition(SpecificDefinition definition)'''
9    «val adaptedAnalysis = definition.body as AdaptedAnalysis»
10     Analysis «definition.header.name»_generated {
11       "Generated analysis definition. «definition.header.description»"
12       as «adaptedAnalysis.analysisStyle» «adaptedAnalysis.resultType»
13       «createAnalysisDefinition(adaptedAnalysis.analysis, adaptedAnalysis.mappings)»
14     }

15     «IF adaptedAnalysis.analysis.body instanceof GenericCompositionConfiguration»«
16       resolveCompositeAnalyses(adaptedAnalysis)»«
17     ENDIF»
18 '''
```

Listing 5.5: Model-to-text transformation for the analysis package.

For each `AdaptedAnalysis` a respective analysis definition is generated which utilizes the configuration provided in the referenced template. The name, the description, the analysis style as well as the result type are defined according to the specification within the `AdaptedAnalysis` definition (lines 10 - 12). The body of the analysis is constructed dependent on the type of the referenced template and the provided mappings (line 13). If the referenced template is configured with a `GenericCompositionTemplate`, the referenced templates within the composition rules have to be transformed too (lines 15 - 17).

To illustrate the generation of the analysis body the Xtend template for the `GenericNodeSetConfiguration` is presented in listing 5.6.

```
1  static def createAnalysisDefinition(GenericNodeSetConfiguration template, EList<Mapping> mappings)'''
2    defined with set definition: «getNodeSetCondition(template.configuration, mappings)»
3  '''
```

```
4  static def String getNodeSetCondition(NodeSetCondition condition, EList<Mapping> mappings) {
5    switch condition {
6      PropertyCondition : return "having property " + resolveProperty(condition.propertyName.name, mappings) +
           " with value " + getValueExpression(condition.propertyValue)
7      NodeReference : return resolveNode(condition, mappings)
8      RelationCondition : return "having relation to (" + getNodeSetCondition(condition.nodeSet, mappings) + ")"
9      AndComposition : return "(" + getNodeSetCondition(condition.left, mappings) +" AND " +
           getNodeSetCondition(condition.right, mappings) + ")"
10     OrComposition : return "(" + getNodeSetCondition(condition.left, mappings) + " OR " +
           getNodeSetCondition(condition.right, mappings) + ")"
11     default : return ""
12   }
13 }
```

Listing 5.6: Model-to-text transformation for the scope analysis using a node set condition.

The `createAnalysisDefinition` method is the starting point for the generation and triggers the conversion of the node set condition according to provided mappings in lines 4ff. A `RelationCondition`, an `AndComposition` and an `OrComposition` are evaluated through a recursive call of this method (lines 8 - 10). Finally, to convert the `PropertyCondition` and the `NodeReference` the utilized variables within the template are resolved according the given mapping (lines 6, 7). Resolving the variables of node references is illustrated in listing 5.7. Properties and edges are processed in the same manner.

```
1  static def String resolveNode(NodeReference nodeReference, EList<Mapping> mappings){
2    switch reference {
3        NodeTypeReference: {
4          val mapping = mappings.findFirst(m | m.reference.name.equals(nodeReference.name))
5          return "node:\"" + mapping.typeName.name + "\" [\"" + mapping.typeName.id + "\"]"
6      }
7      NodeClassReference: return "nodeClass:" + nodeReference
8    }
9  }
```

Listing 5.7: Model-to-text transformation to resolve a `NodeReference`.

A `NodeClassReference` is kept also for the specific analysis definition. (line 7). A `NodeTypeReference` has to be resolved according the provided mappings. In this case the variable utilized within the template has to be replaced with the concrete stereotype. (lines 4, 5).

Applying the presented transformation procedure onto a specific analysis package, will result is an additional, generated analysis package containing an analysis definition for each adapted analysis. These two files are the input for the following analysis execution. The utilized template package is not required for analysis execution.

5.5 Analysis Execution

Within analysis execution, the analysis and template definitions specified in Arla are processed and evaluated. As part of this process they are converted into executable SPARQL queries, respectively DFA configurations. Through the combination of those techniques, we are able to provide an analysis execution environment that is able to deal with different meta models, incomplete EA models while covering most of the analysis types applied in the EA context.

5.5.1 Execution approach

A conceptual overview of the execution approach applied within the A2F is provided in figure 5.6. The *Arla Interpreter* takes an analysis specification as input and determines the relevant analysis module for its execution. Analysis modules utilizing the DFA approach for analysis execution are the *Performance module* (for performance analysis definitions), the *Impact module* (for impact analysis definitions), the *Scope definition module* (for scope analyses with edge definitions), the *Path module* (for path definitions) and the *Custom DFA module*. During analysis execution the assignment of the propagation rules to GMM elements is done according to the provided analysis definition. The data-flow rules itself are defined beforehand at design time. Where required, further configuration parameters are set, to customize the analysis execution.

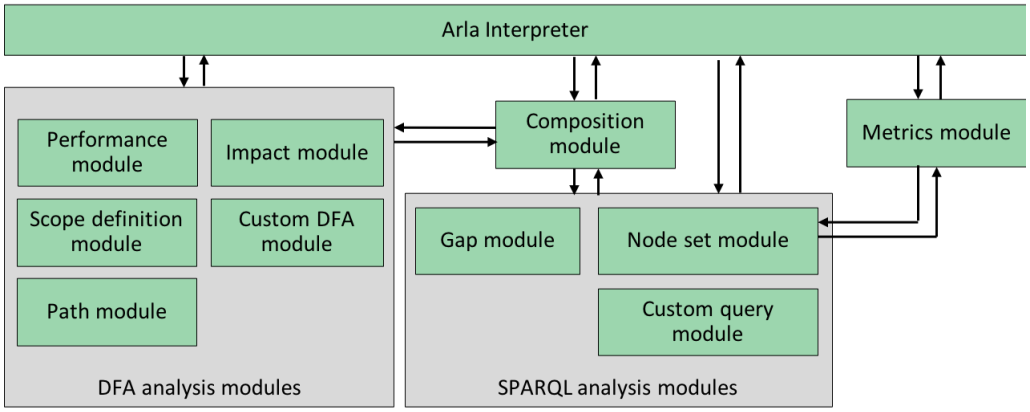


Figure 5.6: Conceptual overview of analysis execution with the A2F.

Analysis modules utilizing SPARQL queries for the execution are the *Gap module* for gap analyses, the *Node set module* for scope analyses with node set definitions, and the *Custom query module*. In this case the respective SPARQL queries are generated or customized, depending on the provided analysis definition.

The *Metric module* does not rely specifically on one of these two techniques. The mathematical operations are evaluated within code. For the evaluation of node set definitions specified within the calculation rule, the respective node set module is triggered. Finally, the *Composition module* triggers the execution of the referenced analyses and combines the provided analysis results. To enable result composition and provide a unified result format, a data structure for its representation was established.

In order to execute a specific analysis definition, the first step is to parse the respective files. The provided parser from the Xtext language infrastructure can be used for this

task and finally provides an EMF model for the analysis package as well as the generated analysis package. The analysis definitions are now available and processable.

SPARQL

Analysis definitions capturing a structural request, like the node set definition for a scope analysis, are evaluated with SPARQL queries. Based on the analysis configuration, the respective SPARQL query is generated and evaluated by the triple store. Figure 5.7 presents an overview of the execution process.

Figure 5.7: Accessing the triple store for the evaluation of SPARQL queries.

For example, the node set definition presented in listing 5.8 requests all elements, having a criticality rating with more than five.

```
1 defined with set definition:
2   having property propertyType:"criticality" with value (> 5)
```

Listing 5.8: Example analysis configuration using a node set condition.

From this analysis configuration, the SPARQL query in listing 5.9 is generated. This query returns a list of URIs identifying elements that have a criticality over five.

```
1  @prefix gmm:   <http://www.smds.de/autoanalyze/eam/gmm/gmm#> .
2  @prefix model: <http://www.smds.de/autoanalyze/eam/gmm/gmmmodel#> .

3  SELECT ?result
4  FROM <http://rentalcarmodel>
5  WHERE {
6    ?model          gmm:elements      ?result
7    ?result         model:properties  ?property .
8    ?property       model:stereotype  ?metaproperty .
9    ?metaproperty   gmm:name          "criticality" .
10   ?property       model:value       ?value .
11   FILTER (?value > 5)
12 }
```

Listing 5.9: SPARQL query generated from the node set definition in listing 5.8.

The elements of the result have to match the following graph patterns: They must have a reference to a `Property` (line 7) and the stereotype of this property must have the name "*mission criticality*" (lines 8, 9). Finally, the value of the property is assigned to the variable `value` (line 10) to which a filter is applied to identify the elements with a value greater than five (line 11). The provided SPARQL result is finally converted into a uniform result structure.

DFA

Behavioral requests and transitive analysis approaches are implemented using the DFA technique. Figure 5.8 illustrates the execution process. Based on the provided URIs in the analysis definition, the *EMF Converter* provides the EMF model.

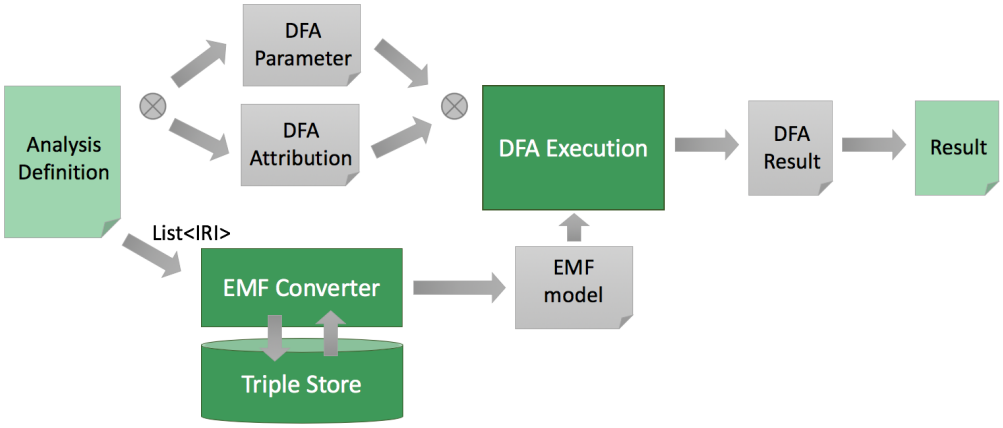


Figure 5.8: Accessing the triple store for the evaluation of DFA rules.

DFA propagation rules can be customized in two different ways: The analysis configuration is either used to generate parameters for DFA execution or an attribution file. If edge classes are used in the analysis configuration, an attribution file is generated, and if concrete stereotypes are used the customization via parameters is applied. This generation step enables the customization of the predefined propagation rules for DFA supported analyses. The DFA result is converted into a uniform representation in the last step.

The concrete evaluation procedures for the different analysis classes are presented in the subsequent sections. Beforehand, in section 5.5.2, the result model for its uniform representation is described.

5.5.2 Result model

The uniform representation of the result enables the utilization of the result within subsequent tasks, like visualization or the integration into an existing EA tool. Figure 5.9 provides the structure of the utilized model for result representation.

The `AnalysisResult` is the top level element. It has a reference to the `Configuration`, utilized for analysis execution. Within the `Configuration` also the respective analysis definition is referenced as well as input parameters like the selected model elements. This class can also be used to provide visualization information like colors for specific values. An `AnalysisResult` can either be an `ElementResult` or an `AggregatedResult`. Depending on the analysis style of the executed analysis the type is chosen. The value of an `ElementResult` is a `Result`, whereas the value of an `AggregatedResult` is provided within a `ResultMap`. Both value types inherit from the super class `ResultValue`.

`ResultMaps` represent the determined result for each element. The element is identified with its `URI` within the map. The retrieved result is provided as `Result`. For example, an entry of the `PathResultMap` consists of the element URI and the respective `PathResult` determined for this element. Within an `IDSetResultMap` this would be an `IDSetResult`, and within the `AttributValueResultMap` an `AttributeValueResult`.

A `Result` depicts one specific kind of result type. Atomic values are `BooleanResult`, `StringResult`, `NumericResult`, and `UriResult`. These atomic result types are utilized within the structured result types `IDSetResult`, `AttributeValueResult` and `PathResult`. An `IDSetResult` consists of a set of `UriResults`. An `AttributeValueResult` is struc-

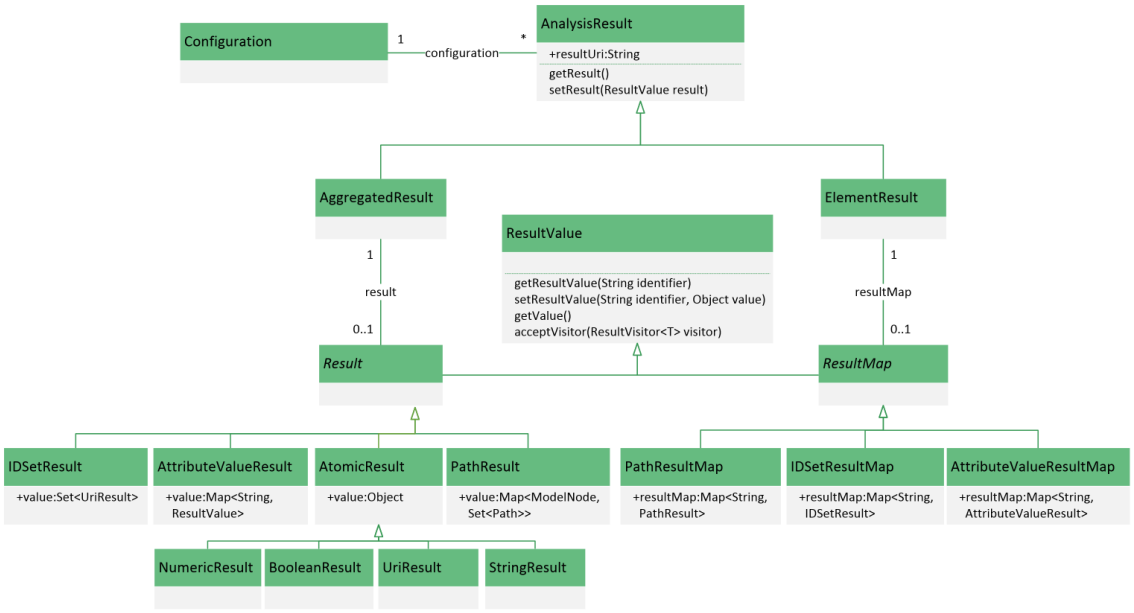


Figure 5.9: Overview of the result model.

tured as a map. The contained `ResultValues` are identified with the attribute name. Therewith, different attribute results can be captured within one `Result` and they can be retrieved according to their identifier, i.e. the analysis that is used to determine them. Finally, the paths within a `PathResult` are structured according to their source node. Thus, the result consists of a map, with a `ModelNode` as key, and a set of `Path` as value. This supports the post-processing of the result for support map creation which is an often used scenario for path analysis.

For the implementation of result specific operations like visualizations or persistence, an abstract `ResultVisitor` is provided to enable operations according to the concrete type of the `ResultValue`.

5.5.3 Execution of scope analysis

The scope analysis class is used to determine views on an EA model. It provides an aggregated result with a set of elements that belong to the defined view. Views can be defined in two different ways in Arla. One is the definition of a `NodeSetDefinition`, specifying constraints over elements, and the other one is the `EdgeDefinition`, specifying constraints over relations. The first one is evaluated using SPARQL queries and described in section 5.5.3.1. The second one is evaluated with the DFA technique and described in section 5.5.3.2.

5.5.3.1 Node set condition

Within a `NodeSetDefinition` constraints on elements are defined. The elements must fulfill these constraints to be part of the view. The node set definition is converted into a SPARQL query during analysis execution. The query returns a list a URIs, identifying the elements within the scope. Based on this list an `AggregatedResult` with an `IDSetResult` is created.

An example conversion of a node set definition into a SPARQL query was provided in listing 5.8 and listing 5.9.

For the generation of the SPARQL query, the node set definition is processed recursively according to its structure. The procedure is illustrated in listing 5.10 using pseudo code. This notation uses a simplification of the utilized Jena API for query creation and execution. The SPARQL query is represented as an object which consists of a `Group` of triples representing basic graph patterns. Filter and further special operators can be added through respective groups, for example a `FilterGroup`.

```
1 static SPARQLResult executeNodeSetCondition(NodeSetCondition object, String resultVariable){
2   Query query = new SelectQuery()
3   query.setResultVariable(resultVariable)
4   query.setFromStatement(object.getRelevantUris())
5   query.setWhereStatement(new NodeSetSwitch(resultVariable).doSwitch(object))
6   return tripleStore.executeQuery(query);
7 }
```

Listing 5.10: Query generation for node set definitions.

Initially the generic query structure is created. This is, setting the query type (line 2), setting the result variable (line 3) and declaring the model URIs within the `FROM` statement (line 4). After determining the `WHERE` part (line 5), the query is executed on the triple store and the result is returned.

The `WHERE` part of the query is determined according to the provided node set definition. For each expression type used within a definition a separate triple group is created. According to the used operator (`AND` or `OR`), the groups are combined. Listing 5.11 depicts the creation of the triple groups for the atomic expressions. To ensure a unique usage of variable names within the different groups, a counter is increased after each group definition and used for variable declaration within the `var` function.

```
1  Group caseStereotypeReference(StereotypeReference object){
2    Group group = new Group();
3    group.addTriple(resultVariable, "model:stereotype", var(stereotype))
4    group.addTriple(var(stereotype), "gmm:name", object.getName())
5    groupCounter++
6    return group
7  }
8  Group caseNodeClassReference(NodeClassReference object){
9    Group group = new Group();
10   group.addTriple(resultVariable, "rdf:type", object.getClassName())
11   groupCounter++
12   return group
13 }
14 Group casePropertyCondition(PropertyCondition){
15     Group group = new Group();
16     group.addTriple(var(model), "gmm:elements", resultVariable)
17     group.addTriple(resultVariable, "model:properties", var(property))
18     group.addTriple(var(property), "model:stereotype", var(metaproperty))
19     group.addTriple(var(metaproperty), "gmm:name", object.getName())
20     if (object.getPropertyValue != null){
21       group.addTriple(var(property), "model:value", var(value))
22       group.addFilterGroup(new PropertyValueSwitch().doSwitch(object.getPropertyValue()))
23     }
24     groupCounter++
25     return group
26 }
```

Listing 5.11: Evaluation of the atomic node set definition expressions.

The atomic expressions, i.e. a `NodeReference` and a `PropertyCondition`, are converted in a straightforward way. For a `StereotypeReference`, the respective triples identifying the stereotype are added (lines 3, 4). For a `NodeClassReference` the triples determining the respective type are added (line 10).

While converting the `PropertyCondition`, the required triples to ensure the appropriate stereotype of the property are added (in specific lines 17 - 19). If the property condition has a further value restriction (line 20), the value is bound to a variable and a `Filter` is used to evaluate it. Within the `PropertyValueSwitch` the filter restriction is created according to the used value comparison operator.

```
1  Group caseAndComposition (AndCompostion object){
2      Group group = new Group();
3      group.addTriples(doSwitch(object.getLeft()) .
4      group.addTriples(doSwitch(object.getRight()
5      return group
6    }
7  Group caseOrComposition (OrCompostition object){
8      return new UnionGroup (doSwitch(object.getLeft()), doSwitch(object.getRight())
9    }
10 Group caseRelationCondition(RelationCondition object){
11   Group incomingGroup = new Group();
12   incomingGroup.addTriple(resultVariable, "model:incoming", var(incomingEdge))
13   incomingGroup.addTriple(var(incomingEdge), "model:source", var(source))
14   incomingGroup.addTriples(new NodeSetSwitch("source").doSwitch(object.getNodeSetCondition()))
15   Group outgoingGroup = new Group();
16   outgoingGroup.addTriple(resultVariable, "model:outgoing", var(outgoingEdge))
17   outgoingGroup.addTriple(var(outgoingEdge), "model:target", var(target))
18   outgoingGroup.addTriples(new NodeSetSwitch(var(target)).doSwitch(object.getNodeSetCondition()))
19   groupCounter++
20   return new UnionGroup(incomingGroup, outgoingGroup)
21 }
22 Group caseNotCondition (NotCondition object){
23   ElementNotExists notExists = new ElementNotExists(doSwitch(object));
24   Group group = new ElementGroup(notExists);
25   return group;
26 }
```

Listing 5.12: Evaluation of the recursive node set definition expressions.

Within listing 5.12 the creation of the triple groups in the case of recursively defined expressions is depicted. These are nested conditions using the `AND` or `OR` operator as well as `RelationConditions`. The `AND` operator specifies that an element has to fulfill both conditions, those on the left side and those on the right side. The respective triples for those conditions can simply joined together (lines 3 - 4). By contrast, when applying the `OR` operator, only one condition has to be fulfilled. This can be implemented with the `union` operation provided by SPARQL (line 8). The result is composed of the one from the first group together with the one from the second group.

Within a `RelationCondition` another node set definition is specified. All elements having a relation to one of those elements are added to the result. For the implementation, two triple groups are created and combined with the `union` operator. One validates all incoming relations (lines 11 - 14) and the other one the outgoing relations (lines 16 - 18). First the respective source or target element is bound to a variable in both cases. This variable is then used to trigger a recursive evaluation of the contained node set definition (line 14 and 18).

Finally, a `NotCondition` is evaluated with the `FILTER NOT EXISTS` operator. The triple

patterns within the filter statement are recursively determined (line 23). The filter statement ensures that for all elements within the result, none of the triple patterns will match.

For example, the following nested node set definition consists of a `NodeReference` (line1) and a `PropertyCondition` (line 2), composed with the `AND` operator:

```
1   node:"application service" ["2f4a3ad3-d698"]  AND
2   having relation to (class:Process)
```

The condition specifies a view of the EA model containing *application services* which have a relation to a *Process*. The generated SPARQL query to receive those elements is presented in listing 5.13.

```
1  @prefix gmm:   <http://www.smds.de/autoanalyze/eam/gmm/gmm#> .
2  @prefix model: <http://www.smds.de/autoanalyze/eam/gmm/gmmmodel#> .
3  SELECT ?result
4  FROM <http://rentalcarmodel>
5  WHERE {
6    { {
7      ?result          model:incoming    ?incomingEdge10.
8      ?incomingEdge10  model:source      ?source10.
9      {?source10       rdf:type          model:Process}
10     }
11     UNION
12     {
13     ?result          model:outgoing    ?outgoingEdge10.
14     ?outgoingEdge10  model:target      ?target10.
15     {?target10       rdf:type          model:Process}
16   } }
17   {
18     ?result          model:stereotype  ?stereotype11.
19     ?stereotype11    gmm:name          "application service"
20   }
21 }
```

Listing 5.13: Generated SPARQL query from the node set definition.

Lines 6 - 16 implement the `PropertyCondition`. A resulting element must have at least one incoming or outgoing edge to an element, identified with the variables `?source10` (lines 7, 8) and `?target10` (lines 13, 14). This source or target element must have the type `Process` (lines 9, 15). Beside this constraint, elements of the result set have to fulfill the triples specified in lines 18 and 19. They represent the `StereotypeReference` and ensure that all elements of the result have a stereotype relation to an element with name *application service*.

5.5.3.2 Edge definition

In contrast to the `NodeSetDefinition`, within the `EdgeDefinition` constraints are assigned to relation types.

The `EdgeDefinition` is evaluated with the DFA approach. For each model node a `scope` attribute is calculated which determines if the respective model node is within the scope or not. An `EdgeDefinition` is evaluated in different ways, depending on its specification using stereotypes or classes. In the first case, the customization of the propagation rules is done with parameters. In the second case, an attribution file is generated which provides the assignment of the propagation rules to the respective meta model classes.

Customization through generation of an attribution file

Customization through the generation of an attribution file is utilized when evaluating a `DynamicEdgeDefinitionByClasses`. For illustration purposes a respective configuration is presented in listing 5.14.

```
1 defined with scope definition: {
2   ModelEdge    in: None    out: None
3   ConsumedBy   in: None    out: Single
4   Provide      in: None    out: Transitive
5 }
```

Listing 5.14: Example configuration for a `EdgeDefinition`.

Within the configuration the scope propagation types `none`, `single` and `transitive`, are assigned to edge classes, for incoming as well as outgoing occurrences. According to this configuration, the edge classes are extended with respective propagation rules to determine the `scope` attribute. The respective `Attribution` file, required to perform the DFA, is generated directly from the configuration. Listing 5.15 presents the Xtend template that is employed for this task.

```
1  static def createAttribution (DynamicEdgeDefinitionByClasses configuration)'''
2    attribution scope {
3      description "Computes a scope attribute.";
4      assignment nodescope :  java call "ScopeByClassesRules.scope_init";
5      assignment edgeincomingscope : java call "ScopeByClassesRules.scope_init";
6      assignment edgeoutgoingscope : java call "ScopeByClassesRules.scope_init";
7      extend gmm.gmmmodel.ModelNode {
8        occurrence nodescope : java call "ScopeByClassesRules.node_scope";
9      }
10     extend gmm.gmmmodel.ModelEdge {
11       occurrence edgeincomingscope :
12         java call "ScopeByClassesRules.incomingedge_«scopeValue(configuration.defaultEdgeIncoming)»scope";
13       occurrence edgeoutgoingscope :
14         java call "ScopeByClassesRules.outgoingedge_«scopeValue(configuration.defaultEdgeOutgoing)»scope";
15     }
16     «definition.edgeDefinitions.forEach[createRule]»
17   }
18 '''
19 static def createRule (EdgeDefinition definition)'''
20   extend gmm.gmmmodel.«definition.class_» {
21     occurrence edgeincomingscope :
22       java call "ScopeByClassesRules.incomingedge_«scopeValue(definition.edgeIncoming)»scope";
23     occurrence edgeoutgoingscope :
24       java call "ScopeByClassesRules.outgoingedge_«scopeValue(definition.edgeOutgoing)»scope";
25   }
26 '''
```

Listing 5.15: Xtend template for scope attribution generation.

After providing the name and a description for the attribution (lines 2 - 3) the attributes that should be determined during analysis are defined together with an initialization rule (lines 4 - 6). In the following, these attributes are assigned to elements of the GMM. To perform the scope analysis, the `ModelNode` is extended with the attribute `nodescope`, and this attribute is determined with the rule `node_scope` (lines 7, 8).

The `ModelEdge` and the respective subclasses are extended with the attribute `edgeincomingscope` and `edgeoutgoingscope`. The rules which are used for the calculation of these attributes are assigned according to the configuration. For the `ModelEdge` this is presented in lines 10 - 14. Additional declarations of scope propagation types are optional.

Thus, the relation classes are only extended in the case of an available declaration in the analysis configuration (line 18 and lines 23 - 28).

A part of the attribution file, generated from the configuration provided in listing 5.14, is shown in listing 5.16. The excerpt depicts the extension of the GMM elements with occurrences of the respective attributes and the assignment of propagation rules for their calculation.

```
1  extend gmm.gmmmodel.ModelEdge {
2      occurrence edgeincomingscope : java call "ScopeByClassesRules.incomingedge_noscope";
3      occurrence edgeoutgoingscope : java call "ScopeByClassesRules.outgoingedge_noscope";
4    }
5  extend gmm.gmmmodel.ConsumedBy {
6    occurrence edgeincomingscope : java call "ScopeByClassesRules.incomingedge_noscope";
7    occurrence edgeoutgoingscope : java call "ScopeByClassesRules.outgoingedge_singlescope";
8  }
9  extend gmm.gmmmodel.Provide {
10   occurrence edgeincomingscope : java call "ScopeByClassesRules.incomingedge_noscope";
11   occurrence edgeoutgoingscope : java call "ScopeByClassesRules.outgoingedge_transitivescope";
12 }
```

Listing 5.16: Excerpt of the generated attribution file.

The super class `ModelEdge` is extended with the attributes `edgeincomingscope` and `edgeoutgoingscope`. As defined in the analysis configuration, the attributes are determined with the `noscope` rule. The attributes for the `consumed by` class are determined by `incomingedge_noscope` and `outgoingedge_singlescope`. And finally, for the class `provide`, the rules `incomingedge_noscope` and `outgoingedge_transitivescope` are used.

Listing 5.17 presents the implementation of the propagation rule for a `ModelNode` in pseudo code. Initially, the status of each model node is `NOT` (line 2). If the current model node is one of the selected start elements, the value `SCOPE` is returned (lines 3 - 4). In lines 6 - 9 the status of the current model node is updated according to the status of its outgoing edges. Therefore, the status from the outgoing scope attribute of the edge is requested (line 7) and the status of the model node is updated. The `setScope` method updates the status according to the strongest value (line 8). For incoming nodes, the same procedure is applied (lines 10 - 13).

```
1   Object node_scope(ModelNode modelNode) throws Exception {
2     ScopeStatus currentScopeStatus = NOT
3     if (currentSelection.contains(modelNode.getUuid())){
4       return currentScopeStatus = SCOPE
5     }
6     for (ModelEdge outgoingEdge : modelNode.getOutgoing()) {
7       ScopeStatus edgeStatus = outgoingEdge.getStatus(ATTRIBUTE.EDGE_OUTGOING_SCOPE)
8       currentScopeStatus = setScope(currentScopeStatus, edgeStatus)
9     }
10    for (ModelEdge incomingEdge : modelNode.getIncoming()) {
11      ScopeStatus edgeStatus = incomingEdge.getStatus(ATTRIBUTE.EDGE_INCOMING_SCOPE)
12      currentScopeStatus = setScope(currentScopeStatus, edgeStatus)
13    }
14    return currentScopeStatus
15  }
```

Listing 5.17: DFA rules to determine the scope value of a `ModelNode`.

For each scope value, i.e. `none`, `single` and `transitive`, a respective propagation rule is defined for incoming as well as outgoing relations. Listing 5.18 presents the propagation rules for outgoing relations in pseudo code.

```
1  Object outgoingedge_transitivescope(ModelEdge edge) {
2    ScopeStatus targetStatus = edge.getTarget().getStatus(
         ATTRIBUTE.NODE_SCOPE)
3    if (targetStatus == SCOPE)
4      return SCOPE
5    return NOT
6  }
7  Object outgoingedge_singlescope(ModelEdge edge) {
8    ScopeStatus targetStatus = edge.getTarget().getStatus(
         ATTRIBUTE.NODE_SCOPE);
9    if (targetStatus == SCOPE)
10     return FINAL;
11   return NOT;
12 }
13 Object outgoingedge_noscope(ModelEdge edge) {
14   return NOT;
15 }
```

Listing 5.18: DFA rules to determine the outgoing scope value of a `ModelEdge`.

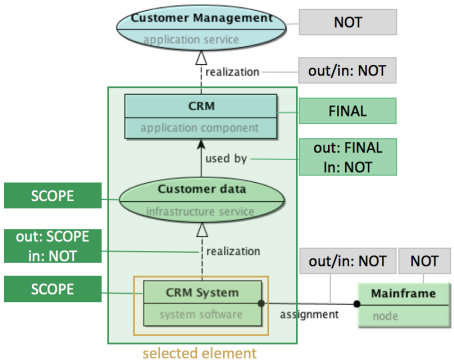


Figure 5.10: Assignment of scope attributes.

In order to enable a context sensitive propagation of the scope status, first, the status of the target element of the current edge has to be requested (line 2 and 8). Based on this information, the status of the current edge is set according to the following rules: In the transitive case (lines 1 - 6), if the target node of the current `ModelEdge` has the scope status `SCOPE`, i.e. this node is included in the scope, then the current model edge is also included (lines 3 - 4). In all other cases, i.e. if the status is `FINAL` or `NOT`, the model edge is not included and thus the status `NOT` is returned (line 5).

The single case is depicted in lines 7 - 12. In contrast to the transitive case, the status of the model edge is `FINAL`, if the target node of this edge is in the scope (lines 9 - 10). This ensures that an edge class, assigned to the `SINGLE` propagation rule, will not be transitively considered. Within the no-scope case (lines 13 - 15), in all cases the status `NOT` is returned.

The rules for the incoming edges are defined accordingly. Instead of requesting the status of the target element of an edge, the source element is used.

In figure 5.10 the application of the rules is illustrated with a small example. Therefore, the analysis configuration in listing 5.14 is used with the selected start element *CRM System*. The *assignment* relationship is part of the `located at` class. Within the analysis configuration no propagation rules are defined for this class, thus the default case (`ModelEdge`) is used and the attribute `NOT` is propagated along the edge and also assigned to the model node *Mainframe*. The `NOT` attribute is also propagated in the other direction, but since the `SCOPE` attribute is stronger than the `NOT` attribute, the value for *CRM System* will not be overwritten.

The *realization* relation belongs to the class `provide` which is evaluated in outgoing direction with the transitive propagation rule. The `SCOPE` attribute is propagated over the edge to the node *Customer Data*. The *usedBy* relation belongs to the class `consumed by` which is evaluated with the single propagation rule in outgoing direction. Hence, the `final` attribute is propagated to the *CRM* application. The *realization* relation to the *Customer Management* propagates a `NOT` value, since the status of its source element is `final`. The determined view according to the edge conditions consists of all elements with attribute values `SCOPE` or `FINAL`.

Customization through the configuration of parameters

If the scope analysis configuration is defined using stereotypes, the customization is performed through the specification of parameters. The parameters consist of four different sets of `MetaModelEdges`. Each set represents one propagation type, i.e. `single` or `transitive` and the direction of the relation, i.e. `incoming` or `outgoing`. The sets are filled according to the `DynamicEdgeDefinitionByStereotypes` configuration.

The propagation rule for the `ModelNode` is the same as for evaluating class configurations (see listing 5.17). The attribute occurrences for model edges are determined differently. Each relation, independently from the relation class, is processed within the `outgoingedge` and `incomingedge` rule of a `ModelEdge`. According to the provided `parameters`, the propagation semantics for the current model edge is defined. For example, the propagation rule for outgoing model edges is shown in listing 5.19.

```
1  @Rule(modelClass=ModelEdge,attribute=edgeoutgoingscope)
2  Object outgoingedge(ModelEdge edge) {
3    ScopeStatus targetStatus = edge.getTarget().getStatus(ATTRIBUTE.SCOPE);
4    if(parameter.getTransitiveOutStereotypes().stream().anyMatch(a -> a.equals(edge.getMetaModelEdge()))){
5      if(targetStatus == SCOPE)
6        return SCOPE
7      return NOT
8    }
9    if(parameter.getSingleOutStereotypes().stream().anyMatch(a -> a.equals(edge.getMetaModelEdge()))){
10     if(targetStatus == SCOPE)
11       return FINAL
12     return NOT
13   }
14   return NOT;
15 }
```

Listing 5.19: Propagation rule to determine the scope value for outgoing model edges.

In this case, the attribution file is stable and no generation is required, since all model edges are evaluated with the presented propagation rule. Thus, the extension of the `ModelEdge` with the attribute `edgeoutgoingscope` and the assignment of the respective propagation be accomplished with Java annotations (line 1). The extension with the attribute `edgeincomingscope` is done respectively.

Within the propagation rule, the scope attribute is set according to the containment of the `MetaModelEdge` of the current model edge in the set of `TransitiveOutStereotypes` or `SingleOutStereotypes`. An example analysis configuration using stereotypes for the scope configuration is provided in listing 5.20. In line 2 the model edge *realization* is assigned to `TransitiveOut`, i.e. an outgoing relation of this type should be evaluated according to the transitive propagation semantic (lines 4 - 8 in listing 5.19). According to line 3, the *usedBy* edge should be evaluated according to the single propagation semantic in the outgoing case (lines 9 - 13 in the listing). The incoming cases as well as all other relations are evaluated according to the no scope semantic (line 14 in listing 5.19).

```
1 defined with scope definition: {
2   TransitiveOut  (edge:"realization" ["c5cf8176-ac56"])
3   SingleOut      (edge:"used by" ["cb2f58ad-8077"])
4 }
```

Listing 5.20: Example scope analysis configuration using stereotypes.

The view consists of all model elements, with a determined attribute value `SCOPE` or `FINAL`. These elements are captured within an `IDSetResult`.

5.5.4 Execution of impact analysis

The impact analysis provides an approximation of the direct and indirect effects of certain events within the architecture, like a change or a failure. For its implementation, the reachability analysis principle of the DFA (presented in section 2.2) is extended with context-specific declarations to restrict the results to a meaningful subset of elements and to reflect different impact types. The effects are simulated by propagating the impact values along the edges of the EA model. The respective propagation rules differentiate between the impact type and the relation type to propagate the correct impact information through the model. In order to execute the impact analysis, the user has to assign impact types to the elements that trigger a respective event, i.e. which are changed or not available.

Within the static execution mode, the propagation semantics are preset for a worst-case and a best-case scenario. Alternatively, they can be customized to enable the individual definition of impact semantics. As within the implementation of the scope analysis, an impact configuration with classes (`ChangeImpactDefinitionByClasses`) is evaluated through the generation of the attribution file. A configuration using stereotypes (`ChangeImpactDefinitionByStereotypes`) is implemented through the definition of DFA parameters.

To formalize the impact semantics for a relation type rt, we employ the following syntax:

$$A.X \rightarrow B.Y \tag{5.1}$$

This statement indicates that, if element A has the characteristic X, then element B will have the characteristic Y. A and B are the source and the target of the relation type rt while X and Y represent the impact type, i.e. $X, Y \in$ {`no impact`, `low`, `medium`, `high`}. Impact types can be clustered on the left-hand side: $A.X, Y \rightarrow B.Z$ states that, if A has type X or Y, B will have the type Z. The impact types are prioritized as followed: `High` overrides `medium` (*med*) overrides `low` overrides `no impact` (*none*). The semantics for the three effects classes used for customization of the impact analysis are defined in table 5.3.

Table 5.3: Impact rules for the effect classes.

effect	**rule**
`strong`	$A.high \rightarrow B.high$
	$A.med \rightarrow B.med$
	$A.low \rightarrow B.none$
`weak`	$A.high \rightarrow B.med$
	$A.med \rightarrow B.low$
	$A.low \rightarrow B.none$
`no effect`	$A.\{high, med, low\} \rightarrow B.none$

If A strongly affects B, this indicates that, if A has a high impact value, B has a high value too. A medium impact type of A leads to a medium status of B and the same applies to extensions. The type of effect is specified for each direction of a relationship. If, for example, an application component realizes a service, then the application component has a strong impact on the service while the service may only have a weak impact on the

application component.

A weak effect denotes that a high impact type for A conducts a medium type for B. A medium value for A leads to a low one for B. Finally, if A has a low impact status, B is not affected by the event. If the relationship is mapped to *no effect*, any impact value of A has no effect on B.

The propagation rule to determine the `impact` attribute for model nodes is presented in listing 5.21. Thereby, the current status of the model node is updated to the strongest value of all incoming and outgoing edges.

```
1  @Rule(modelClass=ModelNode,attribute=impact)
2  Object node_impactstatus(ModelNode node) {
3    ImpactStatus currentStatus = node.getStatus(ATTRIBUTE.IMPACT)
4    for(ModelEdge edge : node.getOutgoing()){
5      ImpactStatus edgeStatus = edge.getStatus(ATTRIBUTE.OUTGOING_IMPACT)
6      currentStatus.update(edgeStatus)
7    }
8    for(ModelEdge edge : node.getIncoming()){
9      ImpactStatus edgeStatus = edge.getStatus(ATTRIBUTE.INCOMING_IMPACT)
10     currentStatus.update(edgeStatus)
11   }
12   return currentStatus
13 }
```

Listing 5.21: DFA propagation rules for the impact status of model nodes.

First the current status of the model node is retrieved (line 3). Then, the outgoing and incoming relations of this model node are processed. The respective impact status attribute is requested (lines 5 and 9) and the current status of the model is updated (lines 6 and 10). This means that the attribute is set to the strongest value according to the prioritization. A `high` will for example override a `medium` value, whereas a `no impact` overrides none of the other impact types.

The implementation of the propagation rules for model edges differs based on the used analysis configuration (static, with edge classes or with stereotypes).

Execution of the static mode

The static mode of the impact analysis, defined with the `StaticImpactDefinition`, differentiates between a worst case and a best case. Therefore, the provided syntax to specify the change semantics is extended to differentiate on the right-hand side between a worst case (WC) and a best case (BC) impact:

$$A.high \rightarrow \begin{cases} B.high & \text{(WC)} \\ B.low & \text{(BC)} \end{cases} \tag{5.2}$$

The rule presents the change semantic for the `provide` class. It indicates that, for example, if a component A has a high impact type, a provided service B has the value high too (worst case). In the best-case scenario, another application may implement the service which is why it only has a low impact type value.

The rules for the relation classes utilized within the static mode are depicted in table 5.4. The first column contains the respective edge class. The second column provides the propagation semantic for incoming relations of this class, and the third column for

Table 5.4: Impact rules for the relationship classes.

edge class	**incoming**	**outgoing**
`located at`	$A.\{high, med, low\} \rightarrow B.none$	$B.high \rightarrow \begin{cases} A.high & \text{(WC)} \\ A.low & \text{(BC)} \end{cases}$ $B.med \rightarrow \begin{cases} A.low & \text{(WC)} \\ A.none & \text{(BC)} \end{cases}$ $B.low \rightarrow A.none$
`provide`	$A.high \rightarrow \begin{cases} B.high & \text{(WC)} \\ B.low & \text{(BC)} \end{cases}$ $A.med \rightarrow \begin{cases} B.med & \text{(WC)} \\ B.none & \text{(BC)} \end{cases}$ $A.low \rightarrow \begin{cases} B.low & \text{(WC)} \\ B.none & \text{(BC)} \end{cases}$	$B.\{high, med, low\} \rightarrow A.none$
`consumed by`	$B.high \rightarrow \begin{cases} A.med & \text{(WC)} \\ A.low & \text{(BC)} \end{cases}$ $B.med \rightarrow \begin{cases} A.med & \text{(WC)} \\ A.low & \text{(BC)} \end{cases}$ $B.low \rightarrow A.none$	$A.\{high, med, low\} \rightarrow B.none$
`structural dependent on`	$A.high \rightarrow \begin{cases} B.high & \text{(WC)} \\ B.med & \text{(BC)} \end{cases}$ $A.med \rightarrow B.none$ $A.low \rightarrow B.none$	$B.high \rightarrow \begin{cases} A.med & \text{(WC)} \\ A.none & \text{(BC)} \end{cases}$ $B.med \rightarrow \begin{cases} A.med & \text{(WC)} \\ A.none & \text{(BC)} \end{cases}$ $B.low \rightarrow \begin{cases} A.low & \text{(WC)} \\ A.none & \text{(BC)} \end{cases}$
`behavioral dependent on`	$A.\{high, med, low\} \rightarrow B.none$	$B.high \rightarrow \begin{cases} A.med & \text{(WC)} \\ A.low & \text{(BC)} \end{cases}$ $B.med \rightarrow \begin{cases} A.med & \text{(WC)} \\ A.none & \text{(BC)} \end{cases}$ $B.low \rightarrow \begin{cases} A.low & \text{(WC)} \\ A.none & \text{(BC)} \end{cases}$
`generalization`	$A.high \rightarrow B.high$ $A.med \rightarrow B.med$ $A.low \rightarrow B.low$	$B.\{high, med, low\} \rightarrow A.none$
`instance of`	$A.\{high, med, low\} \rightarrow B.none$	$B.high \rightarrow A.high$ $B.med \rightarrow A.med$ $B.low \rightarrow A.low$

outgoing relations. For the `provide` example, this means that an element `A` `provides` an element `B`. The equations in the *incoming* column describe the effects on `B`, according to the impact type assigned to `A`. The *outgoing* column describes the opposite direction.

Each edge class is extended with an attribute providing the incoming impact status and the outgoing impact status, in the best case and in the worst case. Listing 5.22 provides the propagation rule of the `provide` class to calculate the attribute value for the worst case.

```
1  @Rule(modelClass=Provide,attribute=incoming_impact_wc)
2  Object incomingprovide_impactpropagation_worstcase(Provide edge) {
3    ImpactStatus sourceStatus = edge.getSource().getStatus(ATTRIBUTE.IMPACT_WC)
4    if(sourceStatus == HIGH) return HIGH
5    if(sourceStatus == MEDIUM) return MEDIUM
6    if(sourceStatus == LOW) return LOW
7    return NO_IMPACT
8  }
9  @Rule(modelClass=Provide,attribute=outgoing_impact_wc)
10 Object outgoingprovide_impactpropagation_worstcase(Provide edge) {
11   return NO_IMPACT
12 }
```

Listing 5.22: DFA propagation rules for the `provide` class in the worst case.

The incoming rule (lines 1 - 8) determines the impact status of the edge according to its source element and the outgoing rule (lines 9 - 12) according to the target element. Depending on the retrieved status, the status of the current edge is defined following the semantics provided in table 5.4 (lines 4 - 7 for incoming relations and line 11 for outgoing relations). The best case and the rules for the other classes are defined accordingly.

Additionally, the `ModelNode` is extended with an attribute `impact_bc` for the best and `impact_wc` for the worst case. The respective propagation rules follow the one presented in listing 5.21. Within the rules the respective worst- or best-case attributes for retrieving the current status and the status of the model edges is used.

Depending on the provided impact analysis configuration, either the worst case or the best case semantic is evaluated. The DFA result is then converted into an `AttributeValue-ResultMap`. The key of the map is the respective URI of the element, and the value is an `AttributeValueResult`. This map consists of one entry for the `impactstatus` with the determined impact type by the DFA.

Figure 5.11 provides the calculated worst-case effects for a severe event at the *Reservation System*. This may be the retirement of the application or a failure. Thus, it is assigned with a `high` impact type to trigger the impact calculation. Within the figure, the colors of the model nodes indicate the impact status of the elements. The propagated impact values along the model edges are illustrated with dashed, colored arrows.

The high impact type is propagated over the *realization* edge. This edge belongs to the `provide` class. Thus, in the worst case a high impact is propagated in the outgoing direction. In the incoming direction no impact is propagated, regardless of the status of the target element.

The subsequent *used by* edge to the business process *Booking a Car* propagates only a medium impact probability. Since this medium status is the strongest one of all incoming and outgoing relations of this element, this is also the final impact status of the process. The further attributes of the relations and elements are determined accordingly.

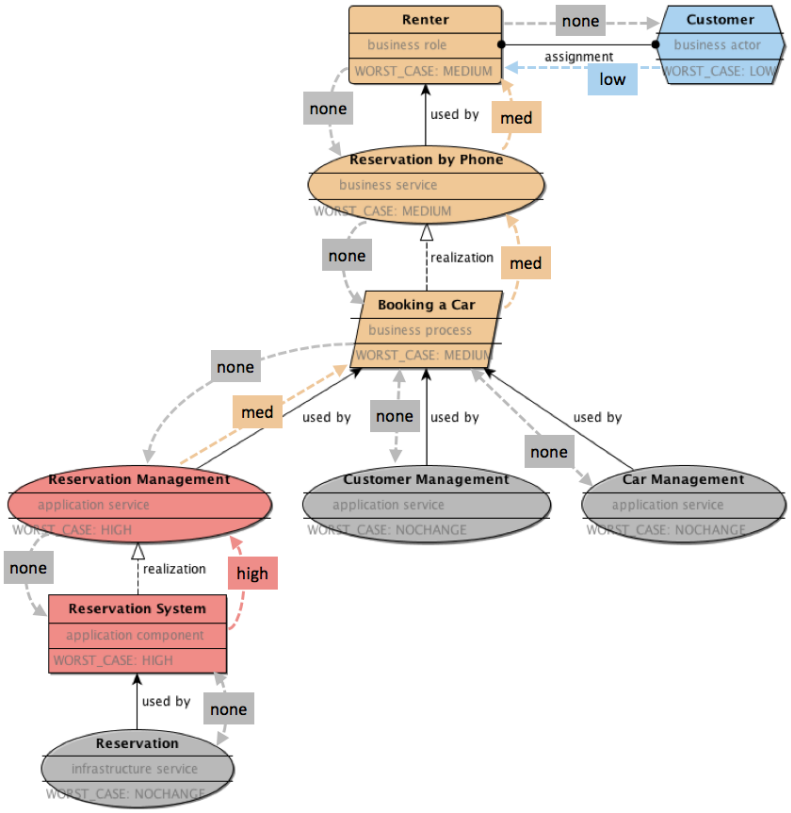


Figure 5.11: Illustration of the result for an impact analysis.

Customization through generation of an attribution file

Despite the static mode to approximate the impact of an event, the A2F supports the individual definition of propagation semantics using the proposed effects types: `strong`, `weak` and `no effect` (see table 5.3 for their semantics). Using the `ImpactDefinitionByClasses` for analysis configuration, these effect types are assigned to the edge classes. Listing 5.23 provides an example impact analysis configuration for this case.

```
1   defined with impact definition: {
2     ModelEdge                in: no_effect       out: no_effect
3     Provide                  in: strong_effect   out: weak_effect
4     ConsumedBy               in: weak_effect     out: no_effect
5     StructuralDependentOf    in: strong_effect   out: weak_effect
6   }
```

Listing 5.23: Impact analysis configuration using classes.

For evaluation of the configuration, an attribution file is generated which provides the assignment of the respective DFA propagation rules to the edge classes. The generation of the attribution files is realized using a model to text transformation. The structure of the respective Xtend template is similar to the one for the scope analysis provided in listing 5.15. The relevant part of the transformation, where the propagation rules are assigned to the respective GMM elements is presented in listing 5.24.

```
1   static def createAttribution (ImpactDefinitionByClasses configuration)'''
2   attribution impact {
3     ...
4     extend gmm.gmmmodel.ModelEdge {
```

```
5         occurrence incoming_impact : java call "ImpactRules.«configuration.defaultEdgeIncoming»_incoming";
6         occurrence outgoing_impact : java call "ImpactRules.«configuration.defaultEdgeOutgoing»_outgoing";
7       }
8       «definition.edgeDefinitions.forEach[createRule]»
9     }
10    '''

11    static def createRule (EdgeDefinition definition)'''
12    extend gmm.gmmmodel.«definition.class_» {
13      occurrence incoming_impact : java call "ImpactRules.«definition.edgeIncoming»_incoming";
14      occurrence outgoing_impact : java call "ImpactRules.«definition.edgeOutgoing»_outgoing";
15    }
16    '''
```

Listing 5.24: Part of the Xtend template for impact attribution generation.

The assignment of propagation rules to the super class `ModelEdge` is done in lines 4 - 8. According to the provided analysis configuration, further attribute extensions are provided to the subclasses (lines 10 and 14 - 19). In addition to the assignment of an `incoming_impact` and `outgoing_impact` attribute to the model edge and its subclasses, an attribute `impact` is assigned to model nodes. The `impact` attribute for the model node is evaluated according to the propagation rule presented in listing 5.21.

To determine the attribute values for the model edges, five different propagation rules are defined: One rule for each effect type and for each direction of the relation. The propagation rules, used to determine the `outgoing_impact` attribute, are presented in listing 5.25. The ones for the `incoming_impact` are defined accordingly. Instead of calling the target element, the status of the source element is retrieved.

```
1  Object strong_effect_outgoing (ModelEdge edge){
2    ImpactStatus status = edge.getTarget().getStatus(ATTRIBUTE.IMPACT)
3    if(status == HIGH) return HIGH
4    if (status == MEDIUM) return MEDIUM
5    return NO_IMPACT
6  }

7  Object weak_effect_outgoing (ModelEdge edge){
8    ImpactStatus status = edge.getTarget().getStatus(ATTRIBUTE.IMPACT)
9    if(status == HIGH) return MEDIUM
10   if (status == MEDIUM) return LOW
11   return NO_IMPACT
12 }

13 Object no_effect_outgoing (ModelEdge edge){
14   return NO_IMPACT
15 }
```

Listing 5.25: Propagation rules for impact calculation for outgoing edges.

Based on the retrieved status (lines 2 and 8) the impact value for the model edge is set. Thereby, the semantics of the effect classes provided in table 5.3 are used. In the `strong` case the same status as retrieved from the edge target is returned (lines 3 - 4). Only in the case of a low impact probability, a `no impact` is returned (line 5). In the `weak` case, a high impact type at the target node leads to a medium impact at the edge (line 9). And a medium one at the target node leads to a low impact type value (line 10). Otherwise `no impact` is propagated (line 11). The `no_effect` rule always returns `no impact`. Regardless of the status at the target element (line 14).

Customization through parameters

If the impact analysis configuration is specified using stereotypes, i.e. a `ImpactDefini-`

tionByStereotypes is used, the customization of the propagation rules is performed with parameters. The parameters are comprised of four different sets of stereotypes:

- IncomingWeakStereotypes,
- IncomingStrongStereotypes,
- OutgoingWeakStereotypes, and
- OutgoingStrongStereotypes.

The content of the sets corresponds to the assignment of the stereotypes to impact effects within the analysis configuration (see listing 5.26).

```
1 defined with impact definition {
2   WeakEffect   In  (edge:"used by" ["cb2f58ad-8077"])
3   StrongEffect In  (edge:"realization" ["c5cf8176-ac56"],
4                     edge:"aggregation" ["4fa14568-ceaa-49f7"])
5   WeakEffect   Out (edge:"realization" ["c5cf8176-ac56"],
6                     edge:"aggregation" ["4fa14568-ceaa-49f7"])
7 }
```

Listing 5.26: Impact analysis configuration using stereotypes.

Within the propagation rules, used to calculate the incoming_impact and outgoing_impact attribute for the model edges, the parameters are used to determine the propagated impact value. This is illustrated with the propagation rule determining the outgoing impact value in listing 5.27.

```
1 @Rule(modelClass=ModelEdge,attribute=outgoing_impact)
2 Object outgoing_edge_impactpropagation (ModelEdge edge){
3   ImpactStatus targetStatus = edge.getTarget().getStatus(ATTRIBUTE.IMPACT)
4   if(parameters.getOutgoingStrongStereotypes().contains(edge.getStereotype()))
5     return propagateStrongEffect(targetStatus)
6   if(parameters.getOutgoingWeakStereotypes().contains(edge.getStereotype()))
7     return propagateWeakEffect(targetStatus)
8   return NO_IMPACT
9 }
```

Listing 5.27: Propagation rules for impact calculation for outgoing edges.

First, the current impact status of the target element of the edge is requested (line 3). For the incoming propagation rule, this would be the source element of the edge. The propagated impact value is determined according to the stereotype of the current model edge and its occurrence in one of the stereotype sets of the parameters. If the stereotype is contained in the OutgoingStrongStereotypes, the status of the model edge is set according to the strong effect semantics (lines 4, 5). If the stereotype is contained in the OutgoingWeakStereotypes, the status of the model edge is set according to the weak effect semantics (lines 6, 7). In all other cases no impact is propagated. Evaluating the propagation rules according to the parameters provides finally the resulting impact attribute for model nodes.

5.5.5 Execution of path analysis

With the path analysis, the direct and indirect dependencies between the elements of the EA model can be analyzed. Three different execution procedures are provided within A2F which correspond to the different configuration possibilities within Arla. These are the determination of all paths between source and target element and the determination of

only the shortest path. Both path types can be further restricted to specific set of relation types. For the all path analysis a restriction based on classes is also possible.

For the implementation of the path analysis the generic flow path analysis template for control flow graphs provided in [Saa14] (see section 2.2) has been tailored to the EA domain. Since an EA does not provide a control flow structure and designated start nodes, the proposed definition has to be adapted. The start element is replaced by a set of start nodes, and since designated end elements are also missing, an optional set of target elements can be provided. The definition of a maximum number of hops, i.e. intermediate elements on a path, provides another stop criterion.

The implementation of the propagation rules differs, whether the `PathByStereotypes` or the `PathByClasses` condition is used to restrict the considered relations. In the first case, the respective incoming and outgoing relation types are provided as parameter within the propagation rules. In the second case, the configuration is used to generate the respective attribution file.

In both cases the following parameters supplement the customization:

- a set of source nodes,
- a set of target nodes,
- the number of maximum hops allowed for each path, and
- a Boolean value which indicates if a path should be ignored if the same stereotype occurs twice on the same path.

Finally, each analysis provides an `AggregatedResult` containing a `PathResult`. The details for the two different implementation modes are described in the following.

5.5.5.1 Path configuration with stereotypes

The DFA propagation rule used to evaluate a path configuration with a `PathByStereotype` condition is presented in listing 5.28. The rule describes the path definition within the `AllPath` mode. In this case, the `ModelNode` is annotated with a data-flow attribute `ALLPATH`. The DFA solver yields a result in the form of a map, where each entry consists of a `ModelNode` and a set of associated outgoing `Paths`. The key identifies the source element of the paths provided within the set.

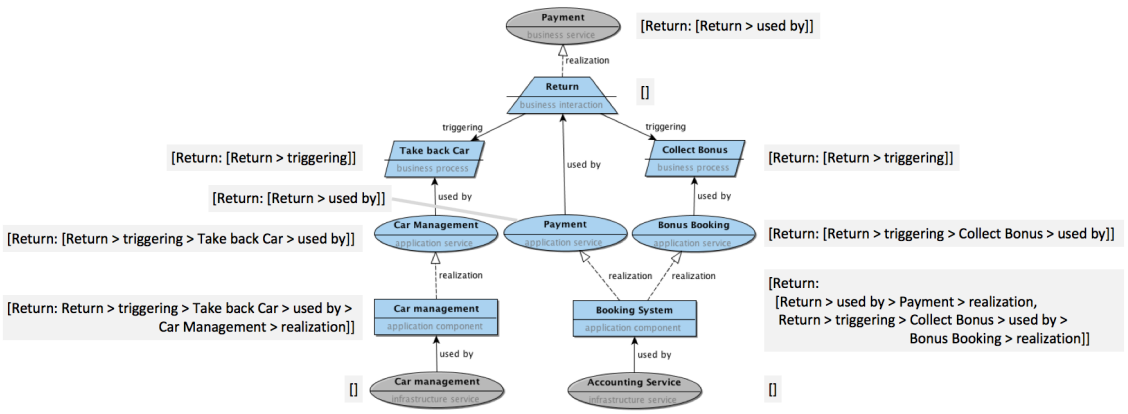


Figure 5.12: Illustration of the execution of the all path analysis.

The idea of the `AllPath` algorithm is that the path from the current model node to the

specified source node is determined through requesting and updating the paths from the referenced elements. Figure 5.12 illustrates the procedure. In the figure the `AllPath` option is chosen to determine all possible path from the *Return* element to *application components*. The maximum number of hops is *4* and same stereotypes on one path are allowed. There are no further constraints provided for the relation types. Hence, all relations are considered. In total, three paths are identified (see figure 4.5 in the previous chapter). The elements that occur at least on one path of the result are highlighted in blue within figure 5.12. The values of the attributes, as determined by the DFA, are annotated for each element.

For path calculation all related elements to the current processed elements are considered (line 3 within listing 5.28). The first step evaluates, if the relation should be considered for path calculation. This includes the verification of the relation type according to the provided path configuration (line 4). Also, if the referenced node is one of the target nodes, the result must not be updated (line 5). This is the case for the infrastructure services *Car management* and *Accounting Service* at the bottom. If the `sameStereotype` parameter is set, and the referenced node has the same stereotype as the current one, no update of the paths is performed (line 6)

New paths are only created for elements, related to a source element, i.e. both *Payment* elements, *Collect Bonus* and *Take back Car* (lines 7 - 11). A new path is initialized with the source element, i.e. `Return` and the respective relation. Since the *Return* element is the start element, there exists no path for it and the path result is empty.

Subsequently, the paths retrieved from the referenced element are retrieved updated (lines 12 - 24). If the retrieved status is still `null`, the next relation is processed (line 13).

```
1  Object node_allpaths(ModelNode currentNode) {
2    Map<ModelNode,Set<Path>> result = new Map<ModelNode,Set<Path>>();
3    for((ModelEdge edge, ModelNode referencedNode) : node.getReferenceMap()){
4      if(!parameters.validate(edge)) continue
5      if(parameters.getTargetNodes().contains(referencedNode)) continue
6      if(parameters.ignoreSameStereotype && referencedNode.getStereotype() == currentNode.getStereotype())
           continue
7      if(parameters.getStartNodes().contains(referencedNode)){
8        Path path = new Path()
9        path.add(referencedNode).add(edge)
10       result.addPath(referencedNode, path)
11     }
12     Map<ModelNode,Set<Path>> targetStatus = referencedNode.getStatus(ATTRIBUTE.ALLPATH)
13     if(targetStatus = null) continue
14     for((ModelNode pathSource, Set<Path> pathSet) : targetStatus){
15       if(pathSource = currentNode) continue
16       for(Path path : pathSet){
17         if(path.getHops() == parameters.maxHops) continue
18         if(path.contains(currentNode)) continue
19         if(paremeters.ignoreSameStereotype && path.exists(e -> e.stereotype == currentNode.stereotype))
               continue
20         Path extendedPath = path.clone()
21         extendedPath.add(referencedNode).add(Edge)
22         result.addPath(pathSource, extendedPath)
23       }
24     }
25   }
26 }
```

Listing 5.28: DFA propagation rules for `AllPath` calculation.

For the element *Bonus Booking*, the paths determined at *Collect Bonus* and at *Booking*

System have to be considered. An update is only performed, if this does not result in a cyclic path to the path's source node (line 15). Additionally, the maximum number of hops, i.e. intermediate model elements, must not be exceeded (line 17). To prevent cycles to the current node, it must not be present on the path (line 18). Finally, the `sameStereotype` parameter is evaluated (line 19). If none of these conditions is met, the retrieved paths from the referenced node are copied (line 20) and extended with the current edge and the referenced node (line 21). The new path is then added to the result set (line 22). In the case of *Bonus Booking* no condition is met for the retrieved value from *Collect Bonus*. Thus, the retrieved path is extended. The paths determined at *Booking System* have no influence, since this element is already one of the target elements.

For the *Booking System*, two outgoing relations have to be considered. Since the respective target nodes of the relations each provide a path, both are extended and added to the result value for the attribute of this node. For the *Accounting Service* no path is determined, since the previous *Booking Systems* belongs to the target nodes. The *Payment* business service is omitted within the post-processing of the result, since the target type of the path is not an *application component*.

The result map is further processed after analysis execution. Within the DFA results, the intermediate path sets, calculated at elements which are not a target element, are included as well. Thus, the paths which do not end with a declared target element, have to be removed afterwards. Finally, all paths are summarized within one `PathResult`.

The `ShortestPath` option is evaluated similar to the `AllPath` option. Instead of determining a set of paths, only the shortest path will be kept within the result. Thus, the attribute value consists of a map with entries `ModelNode` and only one assigned `Path`. The `ModelNode` depicts the source of the related `Path`.

Within the propagation rule, instead of adding the updated path to the result set (see line 22 of listing 5.28), the extended path is compared with actual path to this source node. If it is shorter, then the extended path is kept within the result and replaces the previous one.

The shortest path mode, restricts the potential exponential number of path combinations. Thus, in cases where the shortest path is sufficient, it is recommended to choose this mode.

5.5.5.2 Path configuration with classes

A path definition utilizing the `PathByClasses` condition is implemented with the generation of the respective DFA attribution file and the subsequent evaluation of the DFA propagation rules. The path creation is not only performed for model nodes, also the model edges are extended with respective `incomingedge_path` and `outgoingedge_path` attributes. Model nodes are extended with a `node_path` attribute. Consequently, four DFA propagation rules are defined to determine the values of these attribute occurrences: One for the model nodes, one for edges which should not be considered and one rule for the incoming as well as outgoing path attribute, in the case the relation should be considered.

The propagation rules for the attribute occurrences at model edges are provided in listing 5.30. The `node_path` attribute for model nodes is always determined with the respective `node_rule` provided in listing 5.29. For all elements which are not within the set of start elements, an empty result is returned (line 4). For start elements, all incoming and

outgoing relations are processed.

For each incoming edge the `incomingedge_path` attribute value is requested (line 6). The received paths are extended with the source of the edge, the edge as well as the current node (line 8) and finally included in the result (line 9).

The same procedure applies for outgoing edges. The `outgoingedge_path` attribute value is requested and the path is extended with the respective target node instead of source node (lines 11 - 16).

```
1  @Rule(modelClass=ModelNode, attribute=node_path)
2  Object node_rule(ModelNode currentNode) {
3    Map<ModelNode,Set<Path>> result = new Map<ModelNode,Set<Path>>()
4    if(!parameters.getSourceNodes().contains(currentNode)) return result;
5    for(ModelEdge edge : currentNode.getIncoming()){
6      Map<ModelNode,Set<Path>> targetStatus = edge.getStatus(ATTRIBUTE.INCOMINGEDGE_PATH)
7      if(targetStatus == null) continue
8      targetStatus.updateAllPaths(edge.getSource(), edge, currentNode)
9      result.addAllPaths(edge.getSource, targetStatus)
10   }
11   for(ModelEdge edge : currentNode.getOutgoing()){
12     Map<ModelNode,Set<Path>> targetStatus = edge.getStatus(ATTRIBUTE.OUTGOINGEDGE_PATH)
13     if(targetStatus == null) continue
14     targetStatus.updateAllPaths(edge.getTarget(), edge, currentNode)
15     result.addAllPaths(edge.getSource, targetStatus)
16   }
17 }
```

Listing 5.29: DFA propagation rule for path calculation based on class restrictions.

Figure 5.13 provides the result of a path analysis with a class configuration. The analysis was executed with *Return* as start element and the target elements should be of the type *infrastructure service*. Additionally, the class constraint `Incoming (modelClass:Provide, modelClass:ConsumedBy)` is provided. Thus, only the incoming *used by* and *realization* relations are considered. For all other relations the provided attribute value is an empty path set. According to the propagation rule for `ModelNodes`, only the start element has a non-empty path set.

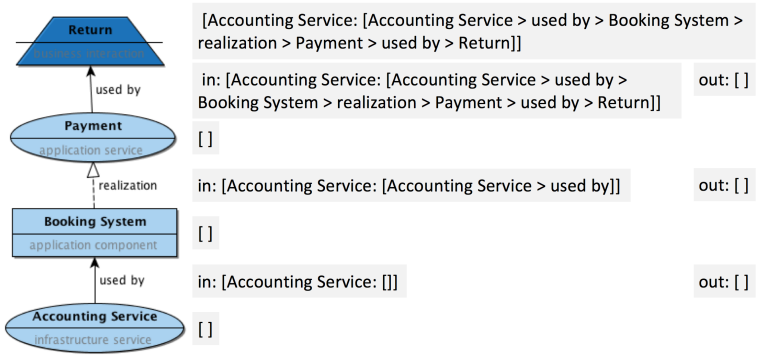


Figure 5.13: Illustration of the execution of the realizing path analysis.

The utilized propagation rules to determine the path sets for the model edges are provided in listing 5.30. This includes one rule to determine paths for incoming directions of relations, one for outgoing directions of relations and one if the relation should not be considered. The rules are assigned to relation classes of the GMM according to the provided `PathByClasses` condition. As for the scope and impact analysis, a generation of

the respective DFA attribution is used for this task.

If no class constraints are provided, all relations, independently of the direction, are considered. Thus, the attribute occurrences for `ModelEdge` are defined with the `incomingpaths_rule` and the `outgoingpaths_rule`.

If constraints are provided, the rule `nopaths_rule` is utilized for the attribute occurrences at `ModelEdges`. The occurrences at the relation classes are defined according to the provided constraints.

The `nopaths_rule` is called if the respective model edge should not be considered during path calculation. Thus, always an empty result is provided (see lines 1 - 3).

If the respective edge should be considered the `incomingpaths_rule`, respectively `outgoingpaths_rule`, is triggered, depending on the requested attribute. To determine the value of the `incomingedge_path` attribute, the source node of the edge will be processed (lines 6ff). If the source element is one of the target elements within the path configuration, the path is initialized with the source element and an empty path (lines 6 - 7). This ensures that the path calculation is finished, if one of the target elements is reached. In the above example, this applies to the outgoing *used by* relation of the *AccountingService*.

In the other case, all incoming edges are processed (lines 10ff). From the `followingEdge`, the current status is requested (line 11). If the status is still null (line 12) or if the `sameStereotype` parameter applies (line 13), the retrieved paths are not considered. Otherwise, all paths are processed, if its source is not equal to the source node of the current edge (lines 14 - 15).

Assuming that the edge under consideration is the *realization* relation in figure 5.13, the `followingEdge` is the outgoing *usedBy* relation of the *Accounting Service*. Since the *Accounting Service* as source element of the path is not equal to the source of the current edge (i.e. *Booking System*) the retrieved path is considered.

```
1  Object nopaths_rule (ModelEdge modelEdge) {
2    Map<ModelNode,Set<Path>> result = new Map<ModelNode,Set<Path>>()
3  }
4  Object incomingpaths_rule(ModelEdge modelEdge) {
5    Map<ModelNode,Set<Path>> result = new Map<ModelNode,Set<Path>>()
6    if(parameters.getTargetNodes().contains(modelEdge.getSource())){
7      result.addPath(modelEdge.getSource(), new Path())
8      return result
9    }
10   for(ModelEdge followingEdge : modelEdge.getSource().getIncoming()){
11     Map<ModelNode,Set<Path>> followingEdgeStatus = followingEdge.getStatus(ATTRIBUTE.
           INCOMINGEDGE_PATH)
12     if(targetStatus == null) continue
13     if(parameters.ignoreSameStereotype && followingEdge.getSource().getStereotype() == modelEdge.getSource.
           getStereotype()) continue
14     for((ModelNode pathSource, Set<Path> pathset) : followingEdgeStatus){
15       if(pathSource == modelEdge.getSource()) continue
16       if(pathSet.isEmpty() && followingEdge.getSource() == pathSource){
17         NodePath path = new NodePath()
18         path.add(followingEdge.getSource()).add(followingEdge)
19         result.add(followingEdge.getSource(), path)
20         continue
21       }
22       for(Path path : pathset){
23         if(path.getHop() == parameters.maxHops) continue
24         if(path.contains(modelEdge.getSource())) continue
25         NodePath extendedPath = path.clone();
```

```
26              extendedPath.add(followingEdge.getSource()).add(followingEdge)
27              result.add(followingEdge.getSource(), extendedPath)
28          }
29      }
30    }
31  }
32  Object outgoingpaths_rule(ModelEdge modelEdge) {
33    ...
34  }
```

Listing 5.30: DFA propagation rules for realizing path calculation.

In the case where the path set is still empty and the source of the `followingEdge` is the desired source of the path, a new path is created and included within the result (lines 16 - 21). Otherwise, the paths are copied and extended with the `followingEdge` and its source element (lines 25 - 27). Only if the provided maximum numbers of hops is reached (line 23) or if the path already contains the source node of the current model edge (line 24) it is ignored. In the example, the retrieved empty path is extended with the *Accounting Service* and its outgoing *used by*. This path is then the result for the *realization* edge.

The procedure for the determination of the `outgoingedge_path` attribute is similar to the presented one. Instead of processing the source node of the edge, the target node and its outgoing relations are processed (lines 32 - 34).

After evaluating the DFA propagation rules the result is transformed into the uniform result structure. All determined paths at model nodes are aggregated into one `PathResult`.

5.5.6 Execution of metrics

An aggregated or an element metric is evaluated through its decomposition into the single expressions. Property references and expressions questioning a set of elements or relations are evaluated through the creation and execution of a SPARQL query. The other non-atomic expression, like metric reference, `SUM`, `MULT` and mathematical operations like addition and subtraction are evaluated recursively. For the evaluation of an aggregated metric and an element metric the same procedure is applied. Except that for an element metric, the provided expression is evaluated for each of the desired elements. In this case, an attribute `currentUri` provides the URI of the actual element.

Listing 5.31 describes the evaluation of the atomic expression. A `TypedPropertyReference` depicts the property value for the current element (line 3). Beforehand, it is validated that the type of the current element corresponds to the type provided within the property expression (line 2). In the other case which also applies if no current element is set, the value `null` is returned (line 4).

The `NodeCount` expression references a `NodeSetDefinition`. The definition is evaluated according to the procedure described in section 5.5.3.1 and the size of the result set is returned (line 6).

```
1  Float caseTypedPropertyReference :
2    if(validate(currentUri, expression.getTypeName()))
3      return SparqlQuery.getPropertyValue(currentUri, expression.getPropertyName())
4    return null;
5  Float case NodeCount :
6    return executeNodeSetCondition(expression.getNodeSetCondition()).size()
7  Float caseEdgeCount :
```

```
8    Float result = 0;
9    if(currentUri.isEmpty()){
10     for(String uri : SparqlQuery.getAllModelElements())
11       result += evaluate(expression.getValue(), uri)
12     return result;
13   }
14   if(validate(currentUri, expression.getTypeName()))
15     return evaluate(expression.getValue(), currentUri)
16   return result;

17 Float caseUndirectedEdgeCount :
18   return SPARQLQuery.getRelations(currentUri, expression.getEdgeType(), getQuery(expression.
         getNodeSetCondition())).size()

19 Float caseOutgoingEdgeCount :
20   return SPARQLQuery.getAllOutgoingEdges(currentUri, expression.getName()).size()

21 Float caseIncomingEdgeCount :
22   return SPARQLQuery.getAllIncomingEdges(currentUri, expression.getName()).size()
```

Listing 5.31: Evaluation procedures for the atomic expressions of a calculation rule.

The number of outgoing and incoming relations of an element is evaluated with the `EdgeCount` expression. In the case of an aggregated metric, no `elementUri` is set and the expression is evaluated for all elements within the model. The single values are summed up (lines 9 - 13). In the case of an element metric, the `currentUri` is set and the expression is evaluated for this element (lines 14, 15). The evaluation of an `EdgeCount` distinguishes between the three types. The `UndirectedEdgeCount`, the `OutgoingEdgeCount` and the `IncomingEdgeCount` to either concentrate on incoming, outgoing or both directions of relations.

The `UndirectedEdgeCount` is expressed with a `NodeSetDefinition`. The result is defined as the number of all relations, from the current element to one of the elements within the node set. Additionally, the type of the relation can be restricted too. For implementation, the query for the `NodeSetDefinition` is retrieved and extended with a triple to ensure the existence of a relation to the current element (line 18).

The `OutgoingEdgeCount` returns the number of outgoing relations from the current element. The `IncomingEdgeCount` returns the number of incoming relations. Both expressions are evaluated using SPARQL queries. The queries are parameterized, to enable their execution depending on the actual value of the current element. Listing 5.32 provides query to retrieve all outgoing relations of type `relationType` for the element `currentUri`.

```
1 SELECT ?resultURI
2 FROM <relevantUris>
3 WHERE {
4   <currentUri>     model:outgoing      ?outgoing.
5   ?outgoing        model:target        ?resultURI.
6   ?outgoing        model:stereotype    ?stereotype.
7   ?metamodeledge   meta:connections    ?stereotype.
8   ?metamodeledge   gmm:name            <relationName>
9 }
```

Listing 5.32: Parameterized query to retrieve all outgoing relations.

The evaluation procedures for the non-atomic expressions are provided in listing 5.33. Thereby, expressions provided within a `Sum` or a `Mult` are either evaluated for all model elements (lines 7 and 16) or only for the elements in the specified node set (lines 4, 5 and 13, 14). The retrieved results are added together or multiplied with each other (line 9 and 18 respectively).

```
1  Float caseSum :
2    Float result = 0;
3    Set<String> relevantUris;
4    if(expression.getNodeSetDefinition() != null && !currentUri.isEmpty)
5      relevantUris = SPARQLQuery.getRelations(currentUri, getQuery(expression.getNodeSetDefinition()))
6    else
7      relevantUris = SparqlQuery.getAllModelElements()
8    for(String uri : relevantUris)
9      result += evaluate(expression.getValue(), uri)
10   return result;
11 Float caseMult :
12   Float result = 0;
13   if(expression.getNodeSetDefinition() != null && !currentUri.isEmpty)
14     relevantUris = SPARQLQuery.getRelations(currentUri, getQuery(expression.getNodeSetDefinition()))
15   else
16     relevantUris = SparqlQuery.getAllModelElements()
17   for(String uri : relevantUris)
18     result *= evaluate(expression.getValue(), uri)
19   return result;
20 Float caseAggregatedMetricReference : return evaluate(expression.getCalculation())
21 Float casePlus : return evaluate(expression.left()) + evaluate(expression.right())
22 Float caseMinus : return evaluate(expression.left()) − evaluate(expression.right())
23 Float caseMultiplication : return evaluate(expression.left()) * evaluate(expression.right())
24 Float caseDivision : return evaluate(expression.left()) / evaluate(expression.right())
```

Listing 5.33: Evaluation procedures for the non-atomic expressions of a calculation rule.

The expression provided within an `AggregatedMetricReference` is evaluated according to the presented procedure for metric execution (line 20). For `Plus`, `Minus`, `Multiplication` and `Division` the expression defining the left side and the expression defining the right side are evaluated and finally combined with each other according the provided type (lines 21 - 24).

5.5.7 Execution of performance analysis

For the realization of the performance analysis in A2F, the calculation rules provided by [JI09] are applied. The proposed formulas for workload, processing time, response time and utilization are implemented with several DFA propagation rules. In contrast to [JI09], these rules are implemented independently from a specific EA meta model.

The `ModelNode` is extended with four data-flow attributes: workload λ, processing time T, response time R and utilization U. The workload is calculated at every model node. Processing time and response time are only determined for elements having an attribute *service time*. The utilization is only calculated for element having an attribute *capacity*. Instead of propagating the values over relations with a concrete stereotype, the edge classes `provide`, `consumed by` and `located at` are used. Only for referencing the required properties, the concrete stereotypes of the current EA meta model are required. They are provided within the Arla analysis configuration for a performance analysis.

According to [JI09] the **workload** λ_a for an element a is defined as

$$\lambda_a = f_a + \sum_{i=1}^{d_a^+} n_{a,k_i} \lambda_{k_i}, \tag{5.3}$$

where f_a denotes the arrival frequency of the node a, d_a^+ its out-degree, and n_{a,k_i} the weight of the edge between a and its child k_i.

This definition has to be translated into a data-flow rule to calculate the attribute *workload* in the context of the class `ModelNode`. The attribute values are computed according to the rule presented in listing 5.34. Line 3 initializes the `workload` with the value of the property *arrivalFrequency* of the currently processed node. The UUID provided within the analysis configuration is used to retrieve the value. If no value is assigned, the default value 0 is used. The loop in lines 5 ff. iterates over all outgoing edges. The result is updated according to the retrieved status of the `weighted_outgoing_workload` attribute. This corresponds to the sum statement within the formula and is illustrated in figure 5.14.

```
1  @Rule(modelClass=ModelNode, attribute=workload)
2  Object node_workload (ModelNode node){
3    Float workload = node.getPropertyValue(
4        parameters.getArrivalFrequencyUUID(), 0f)
5    for(ModelEdge edge : node.getOutgoing()){
6      workload += edge.getStatus(ATTRIBUTE.
           WEIGHTED_OUTGOING_WORKLOAD)
7    }
8    for(ModelEdge edge : node.getIncoming()){
9      workload += edge.getStatus(ATTRIBUTE.
           WEIGHTED_INCOMING_WORKLOAD)
10   }
11   return workload;
12 }
```

Listing 5.34: Propagation rule for workload calculation.

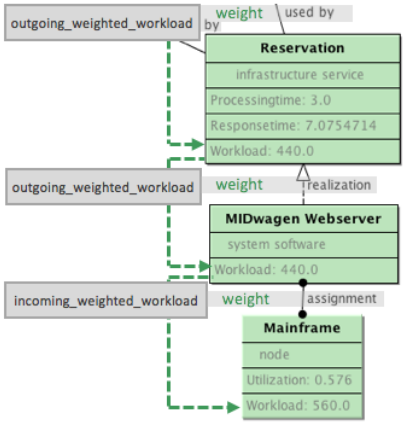


Figure 5.14: Example for workload calculation.

In contrast to [JI09] we also consider incoming relations for the workload (lines 8ff), since we observed that resources, for example a node or a device, are also connected with an incoming localization relationship. Following, only the `located at` class is extended with the respective `weighted_incoming_workload` attribute (see listing 5.35). In this rule, the weight of the edge is multiplied with the retrieved workload status from its source node (line 4). The UUID of the property representing the `weight`, is provided within the parameters. If no weight is specified, the default value 1 is used. Since *workload* is a data-flow attribute, the invocation of *getStatus()* informs the solver that a recursive fixed-point evaluation is required at this point.

```
1  @Rule(modelClass={LocalizedAt}, attribute=weighted_outgoing_workload)
2  Object incoming_propagate_workflow (ModelEdge edge){
3    Float weight = edge.getPropertyValue(parameters.getWeightUUID(), 1f)
4    return weight * edge.getSource().getStatus(ATTRIBUTE.WORKLOAD)
5  }
6  @Rule(modelClass={Provide, ConsumedBy}, attribute=weighted_outgoing_workload)
7  Object outgoing_propagate_workflow (ModelEdge edge){
8    Float weight = edge.getPropertyValue(parameters.getWeightUUID(), 1f)
9    return weight * edge.getTarget().getStatus(ATTRIBUTE.WORKLOAD)
10 }
11 @Rule(modelClass=ModelEdge, attribute={weighted_incoming_workload,weighted_outgoing_workload)
12 Object no_propagate_workflow (ModelEdge edge){
13   return 0f;
14 }
```

Listing 5.35: Propagation rule for workload calculation.

For the classes `provide` and `consumed by` the `outgoing_propagate_workflow` rule is

used to determine the value of the attribute. The weight of the edge is multiplied with the value of the workload attribute at the target node. All other occurrences of this attribute, i.e. for incoming provide and consumed by edges or for behavioral dependent edges, are determined according to the `no_propagate_workflow` rule (lines 11 - 13).

The **utilization** U_r of a resource r is defined as:

$$U_r = \frac{1}{C_r} \sum_{i=1}^{d_r} \lambda_{k_i} T_{k_i}, \tag{5.4}$$

with C_r the capacity of the resource, d_r the number of assigned behavior elements k_i and T_{k_i} the processing time of k_i.

The corresponding DFA rule is provided in listing 5.36. The rule calculates the value for the `utilization` attribute of `ModelNodes`.

```
1  @Rule(modelClass=ModelNode, attribute=utilization)
2  Object node_utilization (ModelNode node){
3    Float capacity = node.getPropertyValue(
4        parameters.getCapacityUUID(), 0f)
5    if(capacity <= 0) return -1f
6    Float sumProcessingTime = 0f
7    for(ModelEdge edge : node.getOutgoing()){
8      sumProcessingTime += edge.getStatus(ATTRIBUTE.
           OUTGOING_PROCESSINGTIME)
9    }
10   for(ModelEdge edge : node.getIncoming()){
11     sumProcessingTime += edge.getStatus(ATTRIBUTE.
           INCOMING_PROCESSINGTIME)
12   }
13   return sumProcessingTime/capacity
14 }
```

Listing 5.36: Propagation rule for utilization calculation.

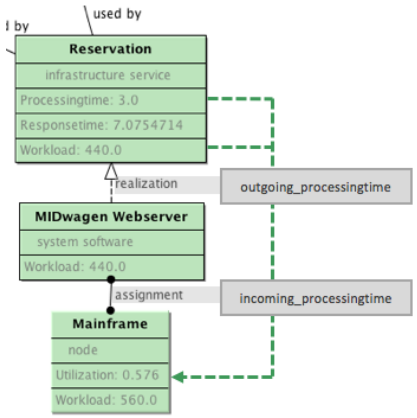


Figure 5.15: Example for utilization calculation.

The value of the capacity property is also identified with the provided UUID in the analysis parameters (lines 3, 4). If no capacity value is given, no utilization will be calculated (line 5). Otherwise, the processing times from assigned behavior elements which are retrieved with the `outgoinging_processingtime` attribute (lines 7, 8) and the `incoming_processingtime` attribute (lines 10, 11) are added up and finally divided by the capacity (line 13).

The `outgoing_processingtime` is calculated for outgoing relations of the class `provide`. The `incoming_processingtime` for incoming `located at` relations. In the first case, the attribute value is determined as the processing time multiplied with the workload of the target element of the `provide` relation. In the second case, the processing time is the sum of all processing times provided at the incoming and outgoing relations of the source of this edge.

The calculation of the utilization is illustrated in figure 5.15. In the figure the relations required to determine the utilization for the *Mainframe* are depicted. The required processing time of the service is propagated in outgoing direction over the *realization* edge and then in incoming direction over the *assignment* edge.

The recursive definition of the **processing time** T_a for a behavioral element a is:

$$T_a = S_a + \sum_{i=1}^{d_a^-} n_{k_i,a} R_{k_i}, \tag{5.5}$$

where S_a is the service time of a, d_a^- is its in degree, k_i is the parent of a and R_{k_i} is the response time of k_i.

The implementation of this equation as data-flow propagation rule is presented in listing 5.37. The processing time is only determined for elements having a valid service time property (lines 5). The service time property is determined according to the given UUID within the parameters. In the following, all incoming edges are processed and the available response time attributes are added to the service time. This provides the final processing time of the current model node.

The attribute `incoming_responsetime` is defined for `provide` and `consumed by` relations. Within the propagation rule for `provide` relations, the attribute is requested transitively for the incoming relations of the source node. All retrieved values are finally added up to provide the processing time. At `consumed by` relations, the processing time provided at the source node is retrieved and multiplied with the weight of the current edge. This is illustrated in figure 5.16, where the required relations are shown to determine the processing time of the *Reservation Management*. This is in specific the response time of the utilized infrastructure service *Reservation*. The response time of this service is propagated over the *used by* and the *realization* relation to the current node.

```
1  @Rule(modelClass=ModelNode, attribute=processingtime)
2  Object node_processingtime (ModelNode node){
3    Float servicetime = node.getPropertyValue(
4        parameters.getServiceTimeUUID(), -1f)
5    if(servicetime < 0) return -1f
6    Float processingtime = servicetime
7    for(ModelEdge edge : node.getIncoming()){
8      processingtime += edge.getStatus(ATTRIBUTE.
           INCOMING_RESPONSETIME)
9    }
10   return processingtime
11 }
```

Listing 5.37: Propagation rule for processing time calculation.

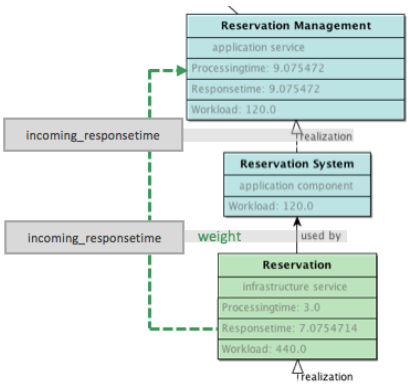


Figure 5.16: Example for processing time calculation.

According to [JI09] the **response time** R_a for an element a is defined as:

$$R_a = \frac{T_a}{1 - U_{r_a}}, \tag{5.6}$$

where r_a denotes the realizing or assigned resource of a.

The response time can be approximated with different queuing models. In the definition above the M/M/1 queuing model is used which is in most cases accurate enough on the architecture level [JI09]. For this approximation it must be assumed that a processing unit is only assigned to one resource.

The respective data-flow rule is presented in listing 5.38. To determine the response time

of a model node, its processing time as well as the utilization of the assigned resource is required. The current value for the processing time attribute is requested in line 3. If no valid processing time is provided, no utilization is calculated (line 4). Otherwise, all incoming and outgoing relations are processed to identify the assigned resource of the current element. If a valid utilization is retrieved with the `status` request, the utilization is set to this value (line 7 and 9). Finally, in line 10, the response time is determined according to the previously presented equation.

```
1  @Rule(modelClass=ModelNode, attribute=responsetime)
2  Object node_responsetime (ModelNode node){
3    Float processingtime = node.getStatus(ATTRIBUTE.
         PROCESSINGTIME)
4    if(processingtime < 0) return -1f
5    Float utilization = 0f
6    for(ModelEdge edge : node.getIncoming())
7      updateUtilization(edge.getStatus(ATTRIBUTE.
           INCOMING_UTILIZATION))
8    for(ModelEdge edge : node.getOutgoing())
9      updateUtilization(edge.getStatus(ATTRIBUTE.
           OUTGOING_UTILIZATION))
10   return processingtime/ (1-utilization)
11 }
```

Listing 5.38: Propagation rule for response time calculation.

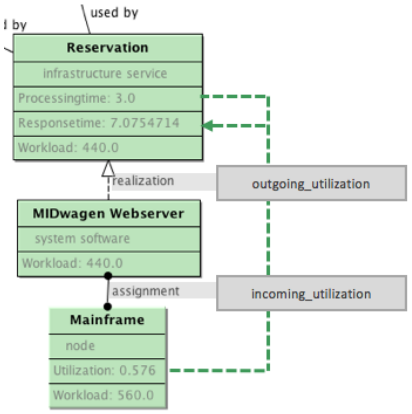


Figure 5.17: Example for response time calculation.

A resource can either be related to a processing unit with an incoming `provide` relation or with an outgoing `located at`. If the source element of an incoming `provide` relation has no valid utilization attribute, the request for a utilization is forwarded to the incoming and outgoing elements of the source element. In figure 5.17 the calculation of the response time for the *Reservation* service is provided. Based on its own processing time and the utilization of its realizing resource, the *Mainframe*, the value is calculated.

5.5.8 Execution of gap analysis

The gap analysis enables the comparison of two different models. Its configuration allows two different execution options: `Differences` and `SuccessorProposals`. With the `Differences` options, the current and target model are compared to each other to determine new elements, affected elements, unaffected elements and deletion candidates. These element attributes are determined based on the three sets proposed in [DB14]:

- `onlyCurrent` := $\{x \mid x \in current \wedge x \notin target\}$
- `onlyTarget` := $\{x \mid x \notin current \wedge x \in target\}$
- `currentAndTarget` := $\{x \mid x \in current \wedge x \in target\}$

In A2F these sets are determined with SPARQL queries. For example, the set of elements from the current architecture that occur also in the target architecture is determined with the SPARQL query in listing 5.39. The primary named graph to execute the query is the one provided within the `currentModelUri` (line 2). To be part of the result, an element must be in the current model (line 4) and there must be an element within the target model, identified with the `targetModelUri` that has the same UUID. Hence, a sub query is defined which provides the UUIDs of the target model (lines 5 - 8). In the final filter statement, the elements of the current model are restricted to those that have a pendant

in the target (line 9).

```
1  SELECT ?resourceURI
2  FROM <currentModelUri>
3  WHERE {
4    ?model  gmm:elements  ?resourceURI.
5    GRAPH <targetModelUri> {
6      ?targetModel     gmm:elements ?targetResource.
7      ?targetResource  gmm:uuid     ?targetResourceID.
8    }
9    FILTER EXISTS {?resource  gmm:uuid  ?targetResourceID.}
10 }
```

Listing 5.39: SPARQL query to determine affected elements within the current architecture.

The affected elements of the target model are determined accordingly. The elements which occur only in the current or only in the target architecture are determined with a similar query. Instead of filtering for the existence of an element, a filter for the non-existence is used.

Depending on the set an element belongs to, the resulting planning status is defined according to the following rules:

1. $x \in onlyCurrent \rightarrow$ `unaffected`
2. $x \in onlyTarget \rightarrow$ `new`
3. $x \in currentAndTarget \rightarrow$ `affected`

For each element assigned to the `unaffected` attribute, a further evaluation is performed to support the identification of deleted elements. A deletion candidate is identified as an element that has a relation to an affected element in the current architecture but does not occur within the target architecture. An `ASK` query is executed and evaluated to true, if such a relation exists for an unaffected element. The query is provided in listing A.1 in the appendix. In this case the element will be assigned with the planning status attribute `deletion candidate`. The strength of the dependency to affected elements can be expressed as metric (using the metric calculation within A2F) and provides further information to the architect. But finally, it is the task of the architect to decide about the planning status of the element, i.e. if it is a `deleted` one or an `unaffected` one.

The determined planning status attributes are finally captured within an `AttributeValueResultMap`. This map contains an entry for each element of the current and target architecture with an `AttributeValueResult` representing the determined attribute `new`, `affected`, `unaffected` or `deletion candidate`.

If the `SuccessorProposals` are evaluated, for each new element which occurs only in the target architecture, a set of potential predecessors is calculated. Three different SPARQL queries are executed to provide this set. The first one provides all elements of the current architecture that have the same name and same type as the target element and do not have a pendant in the target architecture (see listing A.2 in the appendix). The other two ones identify elements as potential predecessor that have at least one relation in common with the target element. This means, where the source, respective target of the relation, are the same according to the UUID. The queries are provided in the appendix in listings A.2 and A.3. Again, the architect has to decide about a successor relationship, an automatic conclusion cannot be made.

The retrieved proposals are finally provided as an `IDSetResultMap`. For each `new` element in the target architecture an entry is created within the map. The stored result value

for this entry consists of an `IDSetResult`. This set contains all URIs of the calculated predecessors.

The information about the dependencies between the current and the target model is stored in a so-called transformation model [AG10a]. In our method, the information is expressed with the planning status attribute value as well as manually defined successor dependencies in the case of a replacement. For the transformation model, a named graph is created in the triple store. Therewith, the status of the elements is accessible for further analyses.

Similarity measures can be used to evaluate successor proposals between elements of the current and target architecture. The similarity measure for two elements e_c, from the current architecture, and e_t, from the target architecture, is calculated as:

$$Sim(e_c, e_t) = \#sharedRelationships(e_c, e_t)/\#Relationships(e_c) \tag{5.7}$$

A shared relationship between e_c and e_t exists, if there is a relationship (e_c, r_c, e'_c) in the current architecture and a relationship (e_t, r_t, e'_t) in the target architecture and e'_c and e'_t are the same elements (identified by UUID) or are related to each other with a successor relationship. The relations r_c and r_t must have the same relation type rt. The measure is 0, if there are no shared relationships and 1 if all relationships of e_c are shared. The higher the value, the higher the probability of a successor dependency.

5.5.9 Execution of adapted analysis

An `AdaptedAnalysis` is executed through evaluating the respective generated specific analysis definition. During analysis definition the specific analysis definition is generated from the information provided within the `AdaptedAnalysis` and the referenced template definition (see section 5.4). The generated definition is then processed according to the execution procedures presented in these sections. The retrieved result constitutes the result of the `AdaptedAnalysis`.

5.5.10 Execution of custom analysis

The custom analysis configurations of Arla, i.e. the `SpecificDFAConfiguration` and the `CustomQuery`, are evaluated with the DFA solver respectively through executing the provided SPARQL query.

Within a `SpecificDFAConfiguration` the location of the respective DFA analysis configuration is provided. This file contains the required information to locate the propagation rules and enables the triggering of the requested analysis strategy. The returned DFA result is converted into an `AttributeValueResultMap`. For each element the determined attribute instances are stored within the map.

A further possibility to extend the scope of Arla is the explicit definition of a SPARQL query(`CustomQuery` configuration). This requires deep knowledge of the utilized GMM vocabulary, but it provides access to the full expressive power of SPARQL. Depending on the analysis style an `AggregatedResult` or an `ElementResult` is created. The provided result type is used to determine the type of the result value.

```
1 Analysis ACustomSPARQLQuery {
2   "Number of instances of a meta model edge connection"
3   as Element Metric
```

```
4    defined with SPARQL Query
5      "SELECT ?stereotype (COUNT (?edge) as ?nrInstances)
6       FROM <http://rentalcarmodel>
7       WHERE { ?model  model:edges       ?edge.
8               ?edge   model:stereotype  ?stereotype
9       }"
10 }
```

Listing 5.40: Example for a custom query analysis definition.

For example, the custom query provided in listing 5.40 describes an `Element Metric`. The metric determines the number of instances for each `MetaModelEdgeConnection`. According to the analysis configuration, the query result is provided as `ElementResult` with the result value type `AttributeValueResultMap`. The first variable within the `SELECT` statement is interpreted as URI which is used as key for the result map. All other variables, in this case only one, are interpreted as the results. Within the example, an `AttributeValueResult` is created with a `NumericResult` as value.

If an `Aggregate Modelelementset` is defined, the first row of the query result will be interpreted as URIs, identifying the elements of the set. All other result types, combined with an `Aggregate` analysis style, are converted into an `AttributeValueResult`. Therefore, only the first row is processed and for each column a result value is created using the variable name as identifier.

An `Element Modelelementset` is converted into an `IDSetResultMap`. The first column of the SPARQL result is interpreted as key of the map, all other columns as the entries of the respective `IDResultSet`. All other analysis configurations with the `Element` style are interpreted as `AttributeValueResultMap`. Again, the first column is interpreted as key of the map. The following columns are interpreted as the result values using the variable name as identifier. The used subclass for the `Result` is selected according to the provided result type in the analysis configuration, i.e. a `Modelelement` will be converted into an `IDResult`, a `Metric` into a `NumericResult` and a `Boolean` into a `BooleanResult`. Default case is the `StringResult`.

5.5.11 Execution of composed analysis

In the A2F, analysis composition can be performed in two different ways. Either the two composed analyses are executed subsequently. Thereby, the second one relies on the result of the first one. In the other case, the two analyses can be executed independently from each other and the results are combined afterwards.

Successive analysis execution

A successive execution can be defined with a `ApplyRule` and a `ApplyEachRule` in Arla. With the `ApplyRule` the result provided in the first analysis is used within the second analysis. This can be a set of elements, where the following analysis is only executed on this part of the architecture. Or the determined attribute value in the first analysis is utilized within the second one. In both cases, the first result is stored in a temporary named graph. The URI of the named graph is added to the relevant model URIs for the second analysis execution to be able to access the information. The result of the second analysis is the final result of the analysis composition. The specific procedures to persist the single result types are provided in the following.

If the first analysis provides a set of model elements, i.e. a `IDSetResult`, `PathResult`,

PathResultMap or IDSetResultMap, the provided result is inverted. This means that the set of model elements is determined that do not occur within the result. For each element in this set a *visibility property* is added to the named graph with value false. If the second analysis is executed with SPARQL queries, a validation pattern ensures that only the desired elements are considered. Listing 5.41 presents the NOT EXISTS statement that is added to SPARQL queries to cope with the restriction to a specific part of the architecture.

```
1  SELECT ?value
2  FROM <modelUri>
3  FROM <analysisResultUri>
4  WHERE {
5    ...
6    NOT EXISTS {
7      ?value        model:properties  ?property .
8      ?property     model:value       "false" ;
9      model:stereotype  ?stereotype .
10     ?stereotype   gmm:name          "visibility"}
11 }
```

Listing 5.41: Extension for visibility restriction wihtin SPARQL queries.

The EA model to be queried is provided within the first FROM statement in line 2. In line 3 the named graph containing the result of the first analysis is specified. In the WHERE clause, first the statements as required for the second analysis are provided (line 5). Afterwards an additional condition using a NOT EXISTS statement is defined, to restrict the resulting elements to those determined in the first analysis. Hence, only elements are included, where no visibility property with the value false exists.

If the second analysis is executed with the data flow approach, a validation step within the propagation rules ensures the restriction to the provided element of the first analysis. Within each propagation rule, used to determine the attribute instance at a ModelNode, the method validateNode(modelNode) is requested. It evaluates to false, if the current node should be excluded from the analysis. In this case no result is provided for the node. The validation is performed only for model nodes to ensure that no important relations are excluded during analysis. If a relation should not be considered, this is indirectly captured with the context-sensitive propagation of the attribute values.

If the result of the first analysis is an AttributeValueResultMap also a new property is created within the temporary model. This property is used to annotate the model elements with the result values in the result map. The result of the first analysis is now accessible like a normal model property within the second analysis. For example, we consider the node set definition: having property property:"Impact" [""] with value "MODIFICATION". This definition is applied to the result of an impact analysis and evaluated with the query provided in listing 5.42.

```
1  SELECT  ?value
2  FROM <http://rentalcar>
3  FROM <http://temp/result/Impact>
4  WHERE {
5    ?model            model:elements    ?value .
6    ?value            model:properties  ?nodeProperty10 .
7    ?nodeProperty10   model:stereotype  ?property10 .
8    ?property10       gmm:name          "Impact" .
9    ?nodeProperty10   model:value       ?propertyValue10
10   FILTER ( ?propertyValue10 = "MODIFICATION" )
11 }
```

Listing 5.42: Querying a temporary analysis result with SPARQL.

The result type AttributeValueResult of the first analysis cannot be utilized within a second analysis according to the ApplyRule. Referencing an aggregated metric within another metric can be done within the calculation rule.

With the ApplyEachRule in Arla the selection of model elements, is updated according to the elements provided in the result of the first analysis. The input parameter *selected model elements* is important for analyses like the scope analysis, the impact analysis, the path analysis or an element metric. Within the scope and path analysis the selected elements are interpreted as start elements. For impact analysis, these elements represent the ones that are affected by an event. And within executing an element metric, the selected elements are those elements, the metric is calculated for.

Thus, for the execution of an ApplyEachRule the first analysis is executed and the provided result is converted into a set of element URIs. This can only be done for an IDSetResult, a PathResult, a PathResultMap and an IDSetResultMap. For all other result types, the ApplyEachRule is not implemented. The set of selected elements for the second analysis is updated according to the result of the first analysis. Then, the second analysis is triggered and its result is also the result of the composed analysis.

Result combination

A combination of two results can be defined with a CombineRule and a SpecificCombineRule in Arla. In both cases, the two referenced analyses are executed independently from each other and the retrieved results are combined afterwards. Thereby, the SpecificCombineRule provides an operator which defines the procedure for combination. The operator can be an intersection, an union or a diff and is only applicable to results providing a set of elements. For those results the CombineRule follows the semantics of the union operator.

The utilized combination strategies for evaluating the CombineRule are presented in table 5.5. Combinations that are not presented in the table lead to an empty result set, since there are no combination strategies available. For example, an AttributeValueResult providing a metric cannot be combined with an IDSetResult, i.e. a set of model elements.

Combining two results, *result1* and *result2*, of the same type is done straightforwardly. Two IDSetResults, two PathResults or two AttributeValueResults are simply merged with each other. For example, an IDSetResult consists of a set of IDResults. To retrieve the final result of the composition, a new IDSetResult is created and all entries from *result1* and *result2* are added (row 1 in the table).

For the combination of two ResultMaps, also a new result of this type is created. Within the first step, the (key, value) entries of the first result are copied to the final result. In the second step, the entries of the second analysis result map are processed. The results provided for each element are added to the respective existing entry in the final result (see row 4-6 in the table).

Further combination strategies are implemented for the composition of results that can be converted into set of model elements, i.e. for IDSetResult, PathResult, PathResultMap and IDSetResultMap. These specific strategies are presented in rows 7-12 at the bottom of the table.

The SpecificCombineRule is only implemented for result types that can be converted into a set of elements. Following a PathResult, a PathResultMap and a IDSetResultMap are

Table 5.5: Result combination strategies.

result types	*combination strategy*
IDSetResult result1 IDSetResult result2	result = new <*Type*>Result() result.addAll(result1) result.addAll(result2)
PathResult result1 PathResult result2	
AttributeValueResult result1 AttributeValueResult result2	
IDSetResultMap result1 IDSetResultMap result2	result = new <*Type*>ResultMap() result.addAll(result1) result2.forEach[(uri, values) -> result.add(uri, values)]
PathResultMap result1 PathResultMap result2	
AttributeValueResultMap result1 AttributeValueResultMap result2	
IDSetResult result1 PathResult result2	result = new IDSetResult() result.addAll(result1) result.addAll(result2.getAllNodes())
IDSetResult result1 PathResultMap result2	result = new IDSetResult() result.addAll(result1) result2.forEach[(uri, pathResult) -> result.addAll(pathResult.getAllNodes())]
IDSetResult result1 IDSetResultMap result2	result = new IDSetResult() result.addAll(result1) result2.forEach[(uri, setResult) -> result.addAll(setResult)]
PathResult result1 IDSetResultMap result2	result = new IDSetResult() result.addAll(result1.getAllNodes())) result2.forEach[(uri, setResult) -> result.addAll(setResult)]
IDSetResultMap result1 PathResultMap result2	result = new IDSetResultMap() result.addAll(result1) result2.forEach[(uri, pathResult) -> result.(uri, pathResult.getAllNodes())]
PathResult result1 PathResultMap result2	result = new PathResult() result.addAll(result1) result2.forEach[(uri, pathResult) -> result.addAll(pathResult)]

converted into an `IDSetResult`. These two sets of elements are finally combined according to the provided operator. Applying the `union` operator returns a result, where all elements of the first and the second result are contained. With the `intersection` operator only those elements are within the final result that occur in both analysis results. And finally, the `diff` operator returns those elements that are within the result of the first analysis and not in the result of the second one.

5.6 Related Work

The existing analysis approaches within current literature use a large plethora of different techniques. In current literature techniques like XML [dBBG+05], SPARQL [SKR13b], extended influence diagrams [JLNS07a], probabilistic relational models [BUF+11], p-OCL [JUB+13] or architecture theory diagrams [JNL07] have been proposed for the definition of analyses. These techniques are dependent on the underlying meta model or schema, an adaption of defined analyses to another context or due to changes requires much effort. Analysis approaches within current literature that try to cover different analysis types are rare.

As shown in [Rau15, RLB16] none of technical categories for EA analyses captures more than six functional categories. This supports the statement that current approaches that try to cover several analysis types and thus, several functional categories are rare. The approaches in current literature are typically isolated ones that are not related to each other. Further challenges that are given only little attention are recursive analysis definitions. Cyclic dependencies in the models as well as incomplete models are sparsely considered. Only a few approaches (e.g. [KA09]) deal with indirect relationships in EA models.

In [JUB+13] the authors propose a multi-attribute framework for EA analysis using a probabilistic extension of OCL. With this approach the authors consider the calculation of dependencies between the properties as well as uncertainties regarding the existence of elements and relationships. The value of the properties is described with probability distributions as well as OCL expressions [JUB+13]. The probabilistic inferencing approach lacks procedures of how to deal with cyclic dependencies as well as it is restricted to the expressiveness of OCL.

Naranjo et al. [NSV14] propose the PRIMROSe framework for visual analysis of EA models. This is a graph based, modular approach to compose predefined analysis functions and create sound visualizations by utilizing selectors and decorators. The architect can define analysis chains, although the scope of the analysis functions is restricted to those that are pre-defined. Analysis functions can be specified using plain java code. The framework does not provide initial support for more advanced analysis techniques.

Frank et al. [FHK09] propose a domain specific language, ScoreML, for the definition of indicator systems. The indicators can be used for dashboards in the EAM context. An indicator can be a property of an element like costs, an aggregation of those properties or a snapshot of its value within a specific time, i.e. average monthly costs. The indicators within ScoreML are extended with a reference the relevant instance information for their evaluation. The DSL corresponds to the functionality of `Metrics` within the A2F. Further analysis types cannot be captured with this approach.

Current EA modeling tools provide extensive analysis support. Limitations are caused by to the modeling approach, the supported meta models and the technical analysis capabilities [NSV15]. Such capabilities can for example only include conformity checks or the generation of predefined views. Some EA tools provide also a possibility to query the model either by providing a DSL or by integrating for example a SQL interface. EA tools provide currently two major approaches to enterprise architecture analysis: Either they are shipped with a predefined and static meta model (e.g. LeanIX [Lea19], iteraplan [ite19]). Based on the meta model they provide several analysis techniques out of the box. In the other case, tools allow the customization of the meta model (e.g. Alfabet [Sof19b]), but vendor support is required for this task. The analysis functions of such a tool typically also

have to be adapted which is associated with a high effort. To support further analyses, they provide an interface to a query language. This is often an interface to execute SQL statements, however this requires knowledge about the utilized data structure of the tool within the database. Iteraplan for example proposes its own query language [ite18].

A tool survey among 213 participants identifies the functionalities of EA tools that are rated as good by the participants [HHKR16]. Only about one quarter rates the support of their EAM tool for the definition and usage of analysis patters, the stakeholder-dependent creation of views and the support for planning scenarios as good. Good support for complex calculations, e.g. for costs, is only mentioned by 20% of the participants.

5.7 Conclusion

In this section we presented the Architecture Analysis Framework A2F which provides an execution framework for EA analysis. The A2F interprets an EA model as stereotyped graph and relies on the proposed metamodel GMM (chapter 3) to enable its generic applicability. The analyses are specified with the Architecture Analysis Language Arla (chapter 4) which allows the specification of different analysis types and provides a uniform interface to the user.

The GMM enables a tool and EA meta model independent execution environment for the analyses. For executing the analyses, we utilize a combination of SPARQL and data-flow based analysis. Both approaches are successfully applied in large application scenarios and thus provide a scalable technical foundation for our analysis execution. The graph-based query language SPARQL provides features to answer structural questions and extract model parts. DFA is perfectly qualified to answer behavioral questions, execute recursive analysis definitions and deal with cyclic dependencies. It enables a forward and also backward traversing of the model. Combining both techniques provides support for the realization of different analysis types as well as the customization of the analyses by the users. Arla abstracts from all those technical details and provides the architect a simple interface to analysis activities. The concrete execution procedures are generated from the Arla definition at runtime within the A2F.

In the following, the design goals provided in section 5.1 are shortly discussed. The **generic applicability** of the framework is ensured through the utilization of the generic meta model GMM. The model storage within the triple store as well as the execution process relies on the concepts of the GMM and interprets the EA model as stereotyped graph. Thus, existing EA models, independently from the utilized meta model, can be captured and processed.

Implementing the analysis language Arla provides a **universal interface** to analysis activities. For all provided analysis classes within Arla, an execution procedure is defined. Following, the A2F supports scope, impact, path, gap and performance analysis as well as the execution of metrics. Composition of those analyses is also possible.

Custom analyses are supported through the individual analysis configurations specified by the user. Depending on the provided configuration, the concrete execution procedure is determined. If the predefined configuration possibilities are not sufficient, A2F as well as Arla enable the definition and configuration of native SPARQL queries or data-flow analyses.

Thereby, Arla enables a **declarative analysis definition**. The user declares what he is interested in using an analysis or template definition. The actual implementation is generated from the analysis configuration. For example, a view can be defined with constraints an element has to fulfill. How these constraints are evaluated, is not task of the user and done automatically.

The **re-use of analysis definitions** is supported with the Arla templates and the adaption of the concept of node and edge classes from the GMM. The later one enables the adaption of pre-defined analysis to a concrete EA model with minimal effort. The re-implementation of the previously static performance analysis [JI09] with DFA propagation rules allows now its generic application. The analysis can be now applied to EA model, independently from the utilized EA meta model. A model transformation step, as proposed by [JI09], is not necessary.

Through relying on the scalable techniques of DFA and SPARQL for analysis execution the **robustness regarding large models** can be ensured.

To enable **robustness towards incomplete models**, the analysis composition mechanism can be used to restrict the model to a valid part. Additionally, the SPARQL queries are designed to query the existence of certain conditions and not their absence. Within the DFA propagation rules, default handling mechanisms are provided when accessing property values. During the evaluation of a DFA rule system, only the present relationships are processed.

Round-trip-engineering is supported, since the original information about UUIDs and stereotypes is stored within the GMM model. This eases the process of writing analysis results back into the model. The uniform result format provides a coherent structure for further processing the results of the analyses.

Part III

Use Cases

6
Identification of Weak Points

Within this chapter the previously presented A2F is utilized to identify weak points in EAs. To generate meaningful results, it is important to ensure a high quality of the model. Weak points are an important asset for the subsequent planning steps. They point out optimization potentials and required actions. Within architecture planning, concrete scenarios are developed to deal with the weak points [Nie06].

Results of the master theses [Ose17] and [Eng17] are utilized for the implementation of the proposed approach for weak point assessment. The author of this thesis was the supervisor of both theses. Within [Ose17] requirements are identified that existing EA analysis approaches set for the meta model and model. The results of [Eng17], an identification and evaluation of microservice characteristics, are previously published in [ELBH18].

In the following we propose a method and tool support for the identification of weak points. To ensure high quality, a preliminary assessment of the model quality is proposed. Weak points are exemplary identified within the scenario of microservice architecture evaluation.

6.1 Overview of the Approach

The size of EA models makes it hard to identify weaknesses within the architecture without analysis support. The architect is no longer able to understand the architecture with its elements and relations at a whole. Employing analyses, optimization potentials within the architecture models can be identified [Han13]. These weak points provide the foundation for subsequent planning steps, in specific the development of planning scenarios [Nie06].

During analysis execution and result interpretation, the problem of incomplete models arises. Properties for elements or relations may not be specified or parts of the architecture are missing. Additionally, outdated data tampers the calculated results. In some cases, it also occurs that the EA model no longer conforms to the employed meta model. For example, the specified multiplicities are not met or mandatory properties are not specified. And finally, the defined modeling guidelines are often not fulfilled.

Without further assessment the suitability of the model for analysis activities cannot be ensured. A sufficient quality of the information in the EA model is important to receive reliable results. Analyzing a model of bad quality provides only results of bad quality. Making decisions based on such analysis results is not recommendable.

Nevertheless, it is important to use analyses to support decision-making. They support the monitoring of goal fulfillment as well as identify weaknesses of the architecture. Weak points can be assessed according to the organization-specific principles and goals. Often used principles are described within [Han13]. This is for example the principle '*Avoid Heterogeneity*'. Within an organization, a blue print is defined that specifies standard technologies. With respective analyses, those architecture elements can be identified that fail to comply with the blue print. Another principle regarding IT support is '*Avoid Redundancy*'. Analyses can be employed to identify duplicate IT support, i.e. business functions that rely on more than one information system or business data that is manipulated by more than one information system. Additionally, metrics can be used to quantify violations and monitor their improvement over time.

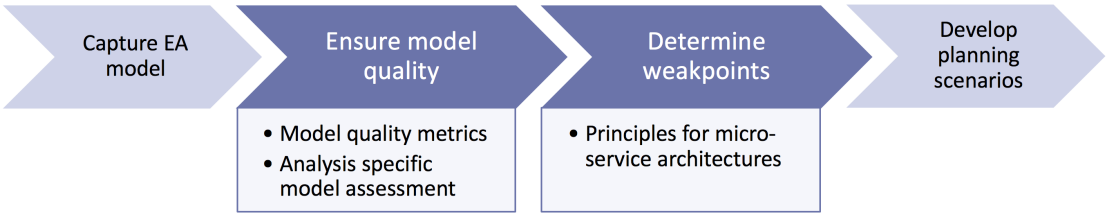


Figure 6.1: Overview of the process for weak point identification.

We propose the procedure presented in figure 6.1 for weak point analysis within EA models. The figure also depicts its integration into EAM activities. Beforehand the current EA model has to be captured. Subsequently, planning scenarios are developed to cope with the identified weak points.

A previous assessment of the model quality ensures a sufficient data quality for the actual weak point analysis. Within this assessment, the quality of the EA model is evaluated. On the one hand, this is done with modeling metrics, used to quantify constraints provided within the original meta model or within organization specific modeling guidelines. These metrics are developed using the A2F.

On the other hand, an analysis-specific assessment of the model quality is performed. The model is evaluated regarding the specific constructs required for analysis execution. Based on the analysis approaches within literature, the common requirements that analyses pose towards the quality of EA models are determined. The requirements are implemented with the A2F and their relevance for the analysis classes of Arla is outlined.

Depending on the results of the model quality assessment the architect can either:

1. Proceed with the weakness analysis,
2. Improve the model quality, or
3. Restrict the analysis to a part with sufficient quality.

After the assessment of the model quality, and a possible conduction of further actions, the weak points within the architecture are determined. Foundation for weak point analysis are the organization-specific architecture principles and goals. For weak point analysis, mainly metrics and scope analyses are applied. The first one provides a quantitative evaluation, whereas the second one retrieves the elements or the part of the architecture with a weak point.

In this thesis, we illustrate weak point analysis in the context of microservice architectures. Within current literature the typical characteristics of this architectural style are identified. Additionally, current challenges within projects in practice are determined in interviews. Based on these insights, principles regarding the design of microservices and microservice architectures are identified. Using the Goal Question Metric approach [BR88] in total nine metrics are derived from these principles. The metrics are implemented with the A2F using templates.

In the following the three steps, model quality metrics, analysis-specific model assessment and the analysis regarding the principles of microservice architectures are described in detail.

6.2 Model Quality Metrics

Within the case studies, we were faced with the problem of an insufficient data quality of the EA models. The analysis capabilities including view generation, enabled a deep insight into the models and uncovers several problems. In specific this were:

- Missing annotation of properties
- Missing relations between elements
- Contradicting information
- Violation of modeling principles

In order to increase the model quality and thus, also the analysis results, these flaws should be identified and solved. The procedure for their identification depends on the set up of the organization. If a formal meta model is provided the contained information about mandatory properties and multiplicities for relations can be employed. Further restrictions can be provided within model guidelines. They describe patterns, how a certain situation should be represented within the model.

Metrics can be used to quantify the restrictions from the EA meta model and from the modeling guidelines. They provide a mean to monitor the model quality over time. If the metric results indicate a bad model quality, further analyses can be applied to locate the causal elements.

We illustrate the identification and quantification of flaws by utilizing Arla to implement representative templates for each of the above-mentioned problems.

Missing annotation of properties

Within each model, there exist mandatory properties for elements. Typically, this is defined in a formal meta model (the original meta model of the source data). A common mandatory property for application components is the responsible person.

The following Arla template definition provides the ratio of elements from type `stereotype`, where a mandatory property with type `propertyType` is missing.

```
1  Template MissingPropertyMetric {
2    "Determines the ratio of elements with type 'stereotype' that have no property
         with type 'propertyType'."
3    as Aggregate Metric
4    defined with calculation rule ratio:
5      COUNT(nodeType:"stereotype" AND
6        (NOT (having property propertyType:"propertyType")))
7          /
8      COUNT(nodeType:"stereotype");
9  }
```

Listing 6.1: Template definition for a missing property annotation metric.

The concrete type of the element and the property can be specified during adaption of the template. For the RentalCar example this would be *application component* for the `stereotype` and `responsibility` for the mandatory `propertyType`. The analysis configuration of the respective adapted analysis is presented in listing 6.2.

```
1  adapt ModelQualityMetrics.MissingPropertyMetric {
2    map "stereotype"   to node:"application component" ["cc277f97-a4fd"]
3    map "propertyType" to property:"responsibility" ["644255cd-1e1e"]
4  }
```

Listing 6.2: Adapted analysis configuration for the missing property metric.

Missing relations between elements

Despite mandatory properties, there also exist mandatory incoming or outgoing relations between elements. This can be derived from the multiplicities within a formal meta model or from the modeling guidelines. If the lower bound for a relation is not zero, at least one relation has to be present. An example for such a mandatory relation would be a related server for each application component. A metric quantifying the ratio of elements with a missing mandatory relation within an EA model is provided in listing 6.3.

```
1 Template MissingRelationMetric {
2   "Determines the ratio of elements of a specific 'sourceType' that have a
        missing mandatory relation to an element of type 'targetType'."
3   as Aggregate Metric
4   defined with calculation rule ratio:
5   COUNT(nodeType:"sourceType" AND
6     (NOT (having relation to nodeType:"targetType")))
7       /
8   COUNT(nodeType:"sourceType");
9 }
```

Listing 6.3: Template definition for a missing property annotation metric.

This template definition provides the ratio of elements of a specific `sourceType` that have no relation to an element with a specific `targetType`. Thereby, source and target do not imply the direction of the relation. This might be an incoming as well as on outgoing one. Within the example provided above the `sourceType` would be mapped to *application component* and the `targetType` to *server*.

Contradicting information

Flaws within the EA model can also be caused by contradicting information. For example, if a service has two related application components, this is typically caused by a modeling fault. The realization relation between service and application is in most cases unambiguous. A composition of the template definitions in listing 6.4 provides a quantification of this issue. Within the first query, for each element of a specific `nodeType`, the number of incoming relations of the type `edgeType` is provided. Within the example, this would be the number of incoming *realization* relations for *application services*.

```
1  Template RelatedSourceElements {
2    "Provides the number of source elements for a specific edgeType."
3    as Element Metric
4    defined with calculation rule: COUNT (incoming edgeType:"edgeType");
5    for types (nodeType:"nodeType")
6  }
7  Template SeveralIncomingRelationsMetric{
8    "Provides the ratio of elements with more than one related source element."
9    as Aggregate Metric
10   defined with calculation rule ratio:
11   COUNT(nodeType:"nodeType" AND
12     having property propertyType:"RelatedSourceElements" with value (> 1))
13       /
14   COUNT(nodeType:"nodeType");
15 }
```

Listing 6.4: Template definition for quantifying unambiguous relations.

Within the second query, this result is utilized to provide the ratio of elements that have more than one incoming relation of this type. Therefore, within a `PropertyCondition`, the element metric result is referenced. If this value is greater than one, the element is

counted (line 12). Finally, the total amount is divided by the total number of elements of type nodeType (lines 6,7).

The two templates have to be composed according to the composition rule:

```
apply SeveralIncomingRelationsMetric on RelatedSourceElements
```

This enables the access to the result of RelatedSourceElements within the SeveralIncomingRelationsMetric template.

Violation of modeling principles

The last quality problem within EA models is the violation of organization-specific modeling principles. Examples for such modeling principles are that *business processes only access applications via the corresponding application services* or that *services always have an assigned application*. The degree of fulfillment of such modeling principles can also be monitored with metrics.

In listing 6.5 a metric is defined that quantifies the availability of a relation between an element of sourceType and an element of targetType. The metric is similar to the one provided in listing 6.3, quantifying missing relations between elements. But in this case the RelationCondition is not negated. Here as well, source and target do not imply the direction of the relation.

```
1 Template AvailableRelationMetric {
2   "Determines the ratio of elements of a specific sourceType that have an
        undesired relation to an element of type targetType."
3   as Aggregate Metric
4   defined with calculation rule ratio:
5   COUNT(nodeType:"sourceType" AND (having relation to nodeType:"targetType"))
6       /
7   COUNT(nodeType:"sourceType");
8 }
```

Listing 6.5: Template definition to measure the availability of a certain relation type.

Mapping the sourceType to *business process* and targetType to *application component* allows the quantification of the first guideline provided above. The metric determines in this case the ratio of business processes that directly access application components which is not desired.

The other example, i.e. *services always have an assigned application*, can be quantified through adapting the MissingRelationMetric from listing 6.3. The respective adapted analysis configuration is provided in listing 6.6.

```
1 adapt ModelQualityMetrics.MissingRelationMetric {
2   map "sourceType"  to node:"application service" ["2f4a3ad3-d698"]
3   map "targetType" to property:"application component" ["cc277f97-a4fd"]
4 }
```

Listing 6.6: Adapted analysis configuration to validate the service application assignment.

The sourceType is mapped to *application service* and the targetType is mapped to *application component*. Therewith the final metric provides the ratio of application services that have no related application component.

6.3 Analysis Specific Model Assessment

Despite the metrics proposed in the previous section to assess the model quality according to the formal meta model and the modeling guidelines, in this section the specific requirements for executing the analyses are considered. Therewith, it can be ensured that the model suits for the desired analysis approaches and their execution provides a meaningful result. Not only constraints for the meta model, e.g. the existence of a certain property, are considered, but also their usage within the model. That means, if the property is defined for all elements with valid values. This enables the assessment of the model towards the meaningfulness of the analysis execution. The technical executability is ensured through employing the A2F.

In the following the specific requirements of EA analyses towards the EA model are examined and their validation using SPARQL queries is provided.

6.3.1 Identification of analysis requirements

To identify respective requirements, one analysis approach for each of the technical dimension proposed presented in section 4.1 was chosen and considered in detail. An overview of the representative for each dimension is provided in table 6.1.

Table 6.1: Representative approach for each technical dimension.

Dimension	*Representative*	*Dimension*	*Representative*
Metrics and KPIs	[IOB06]	Matrix	[vSSBB10]
Bayesian network	[LJ08]	Design	[AS11]
Ontology	[SKR13a]	EID	[NJN07]
Weak points	[LLP+09]	PRM	[NFK+12]
Business entities	[DBLR+11]	AHP	[DA09]
Time evaluation	[JI09]	Tree	[NFK+12]
Lifecycle	[ABF+09]	Views	[FJdW97]
Social networks	[KMP11]	Structural	[BMNS09]
Comparison	[YSD06]		

Altogether, we identified five different categories of analysis requirements:

Availability of element types. Within an analysis definition references to certain element types can be made. For example, the system quality analysis presented in [NJN07] requires the element types *information system, source code, security service* and *staff*. If there are no instances for these element types within the current model, the execution of the analysis would provide no usable result.

Relations between elements. Analyses rely not only on element types, they also utilize the relations between them. For example, the performance analysis within [JI09] requires a realization between an internal behavior element and a service. This requirement is interpreted as an implication. This means that if there exists a service within the EA model, there also has to be a realization relationship to an internal behavior element. Despite the actual existence of such a relation, the multiplicities have to be considered, too. For example, the performance analysis can only be evaluated correctly if a processing unit has only one assigned resource [JI09].

Limitation of property values. An analysis approach utilizes the values of properties

assigned to the elements of the model. It is important that these values are within certain lower and upper bounds in the case of numerical values. Additionally, for enumeration values it must be ensured that the assigned values are from a finite domain, e.g. {*high, medium, low*}.

For example, if a resource within the performance analysis is assigned with zero capacity, this cannot be identified as missing value and thus the respective response time will be calculated. This results in a negative response time which does not provide a useful result.

Cyclic relations between properties. Especially analysis approaches relying on probabilistic techniques like Bayesian networks (e.g. [LJ08]) require relations between the properties. These relations determine the influence of the properties on each other. For example, the availability of a server has an influence on the availability of the assigned applications. It must be ensured that these relations exist, as well as the absence of a cyclic dependency.

Adherence to the analysis metamodel. Finally, each analysis approach is built upon its one meta model. It is important that the meta model of the EA model adheres to the analysis meta model, i.e. it is a correct instance of it. Otherwise, it would not be possible to execute the analysis.

6.3.2 Validation of analysis requirements

The five identified requirements are evaluated using SPARQL queries. Where possible, the queries provide those elements that violate the requirements of the respective analysis. Thus, the architect can immediately take further actions. This could be the improvement of the model. If this is not possible, the respective part from the model can be excluded for this analysis.

Availability of element types

Especially for analyses assigned the technical dimension *Views* it is important, whether the referenced stereotypes are utilized within the model. The query proposed in listing 6.7 provides all meta model nodes within the EA model, for which element instances exist.

```
1 SELECT ?stereotype
2 WHERE {
3   ?stereotype   rdf:type  gmm:MetaModelNode.
4   FILTER NOT EXISTS {
5       ?subject     rdf:type          gmm:ModelNode.
6       ?subject     gmm:stereotype    ?stereotype.
7   }
8 }
```

Listing 6.7: Query for determing element types without instances.

To validate an Arla analysis definition, the returned list has to be compared with the utilized `NodeReferences` of the analysis configuration. If one of the node references occurs also in the result list, the execution of the analysis should be reconsidered. Alternatively, this fact should be considered during result interpretation. A solution strategy in this case would be an adaption of the analysis definition, where the respective stereotype is replaced. A broader solution would be the extension of the model, for example with further information sources to include elements of the missing type.

Relations between elements

To assess the existence of a relation between two elements or to validate the multiplicities a SPARQL query utilizing the `COUNT` aggregator is employed. The query in listing 6.8 provides the elements for whom a specific outgoing relation does not exist. Despite the type of the relation, the stereotype of the source and target element can be restricted too. If these restrictions are not required, the line 4, line 9 or both lines can be omitted.

```
1  SELECT DISTINCT ?subject
2  WHERE {
3    ?subject  rdf:type              gmm:ModelNode;
4              gmm:stereotype        <source_stereotype>.
5    FILTER NOT EXISTS {
6        ?edge       gmm:source         ?subject;
7                    gmm:stereotype     ?edgeStereotype;
8                    gmm:target         ?target.
9        ?target     gmm:stereotype     <target_stereotype>.
10       <mm_edge>   gmm:connections    ?edgeStereotype.
11 }
```

Listing 6.8: Query to determine elements with a missing relation.

Due to the design of the A2F, there are no strict requirements for the existence of relations. The utilization of relation types, referenced within an `EdgeTypeReference`, can be validated with the above query. Since only the type is relevant, lines 4 and 9 are not required. For a `EdgeClassReference` lines 9 and 10 are replaced with the triple `?target rdf:type <class>`. Both node stereotype restrictions (source and target) are omitted as well. There may be further requirements for an analysis that cannot automatically be identified from the analysis definition. They have to be provided manually by the analysis designer.

The query proposed in listing 6.9 retrieves the elements that violate a multiplicity specification for outgoing relations. With this query only those relations are assessed that exist at least once within the model. The existence of relations, i.e. an assessment regarding the multiplicity zero can be done with the query in listing 6.8.

```
1  SELECT DISTINCT ?subject (COUNT (DISTINCT) ?target AS ?nrRel)
2  WHERE {
3    ?subject       rdf:type           gmm:ModelNode;
4                   gmm:stereotype     <source_stereotype>.
5    ?edge          gmm:source         ?subject;
6                   gmm:stereotype     ?edgeStereotype;
7                   gmm:target         ?target.
8    ?target        gmm:stereotype     <target_stereotype>.
9    <mm_edge>     gmm:connections    ?edgeStereotype.
10 }
11 GROUP BY ?subject
12 HAVING (?nrRel  comparative_operator  multiplicity)
```

Listing 6.9: Query to validate the multiplicities of outgoing relations.

To validate the multiplicity requirement, all outgoing relations of the type `mm_edge` for elements of the type `source_stereotype` are considered (lines 3-7). If desired, the target elements of the relations can also be restricted having a specific `target_stereotype` (lines 8). Finally, the number of target elements is counted (line 1) and filtered with the condition provided in line 12. For example, an upper limit of one, would be represented as `> 1`. Executing the query returns all elements that have more than one relation. Here as well, the stereotype limitations can be omitted as well as replaced with class references.

Both queries (listings 6.8 and 6.9) only consider outgoing relations. Incoming relations can be assessed through interchanging the source and target assignments.

Limitation of property values

Property values are assessed in three different ways. First, the existence of a value for a specific property type is determined. Additionally, the adherence to certain lower and upper limits or enumeration types is validated. And finally, duplicate annotations of the same property value are identified.

The query provided in listing 6.10 returns those elements, for whom no property of the required `property_stereotype` is provided or for whom no value is specified. The type of the element is restricted to the value `node_stereotype` in line 4 which can be omitted if not required.

```
1  SELECT ?subject
2  WHERE {
3    ?subject  rdf:type         gmm:ModelNode;
4              gmm:stereotype   <node_stereotype>;
5    FILTER NOT EXISTS {
6      ?subject      gmm:property     ?property.
7      ?property     gmm:stereotype   <property_stereotype>;
8                      gmm:value        ?value.
9    }
10 }
```

Listing 6.10: Query to verfiy the existence of a property value.

All property references used within Arla analysis definitions, can be evaluated with this query to ensure their existence. For a simple `PropertyTypeReference`, the stereotype restriction for the elements in line 4 is omitted. For the `TypedPropertyReference` the specified node type is used within the query.

A more detailed consideration of the property value can be performed with the query in listing 6.11, The query assesses the properties with type `property_stereotype` and verifies their value. Within the `FILTER` expression in line 8 the adherence to enumeration values is ensured. Each element, whose property value is not one of the defined enumeration values, is provided within the result of the query.

```
1  SELECT ?subject ?value
2    WHERE {
3    ?subject    rdf:type           gmm:ModelNode;
4                gmm:stereotype     <node_stereotype>;
5                gmm:property       ?property.
6    ?property   gmm:stereotype     <property_stereotype>;
7                gmm:value          ?value.
8    FILTER ( !( ?value = ... [OR ?value = ...] ) )
9  }
```

Listing 6.11: Query to verfiy the property value.

Numerical values can be evaluated using a comparative operator within the `FILTER` expression. For example, the requirement for non-negative values is evaluated with the expression `FILTER(?value < 0)`. In this case, the query provides all elements having a negative property value.

The required information for evaluating constraints on property values, cannot be extracted from the Arla analysis definitions. The analysis designer has to provide them manually for verification purposes.

Finally, within an RDF graph it is possible to assign the same property to an element twice. During analysis execution it is not clear which value to use. This issue can be identified with the query provided in listing 6.12.

```
1 SELECT DISTINCT ?subject ?propertyType
2 WHERE {
3   ?subject    rdf:type          gmm:ModelNode;
4               gmm:property      ?property1;
5               gmm:property      ?property2.
6   ?property1  gmm:stereotype   ?propertyType.
7   ?property2  gmm:stereotype   ?propertyType.
8   FILTER (?property1 != ?property2)
9 }
```

Listing 6.12: Query to identify multiply defined properties.

The query can be performed independently from the analysis definitions. It provides all elements that have two values for one property. A concrete specification of the property type is not necessary.

Cyclic relations between properties

Since we do not consider probabilistic analysis approaches within Arla, the requirements regarding property dependencies are not relevant.

Beside this issue, the existence of a cycle within property dependencies cannot be identified solely within SPARQL. A fix point calculation is required. A data-flow analysis can be utilized to determine the transitive closure for each element. If the current element is also within the transitive closure, a cycle is present. Alternatively, strongly connected components according to the property dependencies can be determined. Their presence points out a cyclic dependency.

Adherence to the analysis metamodel

This requirement is always met, when defining the analysis with Arla and having a representation of the EA model according to the GMM format. The A2F ensures the technical executability of the analysis definition on the EA model.

6.4 Assessment of Microservice Characteristics

Microservices are an emerging style for designing software architectures to overcome current issues with monoliths. This includes the difficulties of maintenance and system evolvement but also their limited scalability. The style is characterized by building an application through the composition of independent functional units, running its own process, and communicating through message passing [DGL+17]. The microservice architectural style enables the creation of scalable, reliable and agile software systems [HS17]. With an automatic deployment, they enable shorter product development cycles and serve the need for more flexible software systems.

Microservice systems are composed of up to hundreds or even thousands of services [GCF+17]. Complexity increases also due to the nature of a distributed systems and the typically high release frequency [BWZ17b]. Managing and monitoring those systems is essential [HS17, GCF+17, BWZ17b]. This includes the detection of failures and anomalies at runtime but also ensuring a suitable architecture design for example regarding consistency and fault tolerance [HS17].

Organizations employing microservice architectures require a "systematic understanding of maintainability as well as metrics to automatically quantify its degrees [of effectiveness and efficiency with which a software system can be modified to correct, improve, extend, or adapt it]" [BWZ17b]. Analyzing the performance of the system and understanding failures that occur during runtime, is essential to understand the system as a whole [AAE16]. Existing measurement approaches for service- or object-oriented systems are difficult to apply within microservice architectures [BWZ17b]. Since microservice systems can contain up to thousands of single services, the architectural models get very large and complex. If they are manually created, typical problems in practice are inconsistent and incomplete models. Often, information is missing or only a part of the architecture is represented.

In the following we identify metrics that are used to evaluate a microservice architecture and to identify weak points. First, an analysis of current literature as well as an assessment of five microservice projects was performed. Based on the characteristics and challenges, common principles for the design of an microservice-based architecture are identified. From these principles, we derived metrics for the identification weak points. Weak points denote those parts of the architecture, where problems may occur and thus require a detailed consideration. The implementation of the metrics is done with the A2F.

6.4.1 Characteristics of microservice architectures

A microservice system is a decentral system, composed of several small services that are independent from each other. The communication takes place with light-weight mechanisms i.e. messaging. Instead of a central orchestration, a decentral management like choreography is applied. Core principles of a microservice architecture are loose coupling and high cohesion [DGL+17, New15, Tho15, AMMN16]. Microservice architectures follow a domain-driven design, where each microservice is responsible for one bounded context [HS17, AMMN16, Eva03, FL14]. That means that a microservice provides only a limited amount of functionality serving specific business capabilities. Microservices are developed independently from each other i.e. they are independent regarding the utilized technologies and programming languages, the deployment and also the development team [HS17, FL14, APC+15, FML17]. A team is responsible for the full life cycle of a

microservice, from development to operation [DGL+17]. The build and release process is automated and enables continuous delivery [DGL+17, AMMN16, APC+15]. According to [AAE16] scalability, independence, maintainability (including changeability), deployment, health management and modularity (i.e. single responsibility) are the most mentioned attributes of microservice systems in literature.

During implementation of a microservice system, two major issues have to be considered: The first one addresses challenges due to the nature of distributed systems and the second one the context definition of a microservice. Finding the right granularity of microservices is essential, since badly designed microservices increase the communication complexity of the system [HS17, FL14, APC+15]. They also reduce the extensibility of the system and impede later refactoring and the overall management.

Since communication is essential in microservice-based systems, network communication, performance and complexity are important issues [DGL+17, AAE16, APC+15, FML17]. The system must be able to deal with latency and provides mechanisms for debugging, monitoring, auditing and forensic analysis. The architectural design has to support the openness of the system and has to deal with the heterogeneity [DGL+17, CDK11]. Additionally, fault handling and fault tolerance [AAE16, CDK11] as well as security issues [DGL+17] must be considered.

6.4.2 Challenges within microservice architectures

Within an exploratory study we analyzed the challenges in five different microservice projects. Table 6.2 presents the projects with their type, duration, size (in number of microservices MS) and utilized technologies. We conducted in-depth interviews with selected experts from each project.

Table 6.2: Characteristics of the assessed microservice projects.

Project	*Project type*	*Duration*	*Size*	*Technologies*
Pr1	Big data application	2.5 years	20-30 MS	Cassandra, Elastic, HiveMQ, Payara, PostgreSQL, Spark
Pr2	Cloud platform	1 year	3 MS	Akka, PostgreSQL, Spring Boot
Pr3	Cloud platform, Smart Home	6 months	4 MS	Docker, Kubernetes, RabbitMQ, Vert.x
Pr4	ETL project	6 weeks	2 MS	Docker, Kafka, Kubernetes, Openshift
Pr5	Information system	2 years	ca. 50 MS	AWS, consul, Docker, Feign-Client, MongoDB, RabbitMQ, Spring Boot

An interview constitutes of four parts: In the introduction the purpose of this study was presented. Afterwards the interview partner was asked to describe the project including her own role in the project. The first main part of the interview was about identifying challenges and problems within the project. This included also the reason for choosing a microservice architecture. Finally, an open part was left to discuss project-specific topics in detail that were not covered so far. Resulting topics were e.g. the size of microservices, rules for the decomposition into small services, use cases covering multiple

microservices and a refactoring affecting more than one microservice. For some project's asynchronous communication, data consistency, deployment and testing were addressed as well. Table 6.3 provides a summary of the identified challenges within the projects.

Table 6.3: Identified challenges within the microservice projects.

Challenge	*Projects*
Context definition for Microservices	Pr1, Pr5
Microservice-wide refactoring	Pr1, Pr5
Non-local changes	Pr1, Pr3
Cyclic dependencies	Pr1
Keeping an overview	Pr1, Pr4, Pr5
Monitoring and logging	Pr1, Pr4
Consistency	Pr4
Testing	Pr5

Especially in larger projects, context definition of microservices was mentioned as a major challenge (Pr1, Pr5). Questions in this context were '*What is a microservice?*', '*What size does it have?*' and '*Which tasks do the microservice implement?*'. With these questions as baseline, the problems arising through an unsuitable service decomposition were elaborated. Discussed consequences are an increasing communication complexity which leads to performance weaknesses and the difficult integration of new features which result in an increasing time-to-market. An unsuitable breakdown of microservices tends to have more dependencies and thus the developer has to understand greater parts of a system for the implementation of a new feature.

Another challenge is refactoring that affects more than one microservice (Pr1, Pr5). In contrast to monolithic systems, there is no tool support to perform effective refactoring of microservice systems. Additionally, the complexity increases through the utilization of JSON objects or similar formats at the interfaces.

Closely related to that aspect is the issue of non-local changes (Pr1, Pr3). Experiences in the projects have shown that changes made to one microservice may not be limited to this microservice. For example, changing interface parameters has effects on all microservices using that interface. Another reason why changes in one microservice may lead to changes on other services is because of changes in the utilized technologies. Experiences in the projects have shown that the adherence of homogeneity is advised for better maintainability, robustness and understandability of the overall system. Even if the independence regarding technology and programming languages is possible and a huge advantage, system-wide technology decisions should be made, and only if necessary deviations should be made. This is why changes affecting the technology of one microservice may also lead to changes in other microservices.

A challenge that addresses the design of microservice systems are cyclic dependencies (Pr1). These dependencies cannot be identified automatically and cause severe problems when deploying new versions of a service within a cycle. To prevent a system failure, the compatibility of the new version to all services in the cycle has to be ensured.

Keeping an overview of the system was a big challenge in three projects (Pr1, Pr4, Pr5). Especially large microservice systems are developed using agile development methods within different autonomous teams. Each team is responsible for its own microservices, carries out a refactoring and implements new functionality by its own. Experiences in the

projects showed that it is hard to not lose track of the system. The complexity of this task increases, since the dataflow and service dependencies cannot be statically analyzed like in monolithic systems.

Beside the above challenges, monitoring and logging (Pr1, Pr4), data consistency (Pr4) and testing (Pr5) were also mentioned in the interviews. Monitoring and logging address the challenge of finding, analyzing and solving failures. Testing is aggravated by the nature of a distributed system and the heavy usage of asynchronous communication, causing large problems concerning data consistency.

6.4.3 Evaluation criteria: Principles and metrics

Based on the literature and the results of the structured interviews we identified 10 principles for the design of microservice architectures. Table 6.4 provides an overview of them with their references to literature and the related projects.

Table 6.4: Principles for microservice architectures.

Nr	*Principle*	*Mentions*	*Corresponding metric*
P1	Small size of microservice	[DGL+17, New15, Tho15, AMMN16], Pr2, Pr5	M1: #(A)synchronous interfaces
P2	Single responsibility principle	[DGL+17, Tho15, Kil16], Pr1, Pr2	M1: #(A)synchronous interfaces M2: Security similarity M3: Distribution of call frequency
P3	Scalability	[AAE16, Kil16], Pr1, Pr5	M3: Distribution of call frequency M4: Distribution of data volume
P4	Loose coupling and high cohesion	[DGL+17, New15], Pr1, Pr3	M5: #(A)synchronous usages
P5	Domain-driven design	[Eva03, FL14],Pr1, Pr2, Pr3, Pr5	M5: #(A)synchronous usages M6: Longest synchronous call trace M7: Average size of messages
P6	Clarity of the system design	[AAE16, FL14, APC+15], Pr1	M5: #(A)synchronous usages
P7	Low network complexity	[DGL+17, FL14], Pr2	M5: #(A)synchronous usages
P8	Performance	[AAE16, APC+15, FML17], Pr1	M6: Longest synchronous call trace M7: Average size of messages
P9	Independence	[AAE16, HS17, FL14, DGL+17, AMMN16, APC+15, FML17], Pr1	M8: Utilization degree of technologies
P10	No cyclic dependencies	[Sha17], Pr1	M9: Elements in synchronous dependency cycles

To evaluate the principles, we derive metrics using the Goal Question Metric approach [BR88]. For metric definition, we also considered the proposed metrics by [BWZ17b] to measure quality attributes of a service-oriented system like coupling, cohesion and granularity. The metrics proposed in [ZNL17] were not practicable for our approach, since the required information cannot automatically be gathered from communication logs. Especially in-memory dependencies are not graspable.

The overall goal, according to the Goal Question Metric approach, is the adherence to the principles. A possible question in this context is for example '*Are the microservices of a*

small size?' (P1). The derived metric for the small size principle counts the number of synchronous and asynchronous interfaces of a service (M1). An (a)synchronous interface indicates a provided functionality of a service via an endpoint that can be consumed synchronously respectively asynchronously. The number of interfaces is an indicator for the amount of implemented functionality and can thus be used to measure the size of a microservice.

Due to the derivation of the implemented functionality, the metric can also be used for evaluating P2. The single responsibility principle depicts that one microservice should realize one task. Additionally, the similarity of security requirements can be assessed (M2). Different requirements at the implemented interfaces are a hint for the provisioning of different tasks. Finally, the distribution of the call frequency at interfaces (M3) can be considered. Therefore, the empirical variance of the call frequency at the interfaces of a service is considered. Having a large variance indicates that the number of interface calls varies widely and the microservice may implement more than one task.

A high diversity within the number of interface calls also hampers scalability (P3). Executing several instances of a service to deal with the high number of incoming requests, leads also to a duplication of the functionality behind the less requested interfaces. The same counts for the distribution of data volume at the provided interfaces of a service (M4).

To evaluate the principles loose coupling and high cohesion (P4), the number of synchronous and asynchronous usages (M5) can be used. This metric is calculated at service level and denotes the number of service calls, either via synchronous or asynchronous communication dependencies.

This metric is also used to assess the principle of domain-driven design (P5). If a system is built around business capabilities, the services own the majority of the required data by themselves. In contrast, a system built around business entities contains more dependencies since more data has to be exchanged. A further indicator that speaks against a domain-driven design is the lengths of synchronous call traces. M6 provides the longest synchronous call trace identified in the microservice system.

The principles P6 and P7, low network complexity and clarity of the system design, can be evaluated using the number of (a)synchronous usages (M5). A small amount of dependencies indicates a low complexity of the network, while it also supports the clarity of the whole system. The clarity of the system design encompasses the mentioned issue of manageability of the overall system.

To provide indicators for the system performance (P8), the longest synchronous call trace (M6) and the average size of asynchronous messages (M7) are proposed. Large messages may cause performance bottlenecks and the problem of network latency increases with a rising number of services within one call trace. Both factors have negative impact on the overall performance of an microservice system.

P9 describes the independence of the microservice in different aspects like technology, life cycles, deployment but also the development team. The independence degree can be quantified with the utilization degree of certain undesired technologies for communication and deployment (M8), e.g. the usage of Remote Method Invocation (RMI) for communication.

Finally, P10 deals with cyclic dependencies between the microservices. Overall design goal should be at least the absence of a synchronous dependency cycle. Therefore, the number

of elements that are part of a synchronous cycle are determined (M9). This metric should be zero in the best case.

6.4.4 Implementation with the A2F

The nine identified metrics in 6.4.3 are implemented using the A2F in order to enable their generic application. The provided Arla template definitions can easily adapted to a concrete EA model.

M1: #(A)synchronous interfaces

The number of implemented (a)synchronous interfaces for a service can be determined with the template definition in listing 6.13. The metric is calculated for each *service* (line 6) and provides the number of *interfaces* (line 4) that communicate in an *asynchronous* way (line 5). Only those interfaces are considered that are related to the services with a *provide* dependency (line 4).

```
1 Template AsynchronousInterfacesMetric{
2   "Description"
3   as Element Metric defined with calculation rule:
4     COUNT (connected edgeClass:Provide to (nodeType:"interface" AND
5       having property propertyType:"communication" with value "asynchronous"));
6   for types (nodeType:"service")
7 }
```

Listing 6.13: Template definition for metric M1.

To retrieve the number of *synchronous* interfaces, the value field has to be updated accordingly. For the application of the template the variables `service` and `interface` have to be mapped to the concrete element stereotypes within an `AdaptedAnalysis` configuration. Additionally, the variable `communication` has to be mapped to the respective property type of an interface and `edgeType` has to mapped to the stereotype depicting the *realization* dependency.

M2: Security similarity

To quantify the similarity of security requirements at the different interfaces of a service, a metric is established that provides the ratio of interfaces with no security requirements. Security requirements may specify in this case the need for authentication or authorization. The presence of a value '*no*' or the absence of such a property are assessed.

The template in listing 6.14 provides the `NegatingPropertyRatio`. This metric determines the ratio of interfaces of a service that have the value '*no*' for a certain property type.

```
1 Template NegatingPropertyRatio{
2   "Ratio of connected elements with property value 'no'."
3   as Element Metric defined with calculation rule:
4   COUNT (connected edgeType:"edgeType" to (nodeType:"connectedElementType"
5     AND having property propertyType:"property" with value "no"))
6         /
7   COUNT (connected edgeType:"edgeType" to (nodeType:"connectedElementType"));
8   for types (nodeType:"elementType")
9 }
```

Listing 6.14: Template definition for metric M2.

Considering the authentication property of interfaces, the metric provides for each service the ratio of interfaces that do not require an authentication. Different requirements at the interfaces of a services are always a hint that the single responsibility principle is not met. Thus, only in the case of zero or one all interfaces have the same security requirement regarding authentication. For metric evaluation in a concrete context, the variables within the template have to be mapped respectively. The `elementType` is mapped to *service*, the `connectedElementType` to *interface*, the `edgeType` is mapped to *realization* and finally the `property` to *authentication*.

M3 and M4: Distribution of call frequency and data volume at interfaces

The call frequency and the data volume are important to assess the scalability principle. These values are either measured by monitoring tools or estimated by developers or architects. Therefore, the distribution of these interface properties is determined for each service. It is measured through determining the empirical variance of the numerical property values. The Arla template definition in listing 6.15 contains the calculation rule to determine the average value of call frequency respectively data volume for each service (lines 1-9). Therefore, all `property` values of related elements with the type `connectedElementType` are added and divided through the total number of connected elements with type `connectedElementType`. The metric is executed for all element of type `elementType`.

```
1  Template Average{
2    "Calculates the average of numeric 'property' values for an 'elementType' from
         connected elements with 'connectedElementType'."
3    as Element Metric
4    defined with calculation rule:
5      SUM (propertyType:"property") for connected (nodeType:"connectedElementType")
6        /
7      COUNT (connected (nodeType:"connectedElementType"));
8    for types (nodeType:"elementType")
9  }
10 Template Variance {
11   "Calculates the variance of numeric 'property' values for an 'elementType' from
         connected elements with 'connectedElementType'."
12   as Element Metric defined with calculation rule:
13     (SUM ((propertyType:"property" - propertyType:"Average")*
14       (propertyType:"property" - propertyType:"Average"))
15       for connected (nodeType:"connectedElementType"))
16         /
17     (COUNT (connected (nodeType:"connectedElementType")));
18   for types (nodeType:"elementType")
19 }
```

Listing 6.15: Template definition for metric M3 and M4.

Within the second calculation rule (lines 10-19), the `Average` value is utilized to determine the variance. Therefore, the two analyses have to be composed with each other according to the rule:

```
apply Variance on Average
```

To determine the distribution of call frequency and data volume, the variables have to be mapped according to the following mapping:

Template variable	*Mapping for M3*	*Mapping for M4*
`elementType`	service	service
`connectedElementType`	interface	interface
`propertyType`	frequency	data volume

M5: #(A)synchronous usages

To determine the number of (a)synchronous usages, the same Arla template definition as for M1 (number of (a)synchronous interfaces) can be used. The template is provided in listing 6.13. Since the utilized `EdgeCondition` within the metric is evaluated independently from the actual direction, this template provides the required metric for M5 as well. For M5, `edgeType` has to be mapped to a stereotype depicting a *used by* relation. Then, the ratio of *interfaces* with property value *asynchronous* that are related to the service with a *used by* relation is determined. For the *synchronous* case, the value field within the template has to be updated.

M6: Longest synchronous call trace

To determine the longest synchronous call trace the path analysis is utilized. With an individual analysis configuration, the dependency chains between the services can be assessed. The respective template definition is provided in listing 6.16.

A call trace is interpreted as a dependency chain starting from a service s that synchronously calls an interface i_1. This interface i_1 is realized by another service s_1 that calls the interfaces $i_2, \ldots, i_n$ which are realized by further services $s_2, \ldots, s_n$. For analysis definition we assume that the synchronous call dependency is mapped to the edge class `behavioral dependent of`. This indicates that the service requires the called interface to provide its functionality. If the interface is not available, the service cannot perform its task.

Within the path analysis configuration the source and target stereotypes are restricted to *services*. The `provide` class is considered in incoming direction for path determination and the `behavioral dependent of` in outgoing direction.

```
1  Template CallTrace {
2    "Determines all call traces."
3    as Aggregate Pathset defined with path definition:{
4      path type CustomPath
5      SourceStereotypes (nodeType:"service")
6      TargetStereotypes (nodeType:"service")
7      Incoming (edgeClass:Provide)
8      Outgoing (edgeClass:BehavioralDependentOf)
9    }
10 }
```

Listing 6.16: Template definition for metric M6.

The final result of this analysis is a set of all synchronous call traces. Within Arla currently no further processing of the paths itself is possible.

M7: Average size of messages

The average size of message can be determined with the `AggregateAverage` template in listing 6.17. This template definition specifies an aggregated metric for the whole architecture. This generic definition adds all values of a specific `property` from elements with type `elementType`. The sum is divided through the total amount of elements with

type elementType.

```
1 Template AggregateAverage{
2   "Description"
3   as Aggregate Metric defined with calculation rule avg:
4     (SUM (nodeType:"elementType".propertyType:"property"))
5       /
6     (COUNT(nodeType:"elementType"));
7 }
```

Listing 6.17: Template definition for metric M7.

According to the stereotype of a concrete EA model, the variables have to be mapped respectively. This could be for example a mapping of elementType to *interface* and of property to *message size*. In this case, the interfaces within the EA model have a respective message size property value.

M8: Utilization degree of technologies

The measurement of the utilization degree of technologies is illustrated with the template definition in listing 6.18. The metric determines the ratio of interfaces that utilize *RMI* as communication method.

```
1 Template SOAPUtilizationDegree {
2   "Description"
3   as Aggregate Metric defined with calculation rule avg:
4     COUNT(nodeType:"interface" AND
5       having property propertyType:"communication method" with value "RMI")
6         /
7     COUNT(nodeType:"interface");
8 }
```

Listing 6.18: Template definition for metric M8.

According to the available information within a concrete EA model, further implementations of this metrics can be used to provide a metric measuring the independence of microservice.

M9: Synchronous dependency cycles

Finally, assessing cycles within an EA model is not directly supported with the A2F. To provide a respective metric, a custom data-flow analysis is implemented that determines an attribute cycle. If the element is part of a cycle, the attribute identifies the cycle the element belongs to. Otherwise the attribute has the value 'zero'. For the implementation of the data-flow analysis, the strongly connected component detection algorithm proposed in [Saa14] is adapted. Thereby, only incoming provide and outgoing behavioral dependent of relations are considered. The custom data-flow analysis can be referenced within an Arla analysis definition (see listing 6.19). Therein, the respective data-flow analysis configuration file is provided as well as the strategy which should be evaluated.

```
1 Analysis Cycles {
2   "Determine a cycle id for each element."
3   as Element Metric
4   defined with DFA configuration:
5   Configuration path "CycleAnalysisConfiguration"
6   Strategy "cycle-attribute"
7 }
```

Listing 6.19: Template definition for metric M9.

The provided analysis style and result type define the representation of the DFA result. In this case, the cycle id will be provided as numerical result for each element. A subsequent metric can make use of the result to determine all elements having the `Cycle` attribute with value greater than zero. This depicts the number of elements that are part of a synchronous dependency cycle.

6.5 Related Work

Within current literature several publications propose analyses or metrics to assess the architecture. For example, Hanschke [Han13] provides several analysis patterns to identify needs for action and optimization potentials within EAs. Matthes et al. propose in [MMSS11] an EAM KPI catalog. The 52 proposed KPIs to measure EA management goals. Each metric is related to one or more EAM goals and they provide best practice interpretations of the results. This supports the architect during architecture assessment and during identification of required actions. The implementation of the metrics is not further considered within these two publications.

Regarding the assessment of microservice architecture, scenario-based or metric-based evaluation is proposed by [BWZ17a]. The evaluation results are used to assess the quality of attributes of a software system and to provide a solid foundation for planning activities. There exists a large number of metrics for measuring and evaluating services and service-based systems. [BWZ17a] provides an overview of the metrics in current literature with focus on maintainability and analyzed their applicability for microservice systems. Despite some minor limitations, in particular centralization metrics, the majority of the identified metrics are also important for microservice systems. Specific evaluation approaches for microservices are rare, especially practicable and automatic evaluation methods are missing [BWZ17b].

In [ZNL17] constraints and metrics are presented to automatically evaluate microservice decomposition architectures. The authors define metrics to assess pattern conformance regarding decomposition. They focus on two aspects, the independence of the microservice and the existence of shared dependencies or components. The metrics and constraints rely on information about the connector type, i.e. whether it is an 'in-memory' connector or a loosely coupled connector. For example, to quantify the independence degree the ratio of the number of independent clusters to the number of all non-external components is used. Ideally the cluster has the size 'one' and thus the resulting ratio is 'one'. Components are aggregated to a cluster if they have an in-memory connection.

Bogner et al. [BWZ17b] present a maintainability model for service-oriented systems and also microservice systems. For the identified quality attributes the authors determine applicable metrics to measure the system architecture. For example, the coupling degree of a service can be measured considering the number of consumed services, the number of consumers and the number of pairwise dependencies in the system. Metrics for measuring cohesion are the diversity degree of parameters types at the interfaces. A highly cohesive service can also be identified by considering the distribution of using services to the provided endpoints. Granularity is for example measured with the number of exposed interface operations. Further quality attributes are complexity and code maturity. A weakness of [BWZ17b] is the missing tool support and integration into the development process.

6.6 Conclusion

Within this chapter we propose a procedure for weak point analysis in EA models that takes care of the problem of bad data quality. We propose the subsequent evaluation of model quality metrics to validate the conformance of the EA model to the constraints provided in the original meta model as well as to the modeling guidelines. Additionally, we propose the execution of an analysis specific assessment of the model quality. Depending on the statements within an analysis configuration, the model is evaluated and the elements are determined that may hinder the analysis execution. The metrics are implemented with the A2F, the analysis specific assessment is performed directly with SPARQL queries. The Arla analysis definition can be used as input for query generation.

To evaluate the weak points of an architecture, we consider the design of microservice architectures in detail. Within an exploratory study of five microservice projects we identified several challenges. Major issues are the context definition of microservices, keeping an overview of the systems and the effects of refactoring and non-local changes. Additional, cyclic dependencies, monitoring, logging, consistency and testing are mentioned topics by the interviewed experts. Based on these challenges and the common characteristics identified within literature, we derived principles for the microservice architecture design. Using the Goal Question Metric approach we developed metrics for architecture assessment. The metrics are finally implemented with the A2F. In contrast to other publications like [Han13, MMSS11] the metrics are provided as a directly executable template catalog.

7
Evaluation of Planning Scenarios

In this chapter an applicable method for Enterprise Architecture Planning (EAP) is developed, in specific for the evaluation of planning scenarios. Based on an analysis of the information demands within EA planning, analysis potential is determined. Using the A2F, templates are provided that support the proposed tool-supported method for scenario evaluation.

The research provided within this chapter was previously published in [LB18b] and [LB18a].

7.1 Method Blocks for EA Planning

EA planning deals with the development and implementation of a desired target architecture. A target architecture can be defined as an independent EA model capturing the elements and relations of a future scenario. The target architecture can also be defined in terms of goals that should be reached or strategies to follow.

The target architecture is implemented with so called strategic projects. Despite those strategic projects also demand-driven projects are executed that implement new business requirements or technology changes. It is important to ensure that the projects conform to the current strategies and goals. Adequate evaluation possibilities are required to decide about the fitness and conformance of new projects [ACS15]. The respective stakeholders require aggregated and integrated information for this task [Pul06, RGA07]. Additionally, they utilize different viewpoints, created upon partial EA models, for decision-making [RGA07]. Using quality attributes the strategy and goal fulfillment can be quantified. The measurements enable the monitoring of the architecture transformation towards the desired target architecture. In current practice, the proposed methods for EA planning from literature are not widely adopted, often because of insufficient data quality but also because practical approaches, especially for comparisons, are missing [NFT+17].

In the following, we present the results of a literature research of EA planning processes. We identified common method blocks utilized within these processes. [NFT+17] does not present a process itself but summarizes requirements for EAP from literature and practice. [FB08] identifies best practices in this context. Table 7.1 provides a summary of the seven identified method blocks as well as their mentions within literature.

Table 7.1: Common method blocks for EA planning.

	Method block	**Authors**
M1	Evaluation of cost, risk, metrics and performance	[Nie06, SH93, AGSW09, AS11, PH05, Pul06, NFT+17, The18]
M2	Development of scenarios	[Nie06, NFT+17, AGSW09, AS11, PH05, Pul06]
M3	Use of domain architectures	[AS11, PH05, Pul06, FB08, NFT+17]
M4	Evaluation of conformance to principles and goals	[ASML12, FHBB12, Nie06, SH93]
M5	Evaluation of gaps	[Nie06, SH93, The18, NFT+17]
M6	Evaluation through different visualizations	[Nie06, The18, NFT+17, Pul06]
M7	Evaluation of impact	[ASML12, The18]

The most mentioned method block is the *evaluation of cost, risk, metrics and performance*. This method block summarizes a plethora of different approaches that are used to quantify the EA or a scenario. The metrics are used to analyze and compare current and target model as well as interpret the results with respect to an ideal value [ASML12]. [SH93] propose the measurement of cost savings and economies to determine the return on investment of breaking down redundant support. In [AS11] architecture principles are used as input for the evaluation. The scenarios are evaluated towards their conformance to them and are also subject to business case calculations. Within the change management process in TOGAF [The18] a performance analysis is proposed. Based on the result and

the current performance goals a decision about the change request is made. [PH05, Pul06] propose a discussion of risks, costs and benefits in the context of EA planning.

Most EA planning processes utilize the concept of *scenario development*. The desired target architecture as well as new business requirements can be implemented in different ways [AGSW09]. During EAP, this is captured with the creation of several scenarios or also alternatives [Nie06, NFT+17, AGSW09, AS11, PH05, Pul06]. In order to decide for the best solution regarding the EA strategy, different alternatives should be developed and compared with each other. Finally, the scenario which fits best to the specified goals and requirements is chosen and implemented.

Another important method during EAP is the establishment of *domain architectures*. Although the understanding of domain architectures varies within literature, the underlying concept of reducing the scope of the model under consideration is the same. We follow the definition of [FB08]. They understand a domain architecture as a special focus on a group of products, services or functions.

Ensuring the *conformance of planning scenarios to the principles and goals* is mentioned in [ASML12, FHBB12, Nie06, SH93]. Especially demand-driven projects have to be evaluated regarding their conformance to the EA principles [ASML12]. Also in practice, the necessity to define projects with respect to the EA strategy is recognized [FB08]. The EA strategy can be defined in terms of a vision or target architecture that should be realized but also with a set of principles and guidelines that must be fulfilled. Finally, goals can be established to enable the monitoring of the progress.

The *evaluation of gaps* is an essential mean to compare the current and target architecture with each other. Proposed by [Nie06, SH93, The18, NFT+17], the gap analysis provides the foundation for the creation of the transformation plan. New elements that have to be introduced and retired elements can be identified. More detailed gap analyses can also identify changes regarding the relations among the elements.

The *evaluation based on visualizations* is a commonly mentioned method during EAP. In current practice, decision making during EA planning is mostly made based on a visual comparison and analysis of the models [NFT+17]. The differences between the stakeholder drive the need for the creation of specific ones [Pul06, Nie06]. Within [The18] the creation of a plethora of different diagrams and matrices is proposed to fulfill the information demands.

Finally, the *evaluation of impact* is proposed by [ASML12] to determine the effects of changes to other architectural elements. Also in [The18] impact analysis is proposed to identify potential dependencies of so-called road map candidates to other parts of the EA model. Road map candidates denote elements that are subject to change.

7.2 Evaluation Process for Planning Scenarios

In the following we propose a method for the evaluation of planning scenarios within EAP based on the identified common method blocks. The method was created within an iterative process consisting of the activities development of process steps, enhancing tool support and evaluation of its applicability. Tool-supported analyses are employed for the single method steps to manage the complexity and size of current EA models. The data foundation for the analyses is provided through utilizing established architecture models. To ensure the applicability, the method and the analyses should be adaptable to individual needs as well as combinable with existing EAP processes and EA models. The A2F (chapter 5) is utilized to provide the tool support and enables generic analysis definition as well as the customization of them to the specific needs of the stakeholders.

Figure 7.1 gives an overview of our proposed method and its placement within EAP. Preliminary, the goals and principles for the enterprise architecture have to be defined as well as metrics in order to quantify them. Additionally, the current architecture is documented and up-to-date and partial model(s) representing the scenario(s) are available. Planning scenarios may introduce changes to the business model, like a new provisioning model for products, but also changes within the application or infrastructure architecture. For each scenario, the evaluation process will be executed and finally, the results can be compared to each other. As a result, the architect decides for one scenario and starts the development of an implementation and migration plan. If the evaluation results are not promising, the scenario(s) can be rejected and improved. This triggers a new iteration of the method.

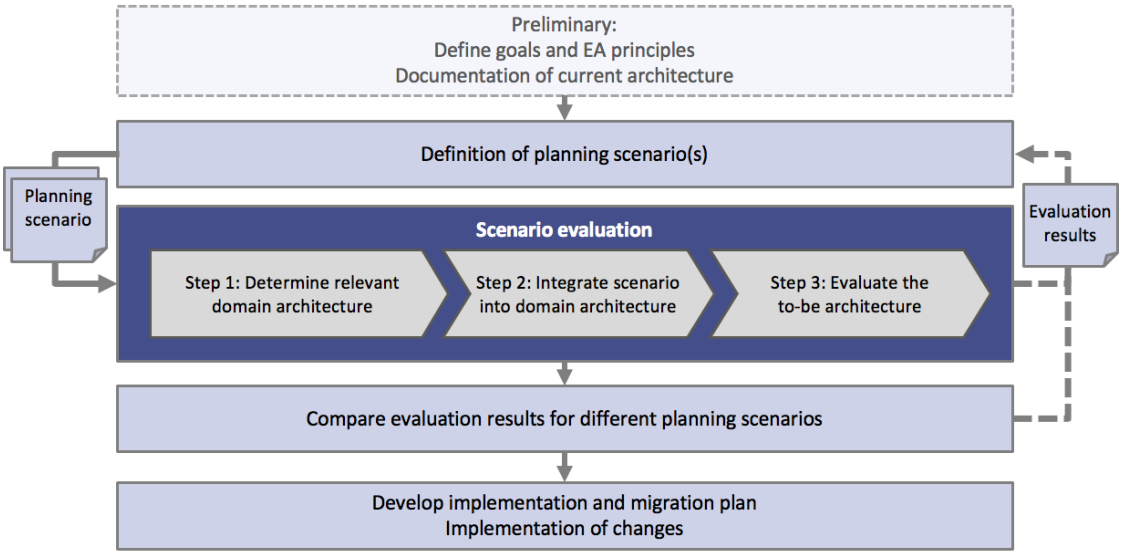


Figure 7.1: Planning scenario evaluation process.

The proposed evaluation process consists of three steps: the determination of the relevant domain architecture in order to narrow the scope and support user-based verification; the integration of the scenario into the domain architecture and generation of the target architecture; and finally, the evaluation of the target architecture.

Domain architectures are an often-proposed concept to provide manageable parts of the

architecture, where dependencies can be obtained visually [AS11, Pul06, FB08]. Due to the large nature of EA models and their high complexity, domain architectures and views are required to enable user feedback throughout the planning process. Full automation without user feedback is not applicable, since the architect has relevant but non-written knowledge about the architecture and the strategy [ASML12]. Thus, user verification of created artifacts and feedback loops are another main concept of our method.

A further key issue within the method is the evaluation, i.e. quantification, of the scenario to be able to decide about its fitness to the EA strategy. Capturing the defined goals and principles with metrics enables their measurement and their comparison. Comparisons can be done between the current and the target architecture but also to ideal values. The consistency of the planning scenario and the resulting target architecture is another important issue. With adequate support the direct and indirect effects are determined and the validity of the model can be assessed.

The key analysis concepts utilized within the single steps are gap analysis, scope analysis, impact analysis as well as metric calculation. Figure 7.2 presents their mapping to the process steps.

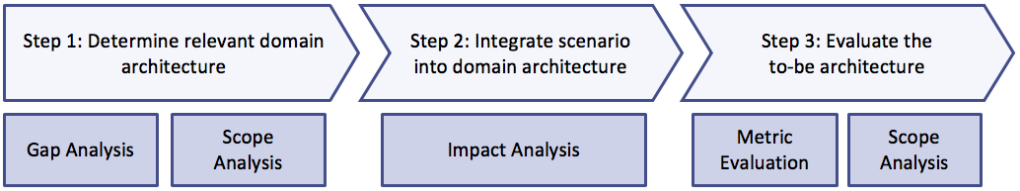


Figure 7.2: Analysis support within the process steps.

The method block M5, gap analysis, is used at the beginning to identify the differences between the current architecture and the planning scenario. This enables the derivation of a planning status attribute for each element and thus, the creation of a transformation model linking both models. With a subsequent scope analysis the relevant domain of the planning scenario is determined (M3). This domain architecture is used for the subsequent steps, since it provides a manageable part of the architecture. The architect is able to make valid statements, since the provided model is limited in its scope.

The planning scenario is integrated in the current architecture to provide a target architecture for further evaluation purposes. With an impact analysis the consistency can be further assessed (M7). Finally, if the target architecture is approved by the architect, it can be used for evaluation purposes. To address M1 and M4, metrics are applied to measure the goal fulfillment as well as cost and benefits. These results support the architect during decision-making. Additionally, further views can be created for the target architecture with the scope analysis to provide more detailed information (M6). Experts can utilize these views for a quantitative assessment of the target architecture. Finally, M2 is supported, since the proposed process can be applied for each scenario. The provided metrics and views can be compared with each other to choose the best scenario according to the specified goals. Additionally, the gap analysis can be utilized again to determine the differences between the scenarios.

The details for each process step are provided within the following subsections. Thereby, the enterprise model of the RentalCar company (see figure A.1) is used for illustration purposes. As planning scenario, the 24h-car return in figure 4.8 is used. The execution

details for the employed analyses are described in section 7.3.

7.2.1 Determine relevant domain architecture

Within the first step, the relevant domain architecture for the planning scenario is identified. Therefore, the dependencies between the current EA and the scenario are defined through a comparison of them. This task is supported by the *Gap Analysis*. The result provided by the gap analysis is presented in figure 4.9. According to the analysis result, a planning status attribute is determined for each element:

Unaffected The element is only present in the current architecture or the element is available in the current architecture and the planning scenario and the properties and relations keep unchanged.

Affected The element is present in the current architecture and the scenario, at least one property or relation is changed.

New The element is only present in the planning scenario.

Deleted The element is only present in the current architecture.

Since the scenario describes uncompleted part of the future architecture, deleted elements cannot be specified automatically. The architect has to define them explicitly. The gap analysis provides a further set of deletion candidates that support him in this task. Deletion candidates are part of the current architecture and not in the scenario, but have a relationship to an element of the scenario.

New elements can have a predecessor element in the current architecture which they replace. This successor information must be added by the user, too. To support this task, the gap analysis calculates a set of predecessor proposals for each new element. The provided successor proposals for the RentalCar example are presented in figure 4.10.

Once the dependencies between the current architecture and the planning scenario are approved by the user, relevant domain architecture is generated. It is defined with respect to the content of the scenario and utilizes the same abstraction layer as the enterprise architecture. The domain architecture encompasses business as well as IT aspects. The *Scope Analysis* is used for this task. We identified the following requirements that have to be met by the domain architecture in order to provide a solid foundation for the subsequent evaluation:

- All affected and deleted elements.
- Elements that are required to provide the affected and deleted elements.
- Elements that are provided or used by the affected and deleted elements.
- Structural dependent elements.

This ensures that potential dependencies or changes that are not considered in the initial scenario, may be detected during the following evaluation. Thus, inconsistencies in the planning scenario can be identified. Result of this step is the relevant part of the current enterprise architecture for scenario evaluation. The scope of the domain architecture has to be big enough to cover all effects of the scenario but also small enough to keep manageable by the architect. The architect has to verify the generated current domain architecture and if necessary, make adaptions like including or excluding further elements. This is important, since the resulting domain architecture is used for all subsequent analysis and evaluation steps. The generated current domain architecture for the running example is provided in figure 7.3.

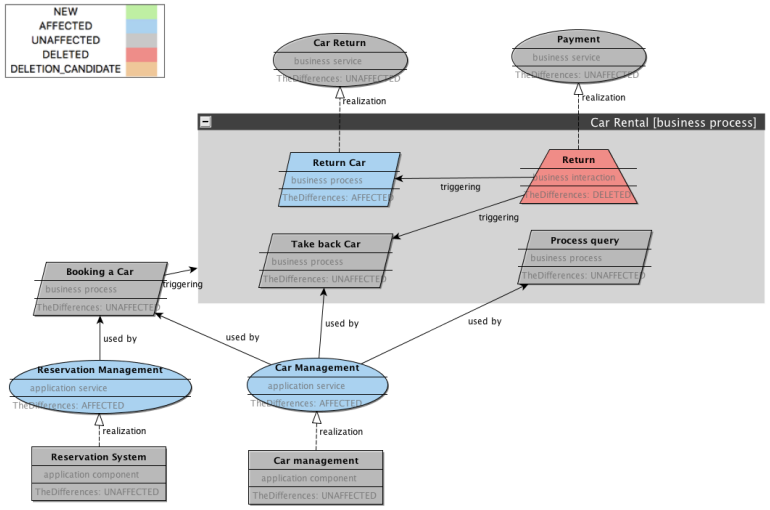


Figure 7.3: Current domain architecture for the planning scenario.

Despite the deleted and affected elements determined by the gap analysis, further unaffected ones are added according to the requirements defined above. These are the application components realizing the affected application services, the business processes that use those application services as well as the realized business services of the affected and deleted business elements. The retired element *Return* is represented as a deleted one. The other deletion candidates are set to the status `unaffected`.

7.2.2 Integrate scenario into the domain architecture

After the definition of the relevant domain architecture, the changes made by the project are integrated and validated. Therefore, the target domain architecture is established and an approximation of the change impact is used for validation. The target domain architecture contains all affected elements and unaffected element from the calculated current domain architecture. A deleted element is only present in the target domain architecture, if no predecessor is provided and thus, this element will be retired. Finally, the newly introduced elements from the planning scenario are added. An attribute indicates the planning status for this element, i.e. if it is new, affected or deleted. If a new element with an assigned predecessor is added, the relations are updated accordingly for the new element. These elements have an additional attribute identifying their predecessor elements in the current architecture. Figure 7.4 shows the target domain architecture for the 24h car return scenario. The color of the elements indicates their planning status.

The automatically defined domain architecture has to be verified by the user and if necessary further adaptions have to be made. In order to validate its scope and quality, the change consistency is assessed, i.e. if all direct and indirect effects of the proposed changes are considered. For this task, each affected element has to be assigned to one of the following impact statuses:

no No changes are actively made to the respective element.
low The element's existing functionality remains, new one is added.
medium The element's functionality is changed.
high The element is no longer available.

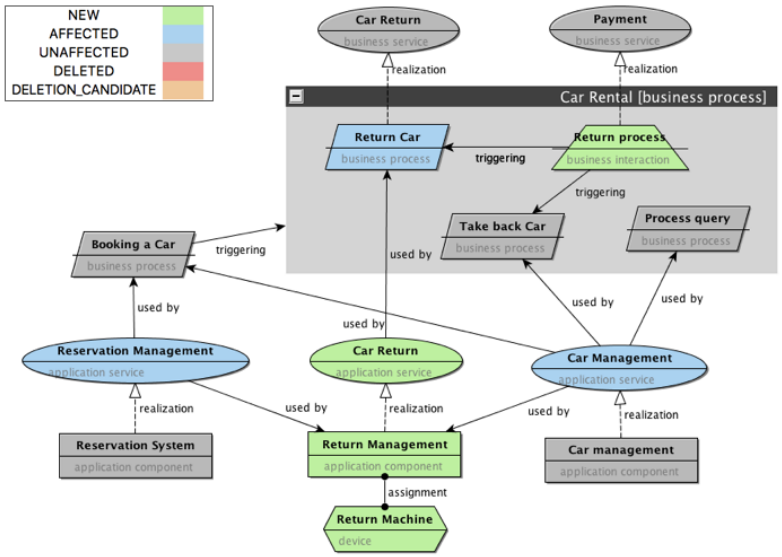


Figure 7.4: Target domain architecture for the planning scenario.

A `high` impact status is automatically assigned to all deleted elements without a successor. For affected elements, the architect has to decide about the impact of the performed changes. The information is utilized to approximate the direct and indirect effects of the changes using the *Impact Analysis*. The result is visualized, for example using different colors, within the target domain architecture and used by the architect for verification purposes.

Indicators for inconsistencies are unchanged elements, for whom a change is calculated or change effects outside the domain architecture. The last case refers to potential changes that may be unconsidered. The architect should evaluate the respective elements and optionally extend the domain architecture. Also, a large number of unchanged elements is an indicator for an insufficient scope of the domain architecture. In order to ease the ongoing evaluation, the scope of the domain architecture should be decreased.

The indicators can be evaluated with metrics. The metric for `inconsistent changes` provides the number of elements which are unaffected but have an impact attribute value low, medium or high. The metric for `scope validity` determines the elements for whom an impact probability is determined, but they are not in the scope of the domain architecture. And finally, the `unchanged elements` metrics provides the ratio of elements without impact probability within the domain architecture. Figure 7.5 presents the result of the impact analysis for the running example.

For the impact analysis, a medium impact type was initially set for the two application services and the business process *Return Car*, representing a modification of these elements. The medium impact type is depicted with an orange color, a low impact type with a blue color. The `inconsistent changes` metric identifies nine elements which are assigned with the planning status `unaffected` but an impact is determined for them. These are for example the business services *Payment* and *Car return*. Since the `unaffected` status was determined automatically during impact analysis, this status may not reflect the actual situation. If the business services will be adapted within the planning scenario, the architect should update the planning status.

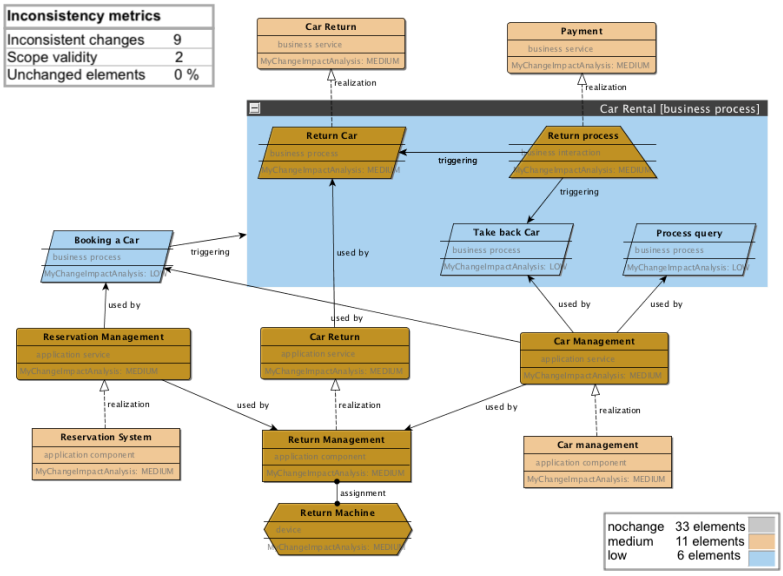


Figure 7.5: Impact analysis result for the target domain architecture.

The `scope validity` provides two elements, the business role *Renter* and the business process *Collect Bonus*, for whom an impact is calculated, but they are not part of the domain architecture. The architect has to decide about adding those elements to the domain architecture, depending on their relevance for the planning scenario.

The metric for `unchanged elements` returns a ratio of zero. For all elements within the domain an impact probability was determined. Thus, no indicator is given that the provided target domain architecture includes unnecessary elements.

The final result of this step is a verified target domain architecture. It provides a more detailed planning status regarding the impact of the performed changes as well as its scope and consistency is verified.

7.2.3 Evaluate the target architecture

Finally, the target architecture is evaluated to ensure the conformance of the scenario to the strategy. We propose view generation and metric calculation to support the architect. The quality of the designed solution can be assessed with respective views in detail. Each view serves a specific information demand and has a different focus, where irrelevant elements are hidden. For example, the business perspective hides elements from the application and infrastructure layer and therewith enables a detailed review of the dependencies between the business elements. To ease this task and to ensure the consistency of the views, they can be created with the *Scope Analysis*.

Figure 7.6 provides two different views on the target domain architecture, each representing the context for an element. Within the view on the left side the context of the two business services is provided, i.e. the services and all direct relations are retrieved. The view on the right-hand side provides the context for the business interaction *Return process*. Here again, all direct relations are retrieved to enable a detailed review of them.

Additionally, metrics can be used to quantify the target architecture and compare it with the current architecture, other alternatives or desired target values. The metrics are used

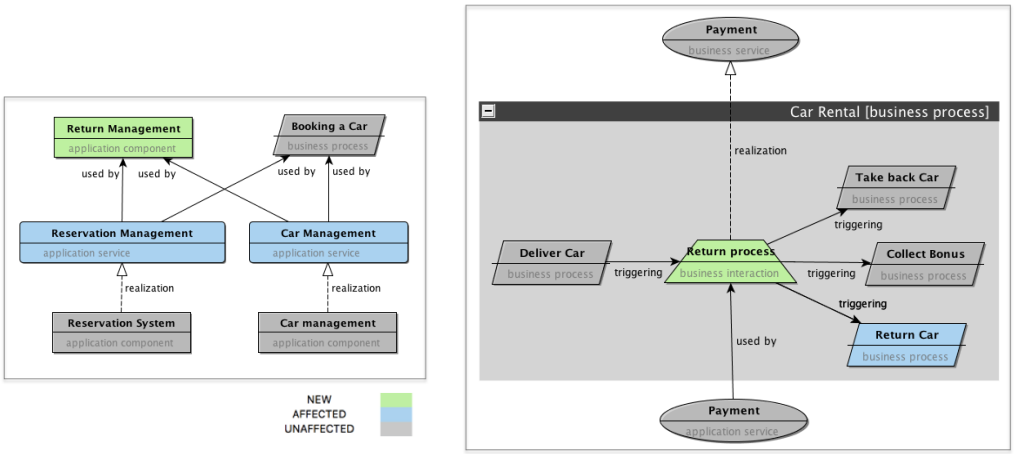


Figure 7.6: Example views determined with the scope analysis.

to quantify goals or to express architecture principles and validate their degree of fulfillment. In the following, we illustrate their application within the running example. The CarRental company has the goals of '*no manual data transfer*' between the applications and that '*a process is supported by only one IT component*'. To measure the goal fulfillment three metrics are defined:

(1) The `IT Coverage` indicates the amount of processes with IT support;

(2) `IT Usage` provides the average number of applications used by a process; and

(3) `Connectivity` as the average number of services that an application uses.

Table 7.2 shows the result for these metrics for the current and target architecture.

Table 7.2: Metric results for the running example.

Metric	Current EA	Target EA	Current domain	Target domain	Target value
IT Coverage	71%	75%	66%	100%	>80%
IT Usage	1,6	1,75	0,6	1,3	>1,2
Connectivity	0,58	0,87	0,62	1,12	>1

Based on the results of the evaluation the decision about the implementation of the scenario can be made. In the example, the target architecture with the integrated planning scenario increases the value of all metrics. Specifically, all measures of the target domain architecture fulfill the target value, i.e. this project conforms to the architecture strategy, respectively the goal that is measured with these metrics. If the evaluation results are not satisfying, they can be used as input for the definition of a new planning scenario that fits better into the architectural strategy.

7.3 Tool Support

In the following the analysis types used in the single steps of the evaluation process are summarized:

(1) Determine domain architecture: Gap Analysis, Scope Analysis
(2) Integrate scenario: Impact Analysis, Metric Calculation
(3) Evaluate target architecture: Metric Calculation, Scope Analysis

For the determination of the domain architecture the gap and the scope analysis are employed. First, the gap analysis is used to identify the differences and commonalities between the current architecture and the planning scenario. The subsequent scope analysis provides the relevant part of the architecture. The change consistency during scenario integration is verified with the impact analysis. And finally, for evaluation purposes, metrics will be calculated and the scope analysis is used in order to determine specific views. The utilized Arla template definitions to enable tool support for the single steps are presented in the following.

7.3.1 Templates for determining the domain architecture

Since the gap analysis is independent from stereotypes used in the EA model, no template is available. The configuration is directly done within a specific analysis definition. Therein, the URI from the current architecture model (the `current model`) and the URI from the planning scenario (the `target model`) are provided. Choosing the `Differences` option provides an attribute value result with the planning status, i.e. for each element it is determined whether it is a new one, an affected one or an unaffected one. Additionally, potential deletion candidates are determined. With the `SuccessorCandidates` option, for each new element within the planning scenario a set of potential predecessors is calculated.

Based on this information the architect has finally to decide about the planning status. In specific, she has to define the deleted elements as well as the successor relationships between elements form the current architecture and the planning scenario.

The approved planning status is utilized to determine the respective domain that is affected by the scenario. A domain architecture is a specific view related to scope of the planning scenario. Characteristics of domain architectures are the reduced scope and the increased level of detail with respect to the EA [BvSF+10]. Thus, the corresponding view includes the business and IT constructs related to the project. The domain architecture is defined in two steps. First all elements that are affected or deleted by the scenario are captured. The respective scope analysis template is provided in listing 7.1.

```
1 Template InitialCurrentDomainArchitecture {
2   "Determines the initial domain architecture."
3   as Aggregate Modelelementset
4   defined with set definition:
5     (having property propertyType:"planning status" with value "AFFECTED" OR
6      having property propertyType:"planning status" with value "NEW")
7   }
```

Listing 7.1: Scope analysis template to determine the initial domain architecture.

To consider the realization and usage context of these elements, for each element within the initial domain architecture the dependent elements are added. The respective Arla templates are provided in listing 7.2.

```
1  Template DomainArchitecture {
2    "Description"
3    as Aggregate Modelelementset
4    defined with composition rule:
5      apply ElementSpecificExtension onEach InitialCurrentDomainArchitecture
6  }

7  Template ElementSpecificExtension {
8    "Scope for an element that is included"
9    as  Aggregate Modelelementset
10   defined with scope definition: {
11     ModelEdge               in: None    out: None
12     ConsumedBy              in: Single  out: None
13     Provide                 in: Single  out: Single
14     StructuralDependentOf   in: Single  out: Single
15   }
16 }
```

Listing 7.2: Scope analysis template to determine the extended domain architecture.

To extend the initial domain architecture with the context of each element, an analysis composition with the `apply on each` rule is utilized (lines 1 - 6). The composition defines that for each element within the initial domain architecture, the respective context should be determined. Finally, all elements are summarized within one result set.

The context is defined within another scope analysis (lines 7ff). This analysis states that for each element all incoming and outgoing `provide` and `structural dependent of` relations should be included. Additionally, incoming `used by` relations should also be considered. With these three conditions, the context of elements that are required for the provisioning or that are used or provided by the affected or deleted elements are included. With the statement in line 13 the structural dependent elements are included, i.e. nested or container elements.

7.3.2 Templates for scenario integration

If the architect has annotated the model with successor dependencies and the planning status of the elements is validated, the integration of the planning scenario into the current domain architecture can be done automatically.

Foundation for the new target domain architecture is the current one. In a first step, all elements with planning status `deleted` are processed. If there is a respective successor in the target architecture, the deleted element is replaced with the new one. An `successor` property keeps the information about the replacement. The relations and properties of the element are updated accordingly. Relations within the current architecture are kept and those from the scenario are added.

If no successor is specified, the deleted element is kept within the target domain with the planning status attribute `deleted`. Afterwards, all new elements of the planning scenario are added to the domain architecture with their relations. The resulting model represents the target domain architecture which will be used for the subsequent analyses.

First, an impact analysis is applied to verify the scope and consistency of the changes. For such a change impact analysis the impact types `high`, `medium` and `low` can be interpreted according to the change types `deleted`, `modified` and `extended` (see section 4.3.2). Listing 7.3 provides the template with the impact configuration.

```
1  Template ChangeImpact {
```

```
2    "Determine the effect of changes."
3    as Element Attribute
4    defined with impact definition: {
5      ModelEdge                in: no_effect       out: no_effect
6      BehavioralDependentOf    in: weak_effect     out: no_effect
7      ConsumedBy               in: weak_effect     out: no_effect
8      InstanceOf               in: strong_effect   out: weak_effect
9      LocalizedAt              in: no_effect       out: strong_effect
10     Provide                  in: strong_effect   out: strong_effect
11     StructuralDependentOf    in: weak_effect     out: weak_effect
12   }
13 }
```

Listing 7.3: Template for a change impact analysis.

In order to capture the semantics of change propagation within an EA model the effect types `no`, `weak` and `strong` are assigned to the relation classes. For example, a change at the source or target of a `provide` relation has always a strong effect on the other element (line 10). In contrast for a `used by` relation (see line 7), a change at the target element has a weak impact on the source. In the other direction no impact is expected.

The templates to determine the metrics for change consistency verification are presented in the following. First, in listing 7.4 the `inconsistent changes` metric is defined (lines 1 - 9). Therein, all elements are counted that have the planning status `unaffected` and for whom during impact analysis an impact type is determined. Within the template `InconsistentChangesMetric` (lines 10 - 15) the analysis composition of the metric with the impact analysis is specified (line 14). The apply rules ensures that the respective `ChangeImpact` attribute is calculated before evaluating the actual metric.

```
1  Template InconsistentChanges {
2    "Determine the number elements which are unaffected but have an impact value
         calculated."
3    as Aggregate Metric
4    defined with calculation rule inconsistentChanges:
5      COUNT (having property propertyType:"planning status" with value "UNAFFECTED"
6         AND (having property propertyType:"ChangeImpact" with value "LOW" OR
7               having property propertyType:"ChangeImpact" with value "MEDIUM" OR
8               having property propertyType:"ChangeImpact" with value "HIGH"));
9  }

10 Template InconsistentChangesMetric {
11   "Ensure the execution of change impact analysis before metric evaluation."
12   as Aggregate Metric
13   defined with composition rule:
14     apply InconsistentChanges on ChangeImpact
15 }
```

Listing 7.4: Template to determine the inconsistent changes metric.

Within listing 7.5 the templates to determine the `scope validity` is provided. Based on the results of the impact analysis, the template in lines 1 - 8 provides the number of elements with an impact type low, medium or high. The respective composition configuration is provided in the template in lines 9 - 14. Finally, the scope validity is determined in lines 15 - 20. Within this composition configuration the `ImpactedElementMetric` is restricted to the elements that are not in the domain architecture (line 19). These elements are provided within another template `NotDomainArchitecture`.

```
1  Template ImpactedElements {
2    "Determine the number elements which have an impact value calculated."
3    as Aggregate Metric
4    defined with calculation rule impactedElements:
```

```
5    COUNT (having property propertyType:"ChangeImpact" with value "LOW" OR
6           having property propertyType:"ChangeImpact" with value "MEDIUM" OR
7           having property propertyType:"ChangeImpact" with value "HIGH");
8  }

9  Template ImpactedElementMetric {
10   "Ensure the execution of change impact analysis before metric evaluation."
11   as Aggregate Metric
12   defined with composition rule:
13   apply ImpactedElements on ChangeImpact
14 }

15 Template ScopeValidity {
16   "Determines the elements with an impact but which are not in the domain
         architecture."
17   as Aggregate Metric
18   defined with composition rule:
19   apply ImpactedElementMetric on NotDomainArchitecture
20 }
```

Listing 7.5: Template to determine the scope validity metric.

The ratio of unchanged elements within the target domain architecture is determined with the two templates provided in listing 7.6. The calculation rule for the metric is defined in lines 1 - 6. Thereby, the number of elements with impact status NO is divided through the total number of elements having an impact status, regardless of the attribute value. As for the two other metrics, an additional template for a composed analysis (UnchangedElementsMetric) is required to ensure the availability of the impact attribute. The configuration of this composed analysis is similar to the on provided in listing 7.4, lines 10 - 15.

```
1  Template UnchangedElements {
2    "Provides the ratio of unaffected elements to all elements"
3    as Aggregate Metric defined with calculation rule unchangedElements:
4      (COUNT (having property propertyType:"ChangeImpact" with value "NO")) /
5      (COUNT (having property propertyType:"ChangeImpact"));
6  }

7  Template UnchangedDomainElements {
8    "Restricts the unchanged elements metric to the domain architecture."
9    as Aggregate Metric
10   defined with composition rule:
11     apply UnchangedElementMetric on DomainArchitecture
12 }
```

Listing 7.6: Template to determine the ratio of unaffected elements.

Additionally, to restrict the metric to the domain architecture a further analysis composition is defined. Within the UnchangedDomainElements the UnchangedElementMetric is applied on the result of the analysis DomainArchitecture. Thus, for metric calculation only the elements within the domain are considered.

7.3.3 Templates for target architecture evaluation

The scenario evaluation is performed depending on the organization's strategy and goals. A common view that can be used during evaluation is the context view for an element. This scope analysis provides all direct relations for one or more selected elements. Listing 7.7 provides the respective template definition. The definition was also used to create the two example views in figure 7.6. For view generation each edge class should be considered

according to the single constraint, i.e. the source respective target of the relation is included within the result but none of the further related elements.

```
1  Template ElementContext {
2    "Provide an element specific view."
3    as  Aggregate Modelelementset
4    defined with scope definition: {
5      ModelEdge                in: None    out: None
6      ConsumedBy               in: Single  out: Single
7      LocalizedAt              in: Single  out: Single
8      Provide                  in: Single  out: Single
9      StructuralDependentOf    in: Single  out: Single
10     BehavioralDependentOf    in: Single  out: Single
11     Generalization           in: Single  out: Single
12     InstanceOf               in: Single  out: Single
13   }
14 }
```

Listing 7.7: Template to determine the context view for an element.

To illustrate the metric application for measuring goal fulfillment, the template for the IT coverage metric is presented in listing 7.8. The metric can be used to monitor the coverage of IT support for business processes. Two variables are utilized: the *business element* and the *application element*. Depending on the concrete meta model they can be mapped to processes or business services respectively application services or application components. The metric is determined by dividing the business elements with application support through the total number of business elements.

```
1  Template ITCoverageMetric {
2    "Indicates the amount of processes with IT support"
3    as Aggregate Metric
4    defined with calculation rule itCoverage:
5      (COUNT(nodeType:"business element" AND
6             having relation to (nodeType:"application element")))  /
7      (COUNT(nodeType:"business element"));
8  }
```

Listing 7.8: Template for the IT coverage metric.

To provide a comparison table as provided in table 7.2 the template is executed within the current and target architecture. Additionally, with an analysis composition the metric can be restricted to the current respectively target domain architecture.

7.4 Related Work

Current EAP processes are not really accepted within practice. Due to a missing practicability of them [NFT+17], the decisions during EAP are often based on a visual inspection of available models. However, these models are not specific for this task and often do not provide the relevant information in an adequate way. The manual creation of the required views as well as ensuring their consistency is very time consuming. The flexible creation of individual views and metrics is not sufficiently supported in current approaches from research (regarding diversity) and in tools (regarding flexibility).

The steps within our proposed method are developed with respect to the EAP processes in current literature. We integrated the method blocks identified in literature (see table 7.1) in our approach as well as propose tool-supported analyses for their execution. Comparing our method with the requirements for EAP in [NFT+17, AG10b], we cover most of them including an analysis and comparison of the current and target architecture, support for different scenarios, consideration of successor relationships between current and target architecture elements as well as consideration of specific requirements from stakeholders. Weaknesses of our approach are a missing support for life-cycles and the derivation of project activities and support for transformation paths. An additional shortcoming of our approach is the dependency on the data quality and completeness of the EA model.

Table 7.3: Comparison with related work (abbreviations according to figure 2.13).

Steps in our method	*Equivalent steps from other work*
Preliminaries (principles and strategy)	A1-A3, B1-B3, C1, D1, D2a, E1-E1
Definition of planning scenarios	A4, B4, C2, C3(a), D2b, E3
Evaluation of planning scenarios	B5, C3A, (A4), D2c, D2d, E4
Compare, decide, implement	A5-A6, B6, D3, E5, E6

In table 7.3 we mapped our EAP process steps onto the existing ones identified in literature. Process steps from literature are represented with the abbreviations introduced in figure 2.13. A step is mapped to a step from an EAP process, if they have similar main activities and the goals correspond to each other. Our main contribution is comprised within the process step *Evaluation of planning scenarios*. For the previous and subsequent steps, we do not provide further details. It is possible to follow other approaches for these tasks and use our proposed method for executing the respective evaluation step (steps A4, B5, C3A, D2c and D2d as well as E4).

7.5 Conclusion

We identified common method blocks from existing EAP processes and requirements in literature to develop a practicable scenario evaluation process. Preliminary for this process is a documentation of the current enterprise architecture, the definition of goals and principles as well as a planning scenario model. Based on these artifacts, the relevant domain architecture for the planning scenario is determined. This is supported with a gap analysis between current and target architecture and a subsequent definition of the relevant architecture part using a scope analysis. In the second step, the proposed changes are integrated in the domain architecture and an impact analysis is performed to ensure the consistency and validate the domain architecture. Finally, views can be determined onto the target domain architecture for quality reviews as well as metrics can be calculated to quantify the target model.

The analysis execution environment A2F provides a single point of access for all the different analysis types. The definition of generic analysis templates eases the execution of the proposed method. The applicability of our approach is also supported by utilizing the information within an EA model. The universality of A2F enables an easy adaption to different EA models as well as provides the functionality to extend the analyses according to specific needs from stakeholders. For example, individual metrics or view definitions can be defined and the rules for the generation of the domain architecture can be adapted.

With this work, we show the benefit of such an integrated environment, since it enables the development of an applicable and customizable method for EAP. Since current EAP methods lack acceptance in practice [NFT+17], we focused especially on this issue. Utilizing EA models as foundation for the analyses provides two advantages: First, there is no need for specific data collection in order to apply the method and second, we can show the usefulness of these models during transformation planning. Establishing EA models is an expensive task and gaining benefits from these models increases the acceptance of an EA initiative. The utilized A2F provides us with the required functionality to keep independent from a specific meta model for enterprise architecture. The required analyses for scenario evaluation are defined as generic templates within the A2F. When applying the process, these templates can be re-used as well as it is possible to further refine or extend them according to specific needs.

Part IV

Case Studies and Conclusions

8
Evaluation

Within this chapter we present the evaluation results for the developed artifacts. Next to assessing the scope of Arla and the A2F regarding existing EA analysis approaches, we employ a scenario-based evaluation. The scenarios are determined according to the technical and functional dimensions. An existing analysis approach is implemented with Arla for each scenario.

Additionally, the relevance and applicability are evaluated through application of the use cases within case studies. Two different implementations of the A2F and several EA data adapters are developed for this. Parts of the evaluation are previously published in [LB17, LB18a, Eng17, ELBH18].

8.1 Implementation

To enable the evaluation of the A2F, we provide two different implementations. One is the integration as plugin into the modeling tool Innovator [MID19]. Thereby, only a subset of the described concepts is implemented. This implementation is further described in section 8.1.1. The other implementation utilizes the model analysis product AutoAnalyze (section 8.1.2). This Eclipse-based product was developed within the research group the author was part of. AutoAnalyze is used to provide a user interaction and visualization regarding model representation and analysis execution. The implementation with AutoAnalyze encompasses nearly all of the previously presented concepts. Finally, several data adapters are developed to load existing data into the AutoAnalyze product (section 8.1.3).

8.1.1 Integration into a modeling tool

The Innovator Enterprise Modeling Suite [MID19] is a modeling tool for business analysis, software architecture, information architecture, business intelligence as well as enterprise architecture. The product from the company MID allows the integrated modeling of the different architectures. For modeling enterprise architectures, the tool relies on the ArchiMate standard. The provided meta model and notation of this standard can be adapted to the specific needs of the organization. This leads in some case to complete custom meta models used for the representation of the EA. The tool enables the definition of different diagrams that represent views of the EA model.

We integrated parts of the A2F as plugin into the modeling tool. This includes the realization of the data-flow based analyses: Impact analysis, performance analysis, path analysis and scope analysis. The customization of the analyses, i.e. analysis definition, is done with a user interface. The user interface provides a more comfortable abstraction from the textual language Arla. Analysis classes based on SPARQL as well as analysis composition are currently not implemented in the plugin.

The model data is extracted from the tool and subject to the selected analysis. Finally, the retrieved result is returned the modeling tool and visualized within the original diagrams. In the case of a scope analysis a new diagram, i.e. a view, is generated. Therewith, the plugin enables a round-trip model engineering.

Before data extraction the scope of the model can be reduced to the relevant part. This part of the model is then converted into an EMF model which is utilized for analysis execution. The GMM, presented in section 3.2, is utilized for internal model representation. This is important to support the custom defined EA meta models within Innovator. Additionally, the user input via the user interface is transferred into the respective analysis configuration. The plugin is in productive use for analyzing the EA model of a public administration. The model contains about 2.500 elements and 6.700 relations. For its description 28 element types and 39 relation types are used.

Figure 8.1 provides a screenshot of the modeling tool. The plugins for the different analysis classes can be found within the ribbon bar. Within the center the currently opened diagram is shown. The visualized model is the RentalCar model. The element colors depict the result of an impact analysis.

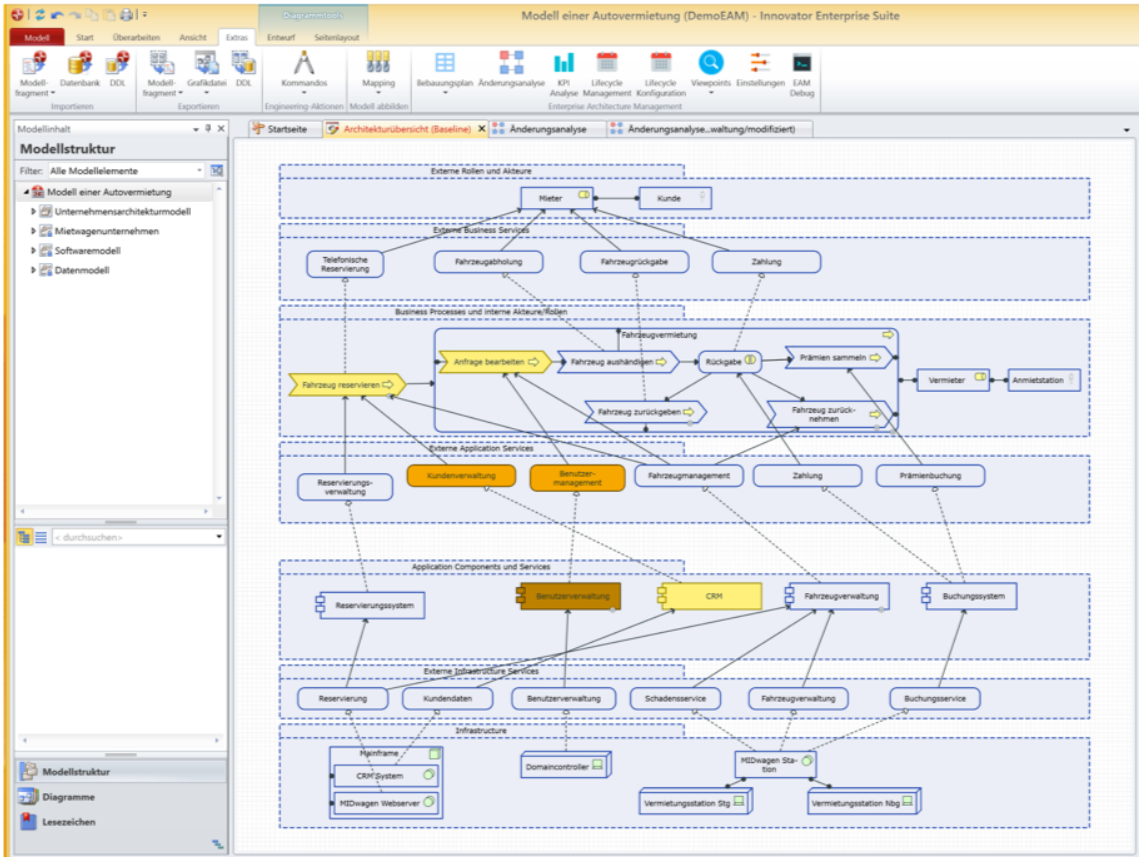


Figure 8.1: A2F Plugin for the Innovator showing the result of an impact analysis.

Meta model support

The plugin was applied in five different EA models using four different meta models. One is the EA model of a medium-sized software product vendor. This model is created utilizing the ArchiMate modeling language without further adaptions. This model is also utilized within the EA planning case study in section 8.4.1. The utilized running example RentalCar is the demo EA model from Innovator and also created along the ArchiMate standard.

The other three EA models analyzed with the A2F plugin each rely on a custom meta model. This is the EA model of a public administration, the EA model of an automotive manufacturer and the EA model of a fictional use case utilized within the tool evaluation report in [MBLS08]. Although this is a fictional use case, the model and the meta model are a good representative for actual EA models. The meta model encompasses about 25 classes, 30 associations and about 90 attributes.

During model extraction and conversion for analysis purposes, the respective meta model part of the GMM is built dynamically. This means that only those concepts are created that are actually used within the model. Additionally, the mappings between the concrete stereotypes and the edge and node classes are resolved during this step.

This procedure could be performed for all five EA models. Their conversion into the GMM format was executed without further adaptions or manual corrections. Only the mapping of the stereotypes to the node and edge classes has to be done manually once. Afterwards

the models can be utilized for the implemented analyses. The supported classes and their implementation details are described in the following.

Impact analysis

The impact analysis is customized in the same way as the analysis and template definitions within Arla. Despite providing the configuration within a textual file, a user interface is used to abstract from it. Figure 8.2 provides the user interface for impact analysis definition. Despite predefined settings for a worst- and best-case scenario, also custom assignments of the effect types can be made. The expressiveness of the user interface corresponds to the `ImpactDefinitionByStereotype` configuration. Custom impact definitions can be stored for later reuse. The worst- and best-case scenario represent the execution modes that can be triggered with the `StaticImpactConfiguration` in Arla. The result for executing the impact analysis definition is presented in figure 8.1. The colors indicate the impact status. Yellow denotes a low impact and orange a medium impact value. The dark orange color denotes the source of the impact.

Figure 8.2: Impact analysis definition within the Innovator analysis plugin.

Scope analysis

The scope analysis is captured in a similar way within the analysis plugin. The definition of a scope analysis according to the `DynamicEdgeDefinitionByStereotype` from Arla is illustrated in figure 8.3. The assignment of the concrete stereotypes to one of the constraint types `single`, `transitive` and `none` can be done within the right table of the user interface. Such a custom configuration can be stored and re-used. Therefore, the information at the top of the user interface is required. The fields denote the name and the respective diagram type that should be used for diagram generation.

The table on the left enables the restriction of the considered element stereotypes. This was necessary since the analysis plugin supports no analysis composition and no restriction of elements within a scope analysis. Additionally, the allowed stereotypes for the start elements can be restricted too. Executing such an analysis definition provides a model element set. Based on this set a new diagram is created in the modeling tool.

Path analysis

The path analysis within Innovator is used for the generation of business support maps. Business support maps provide a matrix visualization depicting the relations within an

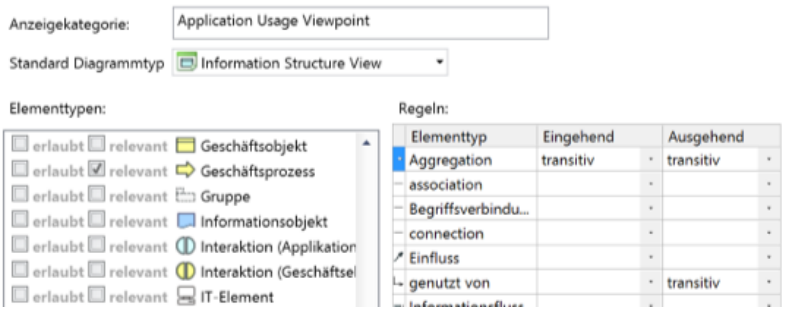


Figure 8.3: Scope analysis definition within the Innovator analysis plugin.

EA model. Figure 8.4 provides an example support map generated from the RentalCar model. The content of the matrix is filled according to the results of the path analysis. The path results that lead to the placement of an element, can be provided at the bottom. Within the figure, there are two paths for *Benutzerverwaltung* that arrange this element in the row of the infrastructure service *Benutzerverwaltung* and the column of *Telefonische Reservierung*.

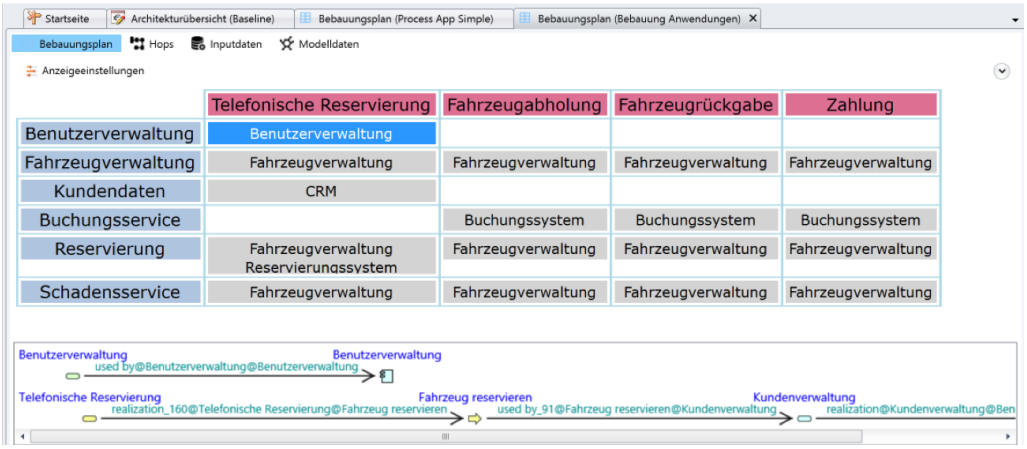


Figure 8.4: Support map within Innovator with results from the path analysis.

The support map generation can be configured with a user interface. Therein the utilized stereotypes for the rows, columns and content as well as the considered meta model paths can be selected (see figure 8.5). The provided stereotypes depict the source and target stereotypes of the path analysis in the A2F. Additionally, it can be chosen between the `AllPath` and the `ShortestPath` execution mode. If this information is provided in a first step the available paths within the meta model are calculated. The result is provided in the two screens at the bottom. The architect can now select those paths that should be considered for the final support map generation.

Performance analysis

Finally, the performance analysis is implemented within the analysis plugin. Currently the analysis cannot be customized by the user according to the property definitions. For execution the edge classes as described in section 5.5.7 are used. The result is presented as table within the modeling tool.

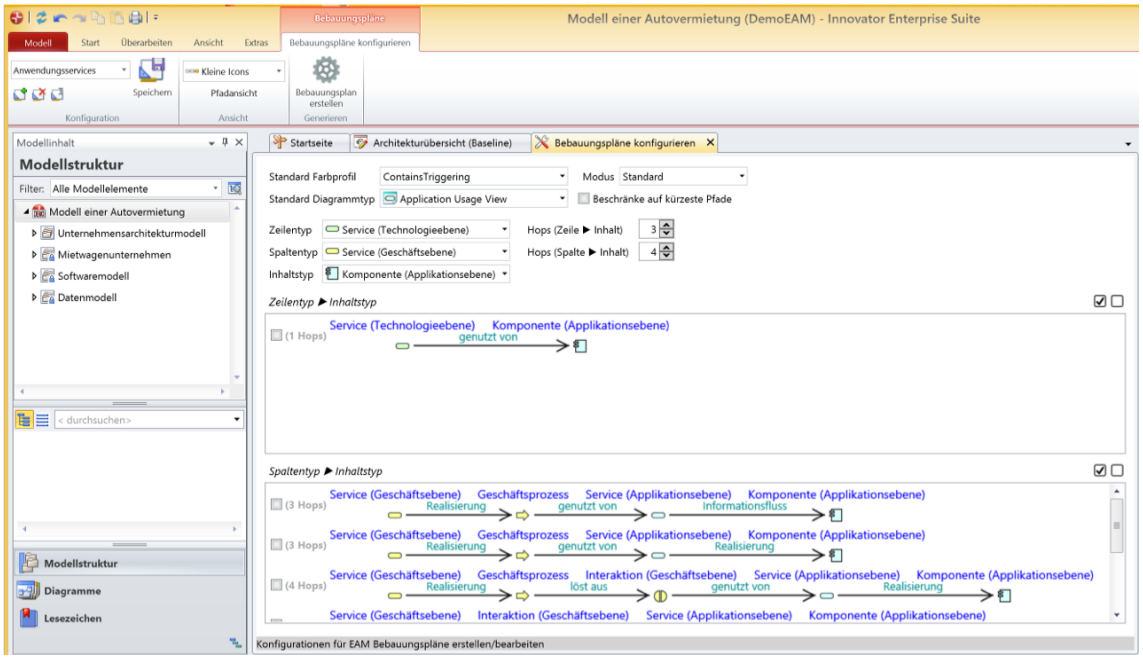


Figure 8.5: User interface for support map configuration.

8.1.2 Integration into AutoAnalyze

Despite the partial integration into Innovator, we provide an almost complete integration of A2F into the Eclipse-based product AutoAnalyze. AutoAnalyze provides a visualization and analysis execution environment for models. The registration of connectors enables the integration of individual analysis and visualization functionality. AutoAnalyze relies on the concepts of the Eclipse Modeling Framework to visualize the model data. The integration of graph libraries enables automatic layouting. The product is utilized to provide a user interface for analysis activities, especially to be able to visualize the EA model and the retrieved analysis results.

For data persistence we choose the Triple Store from Apache Jena [Fou18] as data store. Apache Jena [Fou18] is an open source framework that enables the creation of RDF graphs, to query the models with SPARQL, and provides an inference interface as well as a triple store to persist the data. Alternatives comprising similar functionality are Virtuoso [Sof15], RDF4J [RDF19] or RDF Knowledge Graph from Oracle [Ora19].

The final AutoAnalyze product supports all proposed analysis classes of Arla. A textual editor is provided for analysis specification through integrating the generated Xtext artifacts into the final product. The editor provides syntax highlighting and auto completion to support the specification of analysis configurations.

Figure 8.6 provides an overview of the AutoAnalyze product with the A2F connector. Within the center, the graph of the current EA model is visualized. Within the views on

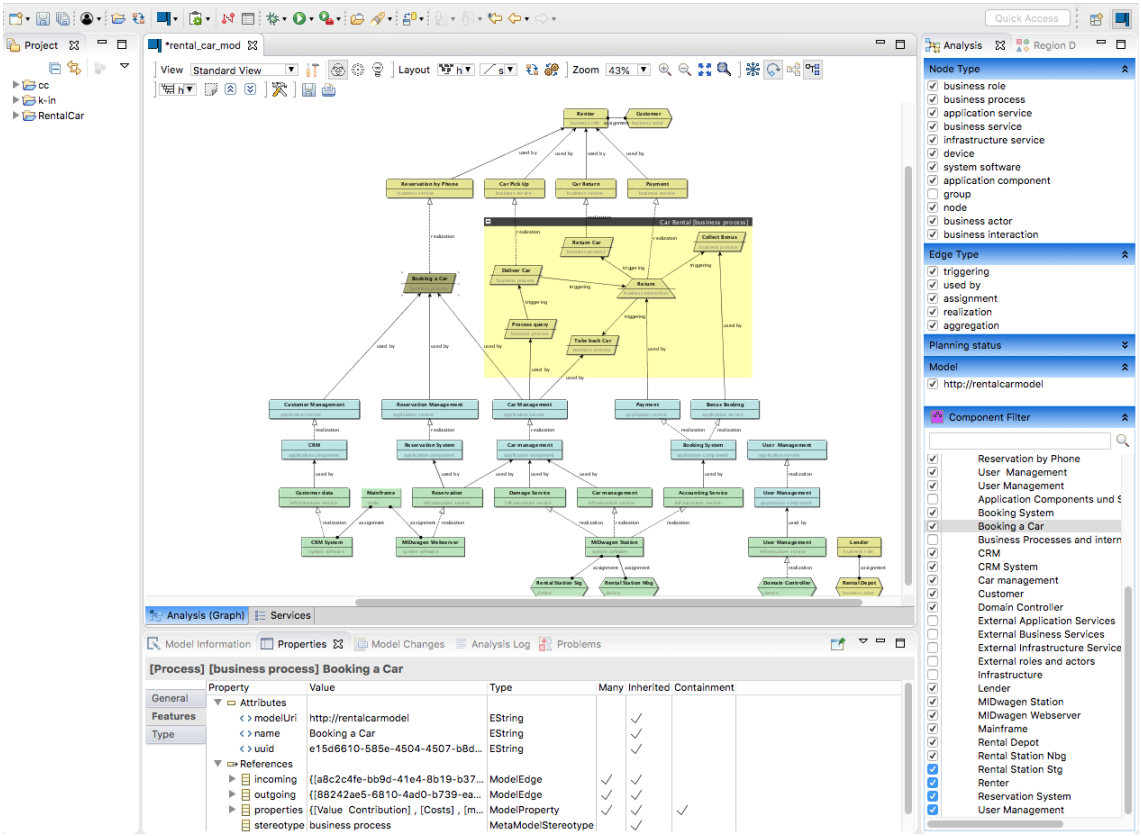


Figure 8.6: Overview of AutoAnalyze with A2F integration.

the bottom properties of a selected element as well as overall model metrics are presented. On the right-hand side, filter options enable the user interaction with the graph. Selecting and deselecting the filters leads to enabling respectively disabling of the respective elements within the graph.

Figure 8.7 provides the dialogue to load Arla analysis definition files as well as to execute the desired analysis definitions. In the table at the bottom the already loaded analysis definitions are provided. The selected ones are considered, when triggering analysis execution. At the top, further input parameters for analysis execution can be provided, like the current model element selection. This input parameter provides the start elements for impact, scope and path analysis. Additionally, it is possible to declare the analysis which should be used for coloring of the elements. The checkbox allows the restriction of the analyzed elements to the current view within the graph. This means that during analyses execution the current filter selection within AutoAnalyze is considered. Finally, the maximum number of hops within path analyses can be specified as well as the impact status that should be assigned to the start element during impact analysis.

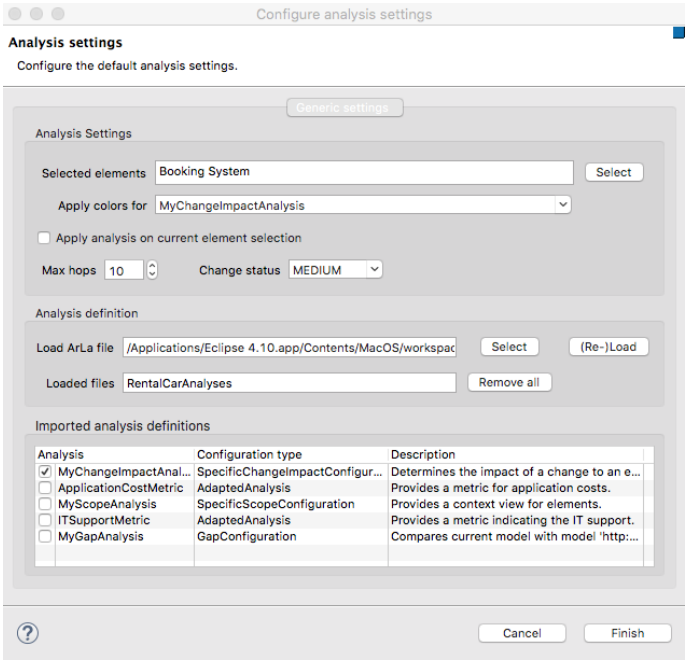


Figure 8.7: Analysis settings within AutoAnalyze.

The visualization of the analysis result is implemented depending on the result type. For model element sets and path sets respective filters are created. Additionally, for each path so called regions are provided. Regions enable a highlighting of certain elements with colors without disabling them. The coloring of the region can be enabled and disabled by the user. Attribute results for elements are always visualized within a property field at each element. The different attribute values can also be indicated with different colors as well as a filter category is created for each value. Further aggregated results like metrics are presented within a model metric view.

8.1.3 EA model adapter

The EA model has to be loaded into the AutoAnalyze product in advance of model visualization and analysis execution. We implemented several adapters for different information sources. The simplest way to accomplish this task consists of a traversal of the source model, creating corresponding GMM nodes, properties and edges on-the-fly. As part of this process, the meta information of the element has to be evaluated as well, and the *GmmMetaModel* has to be extended or updated accordingly. This approach ensures that all relevant information is transferred into the GMM representation, while unused parts of the original EA language are automatically excluded.

We developed adapters for capturing the data provided within EA tools and Excel sheets as well as adapters that focus on the actual communication dependencies between applications and services. Therewith, we address the issue of outdated data which often affects the communication dependencies.

EA tools and Excel files

We captured the syntax of the serialized Innovator model using an ANTLR [ANT14] grammar, to load them into the AutoAnalyze product. From the grammar, ANTLR generates a parser that enables the creation and walk through of the parse tree. We utilize this parser to create an EMF model using the GMM scheme. The EMF model is finally converted to and persisted as RDF graph.

Additionally, we provide support to capture the data from CSV files.
Most EA tools provide export functionality into CSV or XMI files (e.g. iteraplan [ite19], LeanIX [Lea19] or Archi [Bea19]). If no EA tool is established in an organization, the relevant data is often managed within Excel files instead. We demonstrate the CSV import with a loader for Archi export data and a loader for custom created Excel sheets. Archi [Bea19] is an open source modeling tool for EA that utilizes the ArchiMate language for the creation of the models. The loader for custom Excel sheets captures products, applications, services, interfaces and their dependencies.

Communication dependencies between applications

A source for actual communication data is provided by Application Performance Monitoring (APM) tools like Dynatrace [Dyn19a] and Instana [Ins19]. These tools are used to automatically capture operational data. Thus, they provide a very detailed view of the actual state of application landscape on a very technical level. Aggregating and processing those information makes them usable on an EA level. We implemented an adapter for the tool Dynatrace. The adapter accesses either the REST API or utilizes generated XML reports. The communication data is further processed and aggregated to provide a high-level architectural view. It is possible to extract the communication dependencies between the services from this data. Despite information about services, queues, databases as well as their communication dependencies, Dynatrace also provides information about the utilized infrastructure elements, i.e. the host, response and execution times as well as the call frequency within the requested time span.

If no APM tool is established, synchronous communication data can be retrieved with the OpenTracing standard [Ope19a]. OpenTracing is a vendor neutral API for distributed tracing that enables the collection of trace data. OpenTracing unifies the different existing tracing APIs in order to improve the exchangeability of them. Zipkin [Ope19b]

implements the OpenTracing API for all common programming languages like Java, C#, C++, Javascript, Python and Go and even for some technologies like Spring Boot. We implemented a data collector that implements the same API as Zipkin. Thus, the libraries developed for Zipkin can be used to collect synchronous data.

In contrast to synchronous data retrieval, there is no standard available for collecting asynchronous data. Therefore, we implemented our own API for asynchronous data retrieval that is independent of any message broker. The interface expects the service name, the message size, an identification endpoint, the name of the message broker, a time stamp and a type indicating if it was a send or received message. The generic interface was implemented for the message broker RabbitMQ using the built in Firehose Tracer [Piv07] to collect communication data. If the Firehose Tracer is activated, all sent and received messages are also sent to a tracing exchange and are enriched with additional meta data.

Finally, we provide an adapter that assesses log files to retrieve current communication data. Within these files, depending on the organization specific configuration, communication dependencies as well as traces can be extracted. In our case, two different log types were accessible: The *normal log* of a service contains information about the called endpoints, incoming requests and further functional information. The *access log* contains only information about incoming requests, but provides additional information about their duration and data volume. The assignment of the calls and requests to each other, in order to create the respective usage dependencies, was performed utilizing the unique request ID.

8.2 Coverage of EA Analysis Approaches

We evaluated the analysis language Arla with its implementation infrastructure A2F regarding the coverage of existing analysis approaches. We examined the identified analysis types within [Rau13] as well as the different technical and functional dimensions from [Rau15]. We refer to the types and dimensions instead of the single approaches, since it is not obvious to decide whether a following publication of an approach is a new one or an extension to an existing one. Hence, referring to the concrete approaches would provide a biased view on the coverage of the A2F.

Analysis approaches relying on expert interviews (e.g. [DBLR+11, PSP12]) or approaches addressing the method EAM like maturity analyses (e.g. [AS11]) are not considered in our evaluation. Following, we excluded all approaches assigned to the analysis types *Business Entity Analysis, Design Analysis, Run-time Analysis, Intentional Analysis, Maturity Analysis* and *Sensitivity Analysis* from our evaluation. Another 15 approaches of further types are excluded according to these criteria or since they do not directly represent concrete EA analyses. Either they provide surveys about the practical use (e.g. [BFKW06]) or they deal only marginally with the topic EAM like the social network analysis from [TTF79] in the context of organization theory. A final reason for the exclusion of analysis approaches was a missing access to the respective publication (specifically [JBR+99, WC12]). In total, 29 analysis approaches were excluded within the following coverage assessment

For the remaining 67 approaches, we decided whether they can be completely described and evaluated with the A2F (i.e. they are covered by the A2F), partially described and evaluated with the A2F or whether the A2F does not cover them. For example a partial coverage is given for approaches considering date specifications. Currently the A2F does not comprise special expressions for evaluating time spans or point of times. Indirectly, it is possible by utilizing other expression types. Some analyses assigned to the technical dimension PRM [NBE14] are also partially covered. The probabilistic aspects of these analyses which are implemented in a Monte Carlo fashion, cannot be reproduced with the A2F. By replacing the probability distributions with single values, the analyses can be defined and evaluated. An analysis approach is also declared as partially covered, if it is necessary to specify new DFA propagation rules or to create custom SPARQL queries for its realization.

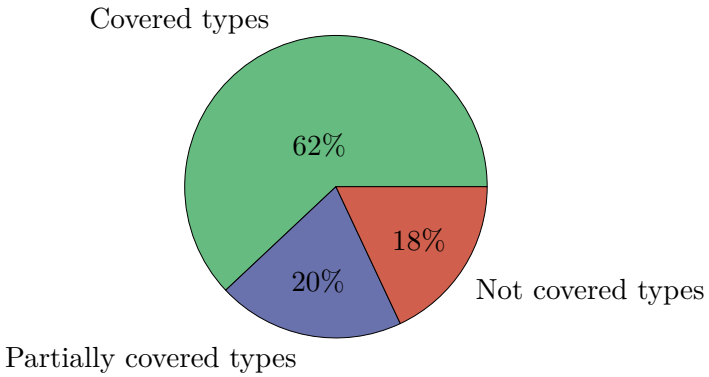


Figure 8.8: Coverage of analysis types.

Figure 8.8 provides a summary of the coverage of the analysis types. A detailed coverage report for each type is provided in figure 8.9.

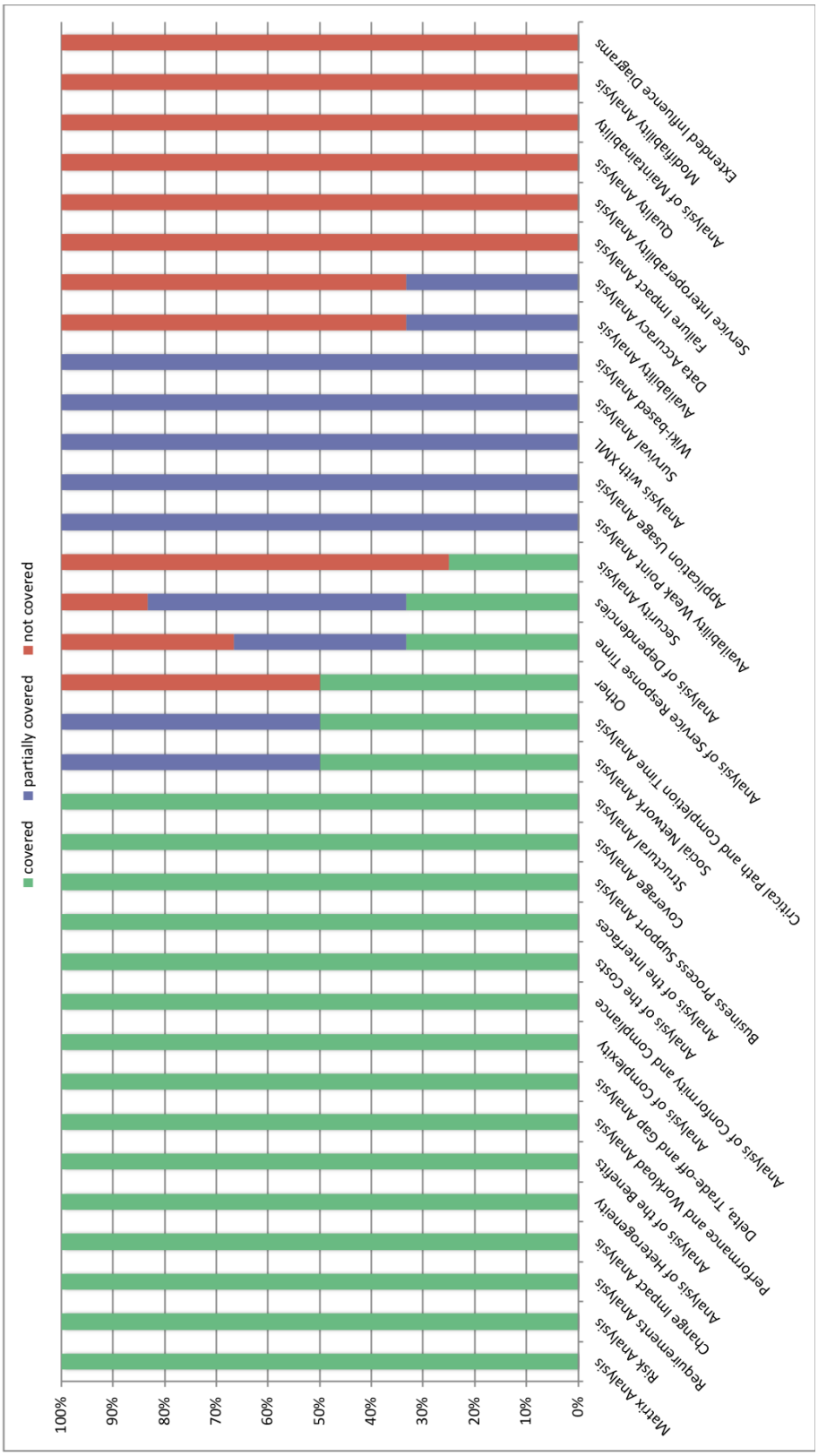


Figure 8.9: Coverage of analysis types in detail.

From the 34 considered analysis types, 21 are covered by the A2F and additional 7 types are at least partially covered. A total of 6 analysis types is not covered. We say that the A2F covers an analysis type, if there exists at least one analysis approach which is assigned to this analysis type and covered by the A2F. If no approach of an analysis type is covered but there is at least one approach that is partially covered, then the analysis type is partially covered. Otherwise the A2F does not cover the analysis type. Within the detailed report in figure 8.9 the amount of covered, partially covered and not covered approaches for each analysis type is provided.

For example 50% of the assigned analysis approaches of the type *Critical Path and Completion Time* are covered by the A2F. The other part is only partially covered. This is the analysis approach described in [Yen09]. The approach utilizes the technique of AHP to combine several measures, among other, the expected time from start to end of a process. A2F does not provide support for AHP itself, but the final combinations of the weighted values can be realized with a metric calculation rule.

The analysis type *failure impact analysis* contains only the approach of Holschke et al. [HNF+09]. They realize the failure impact analysis using Bayesian Belief Networks. This probabilistic technique is not covered within the A2F.

Most of the approaches that are not covered by the A2F, rely on probabilistic techniques. This is illustrated within a further evaluation of the coverage according to the functional and technical dimension proposed in [Rau15]. The coverage of the technical dimension is provided in figure 8.10.

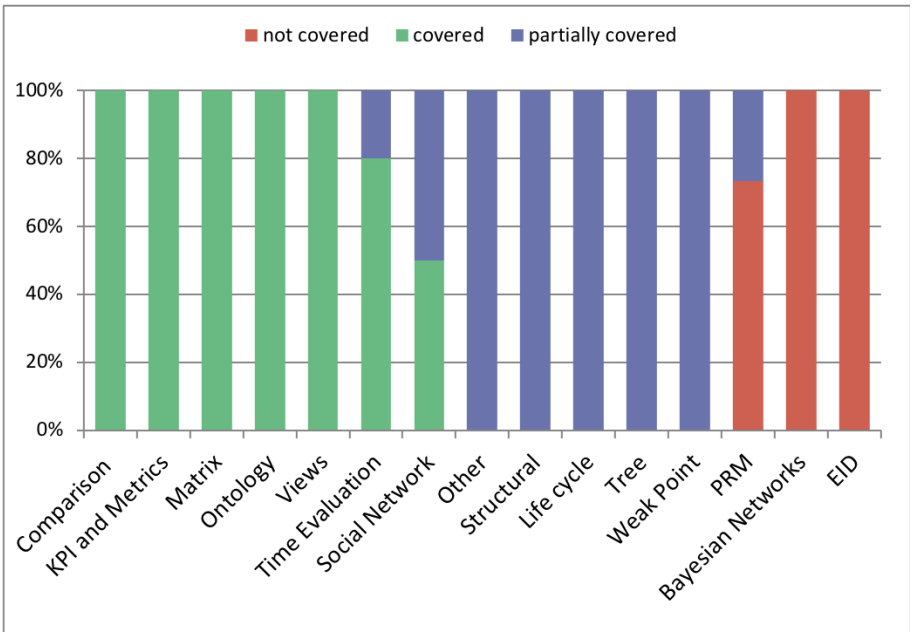


Figure 8.10: Coverage of technical dimensions.

Most of the technical categories are covered or partially covered by the A2F. That means that at least one analysis approach assigned to this dimension can be realized or at least partially realized with the A2F. In figure 8.10 the percentage of covered, partially covered and not covered analysis approaches is illustrated for each technical dimension.

Only the categories containing probabilistic approaches are not covered. These are *Probabilistic Relational Model*, *Bayesian Networks* and *Extended Influence Diagrams*. The

approaches within the technical dimensions *AHP*, *Business Entities* and *Design* were excluded within the evaluation according to the restrictions presented at the beginning of this section. These dimensions are not presented within the figure.

The coverage of the functional dimensions is provided in figure 8.11. Only the dimension *System* is not covered within the A2F. All approaches assigned to this dimension are also assigned to one of the technical dimensions with probabilistic techniques. For all other categories there exist approaches that are covered or at least partially covered (*Data*) by the A2F.

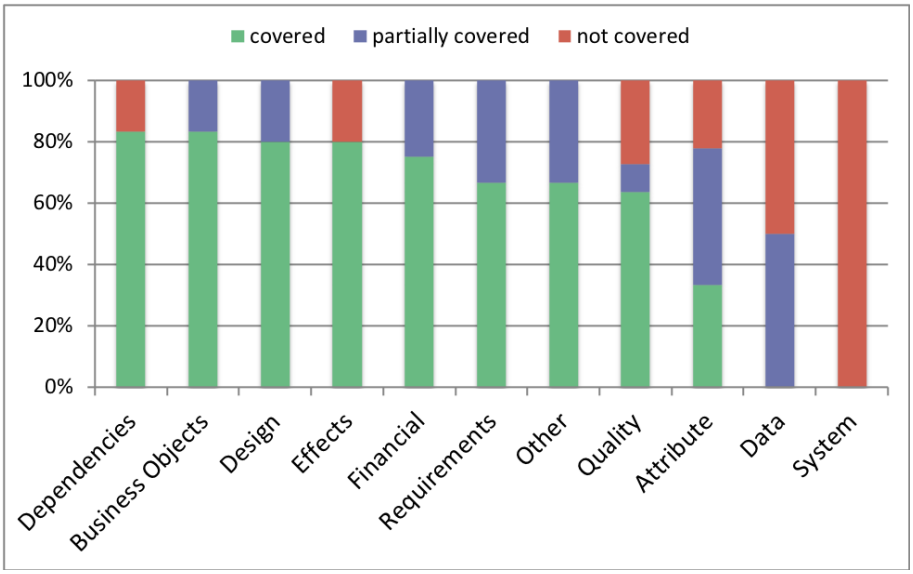


Figure 8.11: Coverage of functional dimensions.

8.3 Scenario-based Evaluation

To assess the applicability of the A2F we utilize a scenario-based evaluation. Based on existing approaches within literature we identified in total 12 different scenarios. The scenarios cover all technical and functional dimensions that are covered by the A2F. Also, each Arla analysis class is utilized at least once.

Only the technical categories `Bayesian networks` and `Extended influence diagrams` are not covered by the scenario-based evaluation, since it is not possible to capture the contained approaches with the A2F. Additionally, the technical dimensions `AHP`, `Business entities` and `Design` are not covered, since all assigned analysis approaches were excluded beforehand due to their scope (see section 8.2). Consequently, no scenario addresses the functional dimension `System`, since we could not identify any approach that is realizable with the A2F.

Table 8.1 provides an overview of the functional and technical dimensions as well as their coverage of the scenarios (S1–S12). If the scenario identifier is provided within brackets, the approach is only partially realized.

Table 8.1: Dimension coverage of the scenarios.

Technical \ *Functional*	*Quality*	*Attribute*	*Dependencies*	*Effects*	*Business objects*	*Design*	*Requirements*	*Financial*	*Data*
Comparison			S1	S1					
Metrics and KPIs	S2		S2				S2	S2	
Matrix	S4	S4							
Ontology					S5	S5			
Views					S9				
Time evaluation	S8								
Social network					S7	S7			
Structural						(S10)			
Life cycle			(S3)						
Tree			(S11)						
Weak point		(S12)			(S12)			(S12)	
PRM		(S6)							(S6)

Additionally a mapping of the scenarios to the Arla analysis classes is provided in table 8.2 Here as well the brackets denote a partial realization of the analysis.

8.3.1 Scenario 1: Change impact analysis

Analysis approach: [dBBG+05]
Technical dimension: Comparison
Functional dimension: Dependencies, Effects
Arla analysis class: Impact analysis

Within the first scenario we established the template for implementing a change impact analysis. The applied analysis approach for the template is presented in [dBBG+05]. The

Table 8.2: Dimension coverage of the scenarios.

Arla analysis class	*Scenario*
Scope analysis (node definition)	(S3), S9, (S10)
Scope analysis (edge definition)	S9
Impact analysis	S1, (S11)
Path analysis	S4, S5, (S12)
Metric	S2, S7, (S10), (S11), (S12)
Performance analysis	S8
Gap analysis	S7
Custom analysis (DFA)	(S6), (S11)
Custom analysis (SPARQL)	(S10)
Composed analysis	S2, S9

authors propose three different change types (remove, extend and modify) that are used to determine the direct and indirect effects of a change event. The respective change type of an element is defined depending on the change type of the immediate neighbors as well as the relation type between them. The authors propose change propagation rules for five different relations types from ArchiMate.

Despite the three change types the authors propose a `signal` to mark elements that are not directly affected by a change but have eventually to be considered. For example the signal is assigned to a process, where the used application service is deleted. In this case the user has to decide about the change effect. For our implementation we interpret a signal in the same way as an extension of an element. The change types can be mapped to the impact types as followed:

high impact:	delete
medium impact:	modify
low impact:	extend

The change semantics for the five relation types can be approximated with the Arla impact analysis template provided in listing 8.1

```
1 Template ChangeImpactAnalysis {
2   "Template for change impact analysis according to [dBBG+05] in the worst case."
3   as Element Attribute
4   defined with impact definition:{
5     WeakEffect   In  (edgeType:"usedBy", edgeType:"realize")
6     StrongEffect In  (edgeType:"access", edgeType:"assign")
7     StrongEffect Out (edgeType:"access", edgeType:"use", edgeType:"realize")
8   }
9 }
```

Listing 8.1: Arla change impact template for scenario 1.

For each relation type in [dBBG+05] a variable is declared and assigned to the best fitting effect type. Since Arla only enables the assignment of three different effect types (strong, weak, no), the actual change semantics as proposed in the paper cannot be captured completely. Especially the behavior in the case of a deletion of an element is concerned of this weakness. Alternatively, it would be possible to implement own DFA propagation rules for this case or to approximate them with the effects type provided in Arla. This can be done through implementing a worst-case and a best-case scenario. The semantics of the `signal` element can also be captured with the approximation of best- and a worst-case.

8.3.2 Scenario 2: Risk and security analysis

Analysis approach:	[IOB06]
Technical dimension:	Metrics and KPIs
Functional dimension:	Quality, Requirements, Financial
Arla analysis class:	Metric, Composed analysis

Innerhofer et al. [IOB06] propose a qualitative approach to combine EA with risk driven security management. Beside the qualitative part, they propose the following supportive reports:

- Number of model elements with status `analyzed = false`
- Number of security requirements of a domain that have status `evaluated = false`
- All security objectives with state `detailed = false`

The reports can be realized with a node set definition in Arla. The node set definition for determining model elements with status `false` is:

```
1 nodeType:"model element"   AND
2 having property propertyType:"analyzed" with value "false"
```

To retrieve the number of those elements, the definition can be wrapped into a `COUNT` statement within a metric. The other reports can be configured by replacing the `nodeType` and `propertyType` with the corresponding values.

Additionally, the authors propose aggregated risk matrices. They depict the risk ratings of the threats for each model element, for each security requirement, each security objective or the whole organization. Exemplary, the realization of the risk matrix for model elements is illustrated. Within the risk matrix, the impact and probability values of the related threats are accumulated. For each impact value (high, medium, low) and each probability value (high, medium, low) an element metric template is defined. The template to determine the number of threats with a low probability value and a low impact value is presented in listing 8.2, lines 1 - 9.

```
1  Template ProbabilityLowImpactLow {
2    "Determine the number of threats with low probability value."
3    as Element Metric
4    defined with calculation rule:
5      COUNT (connected nodeType:"thread" AND
6          having property propertyType:"probability" with value "low" AND
7          having property propertyType: "impact" with value "low");
8      for types (nodeType:"model element")
9  }
10 Template ProbabilityLowValues {
11   "Compose the aggregated probability values."
12   as Element Metric
13   defined with composition rule:
14   combine ProbabilityLowImpactLow and ProbabilityLowImpactMediumAndHigh
15 }
16 Template MatrixValues {
17   "Compose the aggregated probability values."
18   as Element Metric
19   defined with composition rule:
20   combine ProbabilityLowValues and ProbabilityMediumHighValues
21 }
```

Listing 8.2: Arla metric templates for scenario 2.

For the medium and high probability a respective element metric has to be defined, too. And additionally, for each impact type. In total, nine element metrics are created to provide the aggregated threat values for each model element. The metric templates are combined within several Arla composition definition. For example, the `ProbabilityLowValues` template (lines 10 - 15) combines aggregated values for medium and high impact threats with those for low impact threats, where in all cases the threat probability is low. The template definition providing the final result for matrix creation is provided in lines 16 - 21. Therein, the results for the low probability values are merged with those for medium and high probabilities.

Executing the `MatrixValues` template with A2F would provide an `AttributeValueResultMap`. For each element the respective metrics, e.g. `ProbabilityLowImpactLow`, `ProbabilityLowImpactMedium` and so on, are provided to finally create the matrix.

8.3.3 Scenario 3: Analysis of dependencies

Analysis approach:	[Saa10]
Technical dimension:	Life cycle
Functional dimension:	Dependencies
Arla analysis class:	Scope analysis (node)

Within scenario three, we provide the templates for the realization of the time-related dependency analysis proposed in [Saa10]. The authors extend EA elements with properties indicating their status (current, planned), their lifecycle (planned, active, retired), their duration as well as with time stamps indicating their start and end. The properties are considered during the subsequent dependency analysis. In specific, the relations between applications and processes are visualized with respect to a specific point of time.

This analysis can be realized with a scope analysis. Discrete values like `planned`, `active` and `retired` are completely supported. For considering time stamps only indirect support is provided, thus the A2F covers this approach only partially. Since the A2F does not provide a data type `Date`, it has to be represented as integer. Also the date values within the model have to be present as integer values. In this case, a time-related process application view can be defined with the template provided in listing 8.3.

```
1 Template TimeRelatedView {
2   "Time-related dependency view for processes and applications."
3   as Aggregate Modelelementset
4   defined with set definition:
5   (nodeType: "application" AND
6        having property propertyType:"active" with value (< 20190201)) OR
7   (nodeType: "process" AND
8        having property propertyType:"active" with value (< 20190201))
9 }
```

Listing 8.3: Arla scope analysis template for scenario 3.

This scope analysis determines a view which only contains elements that are active at the requested point of time '*01.02.2019*'. It is also possible to add an additional constraint for `retired` property, to ensure that the element is not retired yet.

Additionally, the author introduces *projects* to aggregate changes to model elements within a defined time span. Using the relation conditions in Arla, it is possible to solely focus on elements assigned to one specific project or to validate their time related properties.

8.3.4 Scenario 4: Analysis of conformity

Analysis approach:	[Nie06]
Technical dimension:	Matrix
Functional dimension:	Quality, Attribute
Arla analysis class:	Path analysis

[Nie06] identifies the following questions that should be answered with conformity analysis:

- Are there provisions for the definition of EA elements like business processes?
- Are there best practices for processes like recovery, backup, authorization, authentication defined?
- Are important processes been drafted (e.g. deployment procedures)?
- Are the deployed infrastructure components internally certified?
- Are the applications designed according to the reference architecture models?
- Are there development procedures and tools in use that are not included in the infrastructure standards?

The first four questions can be answered within Arla with property conditions, assuming that the information is present in the EA model. Additionally, Arla can be used to verify the adherence to the provisions, best practices, standards and reference architecture.

More specific, [Nie06] proposes a heterogeneity analysis to identify the elements within the architecture that do not adhere to the reference architecture. Therefore, the utilized technologies are provided within a product process matrix. The product process matrix can be established with two path analyses. The analysis results are used to create a support map with products and processes on the two axis and the respective technology components in the center. The technology components can either be realized as concrete element or they are the property of another element, e.g. an application component. Listing 8.4 provides the path analysis template for the localization of the components on the product axis.

```
1  Template ProductComponentPaths {
2    "Provide the paths to locate the components within the matrix on the product
         axis."
3    as Aggregate Pathset
4    defined with path definition: {
5      path type AllPath
6      SourceStereotypes (nodeType:"product")
7      TargetStereotypes (nodeType:"technology component")
8      IncomingEdges (class:StructualDependentOf)
9      OutgoingEdges (class:ConsumedBy, class:Provide, class:LocatedAt, class:
           StructualDependentOf)
10   }
11 }
```

Listing 8.4: Arla path analysis template for scenario 4.

The all path analysis mode is chosen and source and target elements of the paths are *product* and *technology component*. The considered edge classes for path determination are incoming and outgoing `structural dependent of` relations as well as outgoing `consumed by`, `provide` and `located at` relations. A respective template has to be created to localize the components on the *process* axis.

Figure 8.12 illustrates the construction of the product process matrix, based on the result

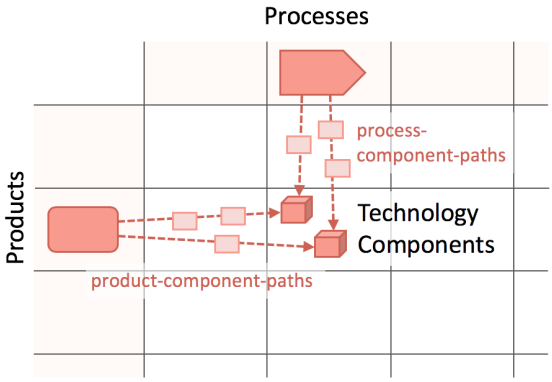


Figure 8.12: Exemplary entry of the product process matrix for conformity analysis.

of the two path analyses. Depending on the actual EA model, the technology components already represent the demanded technologies or they have a property determining it. In the second case, only the property is presented within the matrix. The so created matrix supports the identification of architecture parts that do not conform to the defined reference architecture models.

8.3.5 Scenario 5: Structural analysis

Analysis approach:	[LB09]
Technical dimension:	Ontology
Functional dimension:	Business objects, Design
Arla analysis class:	Path analysis

The structural analysis proposed by [LB09] focuses on specific elements and relations that are used within views. The views enable the assessment of the enterprise performance as well as its structure from different perspectives. Examples for proposed analyses and the corresponding views within [LB09] are:

- *Aggregation-disaggregation analysis*: Organizational view, process view, product family view
- *Market analysis*: Product customer view, product market segment view
- *Pareto analysis of resources or actors*: rank products in terms of contribution to revenue, profit, or units sold
- *Outsourcing analysis*: Activity resource view, activity actor views
- *Reuse analysis*: Product people view
- *Value driver analysis*: Product KPI view

Most of the views are visualized as a two-dimensional matrix. Within the cells an 'x' indicates a dependency between the entry of the row and the entry of the column. Alternatively, a number indicates the number of dependencies. An example result for such a view is provided in figure 8.13. The rows represent products and the columns people that perform activities within the processes. In the cells the number of activities the people perform for the product is provided. Within the underlying metamodel products are directly related to activities, as well as a person is also directed related to an activity [LB09].

Within the A2F we can utilize the path analysis to determine the required information

ResourceName	Barista	Cashier
Finished Cup of Coffee	2	3
Finished Espresso Drink	1	2
Ready Pastry		3

Figure 8.13: Exemplary result for the product people view [LB09].

to fill the matrix. Therewith it is possible to abstract from the meta model, and a direct relation is no longer required. Listing 8.5 provides exemplary one path configuration for the creation of the product people view.

```
1 Template ProductActivityPaths {
2   "Provide the paths to locate the activities within the matrix on the product
        axis."
3   as Aggregate Pathset
4   defined with path definition: {
5     path type AllPath
6     SourceStereotypes (nodeType:"product")
7     TargetStereotypes (nodeType:"activity")
8   }
9 }
```

Listing 8.5: Arla path analysis template for scenario 5.

This template determines the path set that is required to locate the activities according to rows. Since we do not provide further restrictions about the relation types, all possible paths of a given maximum length are determined. If desired, the relation type can be further restricted. An additional template is required to locate the activities according to the participating persons. Instead of visualizing the concrete activities within the matrix, their total number is counted per cell and displayed.

8.3.6 Scenario 6: Data accuracy analysis

Analysis approach:	[NBE14]
Technical dimension:	Ontology
Functional dimension:	Business objects, Design
Arla analysis class:	Custom analysis (DFA)

[NBE14] propose a data accuracy analysis which estimates the accuracy of the data sets provided in the EA model. The analysis relies on services that are related to data sets, as well as on internal behavior elements, i.e. functions that accesses the data sets. The authors assume that a data set contains N elements. A specific part of these elements is accurate, the other part is inaccurate. This ratio may change if a service or function processes the data set. For each service and each function, its correction rate as well as its deterioration rate is provided. The current data set accuracy, i.e. the accuracy at time $t+1$, can be defined with these two factors as well as the input data accuracy (time t).

This definition can be captured with DFA propagation rules. Thereby, the behavior of the accuracy of the data set is simulated according to the presence of respective processing or access relations.

```
1 @Rule(modelClass=ModelNode,attribute=accuracy)
2 Object data_accuracy(ModelNode node) {
3   int currentAccuracy = node.getPropertyValue(parameters.getInitialAccuracyUUID(),-1);
4   if(currentAccuracy < 0) return currentAccuracy
```

```
5    for(ModelEdge modelEdge : node.getIncoming()){
6      int correctionRate = modelEdge.getSource().getPropertyValue(parameters.getCorrectionUUID(),-1)
7      int deteriorationRate = modelEdge.getSource().getPropertyValue(parameters.getDeteriorationUUID(),-1)
8      if(correctionRate == -1 || deteriorationRate -1) return currentAccuracy
9      currentAccuracy = currentAccuracy(1-correctionRate) + deteriorationRate (1-currentAccuracy)
10   }
11   for(ModelEdge modelEdge : node.getOutgoing()){...}
12   return currentAccuracy
13 }
```

Listing 8.6: DFA propagation rule for implementation of scenario 6.

Each `ModelNode` is extended with an attribute `accuracy` that is determined with the rule in listing 8.6. The rule provides only a valid result, if the model node has a respective `initial accuracy` property (lines 3, 4). This property identifies an element as data set. The accuracy of the data set is calculated according to the related processing elements. Therefore the incoming (lines 5 - 10) as well as the outgoing (line 11) relations are processed. If the source, respectively target element, has a valid property for the correction and deterioration rate (lines 6 - 8), the accuracy is updated (line 9).

In [NBE14] it is also possible to add uncertainty conditions for elements, relations and properties. They are evaluated with a Monte Carlo simulation and provide further information about the quality of the result. Since this is not possible with the A2F, this analysis approach is only partially covered.

8.3.7 Scenario 7: Social network analysis

Analysis approach:	[KMP11]
Technical dimension:	Social network
Functional dimension:	Quality
Arla analysis class:	Metric, Gap analysis

[KMP11] propose the extraction of social network data to create a graph of the employees of an organization. The edges within the graph are defined by the sent and received emails. From this graph, several characteristics can be derived to assess the organization and communication structure. Examples are the total number of edges, the average number of edges per node or the range of incoming or outgoing edges. These characteristics can be captured using metrics in the A2F. For example, the template definition provided in listing 8.7 provides the number of incoming edges per node.

```
1 Template IncomingEdgeDegree{
2   "Determines the number of incoming emails per employee"
3   as Element Metric
4   defined with calculation rule:
5     COUNT(incoming edgeType:"emails");
6     for types (nodeType:"employee")
7 }
```

Listing 8.7: Arla metric template for scenario 7.

With a subsequent scope analysis those processes can be identified, whose participants communicate less via emails. They are potentially interesting for optimizations, since the small number of emails could be a hint for much paper work [KMP11].

Additionally, it would be interesting to compare the employee list, according to the organigram of the organization, with the retrieved employees from the graph. This analysis

identifies persons that are either actual employees, but do not send any emails, or are no longer employed at the organization. The difference can be captured with a gap analysis in the A2F (see listing 8.8).

```
1 Analysis GapAnalysis {
2   "Comparison of the communication employees with the organizations organigram"
3   as Element Attribute
4   defined with gap configuration:
5   base model "http://organization/organigram"
6   target model "http://organization/communicationGraph"
7   calculate Differences
8 }
```

Listing 8.8: Arla gap analysis definition for scenario 7.

The gap analysis determines the status `unaffected` for each employee that is present within the organigram and the communication graph. Employees only present within the communication graph are identified as `new` and employees only in the organigram as `deletion candidate`. Either they do not write emails or they already left the organization, but the organigram was not updated.

8.3.8 Scenario 8: Performance and workload analysis

Analysis approach:	[JI09]
Technical dimension:	Ontology
Functional dimension:	Business objects, Design
Arla analysis class:	Performance analysis

The performance and workload analysis provided in [JI09] is completely captured with the Arla analysis class `Performance analysis`. For the definition of the DFA propagation rules the proposed procedure is used. The definition and implementation of the analysis is described in sections 4.3.5 and 5.5.7.

8.3.9 Scenario 9: Business process support analysis

Analysis approach:	[SK11]
Technical dimension:	Views
Functional dimension:	Business objects
Arla analysis class:	Scope analysis (edge, node), Composed analysis

[SK11] present a formalization of viewpoints, required for business process support analysis. Within the A2F, we can implement the formalized definitions and enable the generation of the respective views using the scope analysis.

For example the elements Elt and the relations Rel within the business flow viewpoint ($BPFV$) are defined with the following two definitions:

(1) $Elt(BPFV_{AnBP}) = \{x | x \in AnBP \cup SP_{AnBP}\}$

(2) $Rel(BPFV_{AnBP}) = \{(x,y) | (x,y \in AnBP \cup SP_{AnBP}) \wedge ((x,y) \in Triggering)\}$

$AnBP$ is the set of analyzed business processes and SP_{ANBP} is the set of all behavior sub elements of the analyzed business processes. These sub elements can be of the type sub process, business function, business event, or business interaction and have to be related

to the business process with an aggregation of composition relation. The set of analyzed business process is represented in the A2F through the set of selected start elements.

For the implementation of the viewpoint, two different scope analysis templates have to be provided (see listing 8.9). The first one (lines 1 - 9), restricts the elements according to the presence of a relation to one of the start elements. The second one (lines 10 - 15) further restricts the set according to the allowed element types.

```
1  Template BusinessFlowRelationRestriction {
2    "Provides the relation restriction for the business process flow viewpoint."
3    as  Aggregate Modelelementset
4    defined with scope definition: {
5      ModelEdge in: None out: None
6      BehavioralDependentOf in: Single out: Single
7      StructuralDependentOf in: None out: Single
8    }
9  }
10 Template BusinesFlowElementRestriction {
11   "Provides the element restriction for the business process flow viewpoint."
12   as Aggregate Modelelementset defined with set definition:
13     nodeType:"business process" OR nodeType:"business function" OR
14     nodeType:"business event" OR nodeType:"business interaction"
15 }
16 Template BusinessProcessFlowViewpoint {
17   "Determines the flow viewpoint for a set of selected processes."
18   as Aggregate Modelelementset defined with composition rule:
19     combine BusinessFlowRelationRestriction and BusinesFlowElementRestriction
20     with operation INTERSECTION
21 }
```

Listing 8.9: Arla gap analysis definition for scenario 9.

The elements of the final business process flow viewpoint have to be present within both results. Thus, these templates are composed according to the composition rule in the `BusinessProcessFlowViewpoint` analysis (lines 16 - 21). This analysis composes both results according to the `intersection` operation.

8.3.10 Scenario 10: Wiki-based analysis

Analysis approach:	[BMNS09]
Technical dimension:	Structural
Functional dimension:	Design
Arla analysis class:	Scope analysis, Custom analysis (SPARQL)

[BMNS09] propose the usage of wikis for the documentation, the information communication and analysis of EA models. The wiki enables traceability of decisions through its inherent versioning approach. Differentiating between minor and major changes restricts the amount of information that has to assessed by the user. Within Arla, property conditions of a scope analysis can be used to filter elements according to one of these conditions. Assuming that the respective information is accessible in the model. The scope analysis is also used to determine, for example, all elements tagged with *business process*.

Furthermore, an analysis of the property usage within the wiki is proposed to support the creation of a consistent information model. This can be implemented in the A2F with metrics by providing the usage ratio for each property.

Finally, [BMNS09] propose the comparison of different versions of a wiki page for an element. Since the A2F does not provide initial support for time related analysis, this is

only partially covered. A custom SPARQL query can be used to implement this analysis. For example, the query provided in listing 8.10 provides the values of all properties of an element within two different versions.

```
1  SELECT ?element ?propertyStereotype ?valueNew ?valueOld
2  FROM <currentVersionUri>
3  WHERE {
4    ?newElement    gmm:uuid          ?uuid
5    ?newElement    model:property    ?newProperty.
6    ?newProperty   model:stereotype  ?propertyStereotype.
7    ?newProperty   model:value       ?valueNew.
8    GRAPH <oldVersionUri> {
9      ?oldElement  gmm:uudi  ?uuid.
10     OPTIONAL {
11       ?oldElement    model:property    ?oldProperty.
12       ?oldProperty   model:stereotype  ?propertyStereotype.
13       ?oldProperty   model:value       ?valueOld.
14     }
15   }
16 }
```

Listing 8.10: Custom SPARQL query for scenario 10.

We assume that the different versions of the model are represented with different named graphs. The SPARQL query retrieves all properties of an element in the current model. Additionally, it identifies the respective element within the previous version and optionally provides the corresponding property values. If this property was not defined in the old version, the element and the new value are anyway part of the result. Finally, the A2F provides an `AttributeValueResultMap`, where for each element an entry for the property stereotype, the new value and the old value is provided.

8.3.11 Scenario 11: Analysis of dependencies

Analysis approach:	[FFJ09]
Technical dimension:	Tree
Functional dimension:	Dependencies
Arla analysis class:	Impact, Metric, Custom analysis (DFA)

[FFJ09] propose the utilization of Fault Tree Analysis and Bayesian networks for dependency analysis within EA models. Within in the A2F, we are not capable to perform the dependency analysis on this level of detail. Nevertheless, it is possible to approximate the result using the impact analysis. Custom DFA propagation rules can be used to provide more detailed results including a quantification of the impact.

For example, the authors determine the impacts of computer failures and the absence of an employee to the process quality. The effects on the process quality are represented within a conditional probability matrix. Within the A2F, we approximate this with the discrete impact types `high`, `medium` and `low`. For example, a high probability of a computer failure has a `high` impact on the process quality.

Dependencies between the impact events can also be described with an element metric. For example, a computer failure is dependent on an HDD failure, a screen failure and a power outage. The probability of a computer failure can be defined with the Arla template in listing 8.11. The probability of a `powerOutage` is independent from the element and determined within another aggregated metric template.

```
1  Template ComputerFailureProbability {
2    "Determine the number of threats with low probability value"
3    as Element Metric
4    defined with calculation rule:
5      (propertyType:"HDD failure probability" * propertyType:"screen failure
           probability") * powerOutage ;
6    for types (nodeType:"computer")
7  }
```

Listing 8.11: Arla metric template for scenario 11.

8.3.12 Scenario 12: Availability weak point analysis

Analysis approach: [LLP+09]
Technical dimension: Weak point
Functional dimension: Attribute, Finance, Business objects
Arla analysis class: Path analysis, Metric

[LLP+09] propose a methodology for the analysis of availability weak points in deployment framework of service-oriented architectures. Based on the relationships between business workflows and IT resources an optimal high-availability recommendation is calculated. The analysis, especially the optimization part, is not supported within the A2F. But it is possible to implement the dependency elicitation as well as the calculation of the availability measures. The different availability measures for workflow, application and IT resources can be captured with an element metric.

The dependencies between the business workflows to the IT resources can be captured with the path analysis provided in listing 8.12. Therein the services, the workflow is composed of are considered as well as the applications they use and the resources these applications are assigned to.

```
1  Template BusinessWorkflowResourcePath {
2    "Paths form a business workflow to the IT resources."
3    as Aggregate Pathset
4    defined with path definition: {
5      path type CustomPath
6      SourceStereotypes (nodeType:"Business workflow")
7      TargetStereotypes (nodeType:"IT resource")
8      Incoming(edgeClass:ConsumedBy)
9      Outgoing(edgeClass:StructuralDependentOf,edgeClass:LocalizedAt)
10   }
11 }
```

Listing 8.12: Arla metric template for scenario 12.

8.4 Case Studies

Next to the scenario-based evaluation we employed the A2F within three case studies. This was done in context of the two use cases weak point identification and EA planning, described in chapter 6 and chapter 7. The case studies are used to evaluate to the following issues:

Evaluate the following issues:

- Applicability of use case
- Flexibility and adaptability of the A2F
- Relevance

The applicability of the use cases is evaluated in terms of the out-of-the-box usage of the provided templates within the case studies. Thereby, the generic applicability of the templates is assessed as well as the quality of the provided results. If necessary, the scope of adaptions to the templates is documented.

The flexibility and adaptability of the A2F is assessed through evaluating the effort for EA data import. Also the possibility to define new custom analyses to cope with the specific information demands in the case study is considered.

Finally, the usefulness of the retrieved results is considered to evaluate the relevance of the artifacts. That means, it is assessed whether it was possible to determine weak points and if the change process could be supported. The effort and the results provided by the A2F are compared to the typical effort and results within a manual procedure. The manual procedure is the typical approach, if no or only insufficient tool support is available.

Within section 8.4.1 the EA planning use case is applied to a medium-sized software product company. Sections 8.4.2 and 8.4.3 provide two case studies realizing the weak point use case. One is a microservice service landscape within a logistic company and the other one a backend service landscape providing customer services within the automotive sector.

8.4.1 Case study 1: EA planning

We applied the developed approach for scenario evaluation within a case study of a medium-sized software product company. The company wants to shift the product delivery to a Software-as-a-Service (SaaS) model. The transformation is divided into three main activities which result in three different planning scenarios: Cloud Business Management, SaaS Operational Support and SaaS Security and Policy. The scenarios have a size between only 12 architectural elements and over 80 architectural elements.

The complete manual procedure for EA planning was described within the master thesis [Gra15]. The master thesis provides a description of the current architecture, the desired target scenarios as well as further planning steps like gap analysis and identification of transformation hot spots. The master thesis was supervised by the author of this thesis. For development and analysis of the three different scenarios, the following steps were carried out:

1. Document the current architecture
2. Develop planning scenarios
3. Execute transformation analysis

- Gap analysis
- Transformation analysis
- Future actions

Within gap analysis, the current architecture and the developed target scenarios are compared with each other based on the contained elements. The differences are outlined and for each element it is decided, whether it is included, impacted, new or eliminated. Additionally, the effects of the changes are discussed and potential ripple effects on direct and indirect related elements are considered.

Within transformation analysis, the different changes that will be performed during the scenario are further assessed. A hot spot provides an aggregation of several changes being part of one topic area. The transformation hot spots are merely defined according to the identified key process flows. These flows provide a view on the target architecture that is related to one process and its affected events, functions and products.

Finally, to decide about future actions, the transformation hot spots are evaluated in terms of potential preconditions and time constraints.

For the evaluation, we automated the steps proposed within the master thesis by utilizing the proposed concepts in chapter 7. Additionally, we compared the generated domain models with the manually created ones.

Applicability We implemented the proposed EA planning process for each scenario. Within the first step the gap analysis was performed between the current architecture and the proposed planning scenarios. Figure 8.14 provides the result for the cloud scenario. In total eight elements of this scenario were identified as new elements (green ones). The remaining elements were identified as affected (blue ones). In this planning scenario no unaffected elements were identified. This was the same for the other two planning scenarios. They only contain new and affected elements.

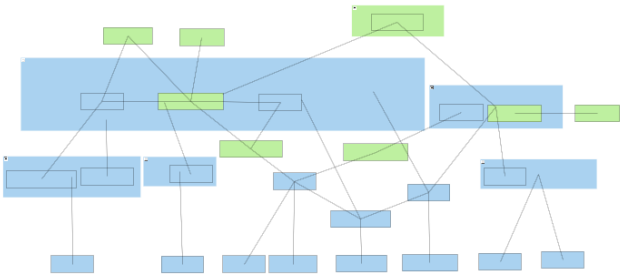


Figure 8.14: Gap analysis result for the cloud scenario.

Subsequently, the domain architectures were generated. The defined analysis templates could be applied to all three scenarios without adaption. We also compared the generated domain architectures with the manually created ones. In all cases the determined domain architecture contains more elements than the manually created model. This is due to the absence of unaffected elements within the target scenario model. We observed that the rules for the generation of the domain architecture have to be adapted for the largest scenario. Otherwise the generated target domain would be too large.

Finally, we verified the scope of the domain architecture with an impact analysis. Figure 8.15 provides an example of the result for a worst-case impact analysis of the deletion

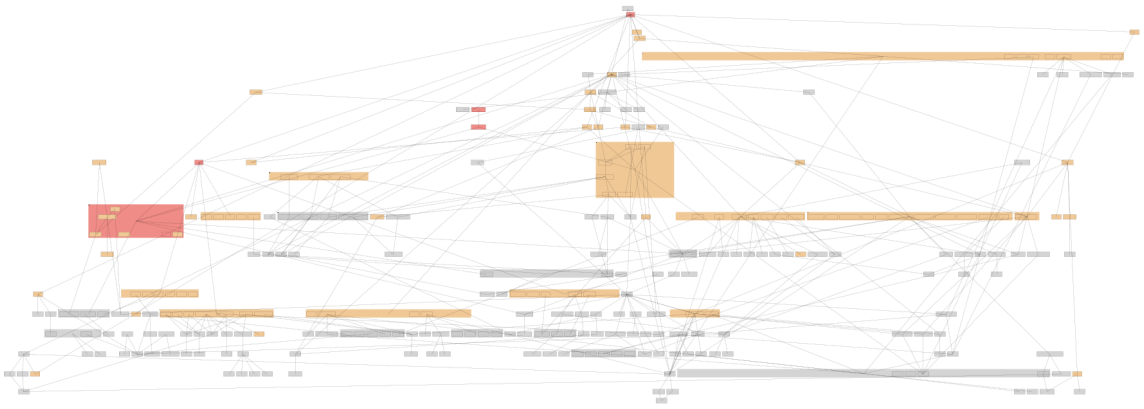


Figure 8.15: Impact analysis result for the target architecture in the cloud scenario.

of one process. The result is visualized for the whole target architecture. According to this result, the domain architecture was extended with further elements. The same occurs for the smallest scenario. Here as well, the domain architecture was further extended based on the result of the impact analysis. Mainly elements that use affected elements of the planning scenario were added.

Within the large scenario, SaaS Operational Support, more than 70% of the contained elements were identified as affected, the other 30% as new elements. After further consideration of the model and the execution of an impact analysis, several of the affected elements were identified as unaffected ones. Based on these findings, several elements were excluded from the domain architecture to enable a better manageability of it.

Finally, the target domain architectures were evaluated through view generation. The utilized key process views within the manual planning procedure can be created automatically with Arla scope analyses. In figure 8.16 the key process view for the cloud business management is presented. Thereby, we observed that the manual created views are not complete. For example, within the Cloud Business Management scenarios a process was overseen. Especially in the large projects, the subsequent view generation was essential to assess the quality of the proposed scenarios and to determine the transformation hot spots and future actions.

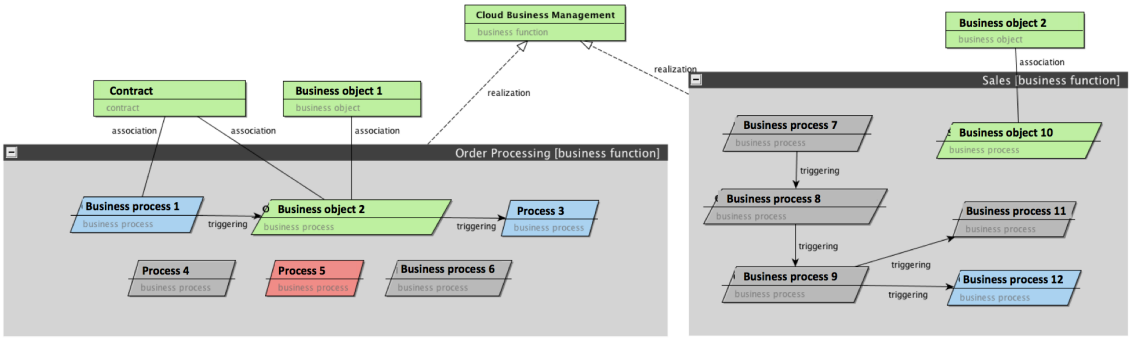


Figure 8.16: Key process view for cloud business management.

Flexibility and adaptability The provided templates for scenario evaluation enabled a quick and easy application of the proposed method. During the evaluation process we identified needs for further customization of the proposed templates based on the retrieved results. These demands could be realized with the available configuration options of the Arla analysis classes.

Additionally, the proposed procedure for scenario evaluation could be mapped into the approach performed within SaaS transformation planning. The first steps including gap analysis, definition of the domain architecture and verification of the change effects were performed manually before. Within the last step, the actual evaluation, the A2F provides enough functionality to support the identification of hot spots and future actions. This case study does not include a quantification of the scenarios. Our proposed planning process as well as the A2F provide the required flexibility and adaptability to capture the procedure of the case study.

Relevance Comparing the tool-supported artifacts with the manually retrieved ones, the effort for their creation is significantly smaller. During the evaluation, several inconsistencies, especially missing elements, could be identified within the available scenario architectures. The automated view generation supports the overall understanding for the different scenarios.

Utilizing the predefined templates provides a fast first feedback regarding the relevant domain and potential impacts. Within the first iteration, the manually created domain architecture provides a higher quality. Based on the first feedback, more precise analyses were defined. The generated domain architectures and determined impacts of these analyses provide more suitable results. These analysis results uncovered inconsistencies within the manually created models. The final view generation for the key processes was done with custom scope analyses.

Summarizing, human input is still required during the whole planning processes. The quality of the results of the automated analyses depend to a large extend on the user. Nevertheless, it is still less effort than without analysis support. Due to the fast creation of views, the user tends to create several different ones. This increases his understanding of the scenarios and decreases the probability of overlooked effects.

8.4.2 Case study 2: Weak points regarding microservice characteristics

We applied the weak point use case within a service landscape consisting of about 50 services at a large logistic company. In a first step, the model quality was assessed, and in a second step, the architecture was considered regarding the identified microservice principles. The results of the weak point analysis were also discussed with the developers and architects of the services.

The required architecture model was retrieved from the communication data between the services. One data reporter was implemented to capture the data from the middleware RabbitMQ [Piv07]. Another reporter implements the OpenTracing API [Ope19a] and provides the synchronous communication data.

The data for the case study was retrieved during a system test. Thereby, 45 microservices were identified which were clustered in 9 groups. 3 466 synchronous calls could be aggregated to 14 different communication dependencies and another 49 847 asynchronous calls to 57 dependencies. The major question within this case study was the coherence of

the landscape to the microservice principles. Nevertheless, we performed a model quality assessment beforehand.

For evaluating the quality of the retrieved model, three different metrics were employed:

(1) Number of services not assigned to a services group

(2) Number of services without usages

(3) Adjusted number of total services

Within the adjusted number of total services, services without usages and without a service group are ignored.

Six services were identified that have no assignment to a service group. These services represent mocks that are used during the system test for the simulation of external systems. The six services are not part of the actual service landscape. These services were kept within the system, since the mock services only have one dependency to an internal service and thus, have no significant effect on the later analysis results.

Additionally, five services without usage dependencies were identified and excluded for later analyses. The adjusted number of services is 34. Services which are not part of the system test are not contained within the data set. At least one missing service could be identified. Summarizing, the quality of the model was observed as good by the developers and architects.

The specific requirements regarding microservice assessment are all fulfilled by the model. There exist elements that can be mapped to utilized variables `service`, `interface` and `realization`. As well as there exists a property at `interfaces` with values `synchronous` or `asynchronous`.

The metrics for microservice assessment were nearly all executed and the results are discussed with an architect of the development team. In figure 8.17 the evaluation of the metric 'Number of asynchronous dependencies' is presented. According to the number of dependencies, the services are colored with green, yellow and red.

Applicability The provided templates in chapter 6 could be utilized for assessing the model quality as well as the microservice characteristics. The services without usages and the services without service groups were identified through adapting the `Missing-RelationMetric` (see listing 6.3). Adapting the template in listing 6.4, no endpoints could be identified that are realized by more than one service.

Evaluating the requirements of the microservice assessment templates provides shortcomings of the model regarding the metric M2, measuring security similarity, and M8, the utilization degree of technologies. Since no respective properties are available, these metrics cannot be applied in this case study. The remaining templates could all be adapted to the model of the case study.

Flexibility and adaptability The adaption procedure for Arla templates was sufficient for the case study to assess the microservice quality. To evaluate the model quality, predefined templates were used as well as individual Arla analyses were defined. Since the data was derived from a system test, not the complete productive system was represented. In this specific case, it was important to compare the adjusted number of services with the target value. This was done with a new metric definition.

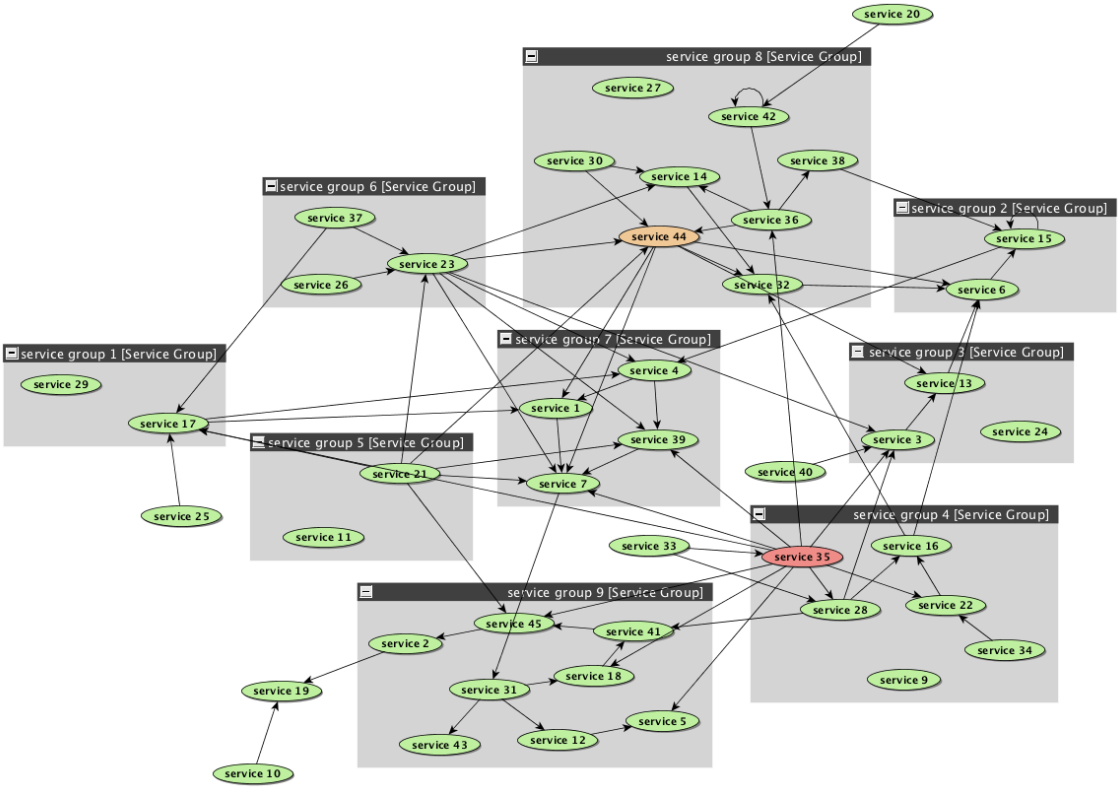


Figure 8.17: Visualization of the number of asynchronous dependencies.

Relevance The metric results advance the discussion about the architectural design in the team. The result for the metric 'asynchronous dependencies' (provided in figure 8.17) identifies one service that does not fulfill the principle of loosely coupled services. The service marked red in the figure has a high number of asynchronous dependencies. Together with its name (originally *timetable*), this indicates a violation of the principle domain-driven-design.

Further implications from the evaluation were a missing data caching at a microservice and a missing service in the system test. These issues were not known to the team before. All over, the positive outcome of the metric evaluation confirms the perception of the team having a quite good architecture.

The beforehand model quality assessment identifies those metrics templates which are not applicable within this case study. The developers and architects can rely on the provided metrics, since the data foundation is sufficient. The missing service, identified within the adjusted number of services, will be included into system test in the future.

8.4.3 Case study 3: Weak points of a backend service landscape

The third case study deals with a large backend system providing customer services. The service landscape comprises clusters which represent the provided products and applications which are a logical aggregation of services. Services provide their functionality via interfaces to other services. The landscape is characterized by a large heterogeneity and the target vision of independent clusters is out of reach. In total the landscape com-

prises 17 clusters, 148 applications, 458 services and about 650 usage dependencies (see figure 8.18). An additional integration of the communication data from the Application Performance Monitoring tool Dynatrace [Dyn19a] provides an incomplete list of databases, their accesses by services and further communication dependencies between the services.

Within the case study a major demand was the model quality. Currently, the documentation of the service landscape was done in Excel files. Especially the usage dependencies, but also information about the single elements is outdated and incomplete. Afterwards, the model is used to assess the microservice characteristics as well as to identify weak points according to the specific demands of the organization.

The model quality was determined with several template adaptions as well as further individual analysis configurations. Thereby, a large amount of isolated services, i.e. services without a usage dependency, were identified. According to the architect, those 128 services are either retired or the usage dependencies are not modeled. In all cases, this is due to a bad quality of the model. There exists no service within the landscape without at least one usage dependency. An additional metric determines in total 110 internal services that are not assigned to a logical application. This too indicates the bad quality of the model.

Afterwards, the applications that are already marked as retired or as active-deprecated are considered. From the three retired applications, one is still in use. This is also not the case in the productive system and is due to outdated data. Additionally, eight active-deprecated applications are also still in use. Either this is due to bad model quality as above, but it could also point out weak points of the architecture. The services relying on the functionality of those applications should be considered and eventually adapted.

Finally, the quality of the available properties was determined. Thereby, the properties are checked for missing ones as well as for empty string fields or `na` values. About 20 interfaces have no valid entry for either transfer security and/or encryption.

With regard to the subsequent microservice assessment, the properties determining the message size and call frequency are considered. In both cases no numeric values are provided, as demanded within the Arla templates. Within the model only discrete values are

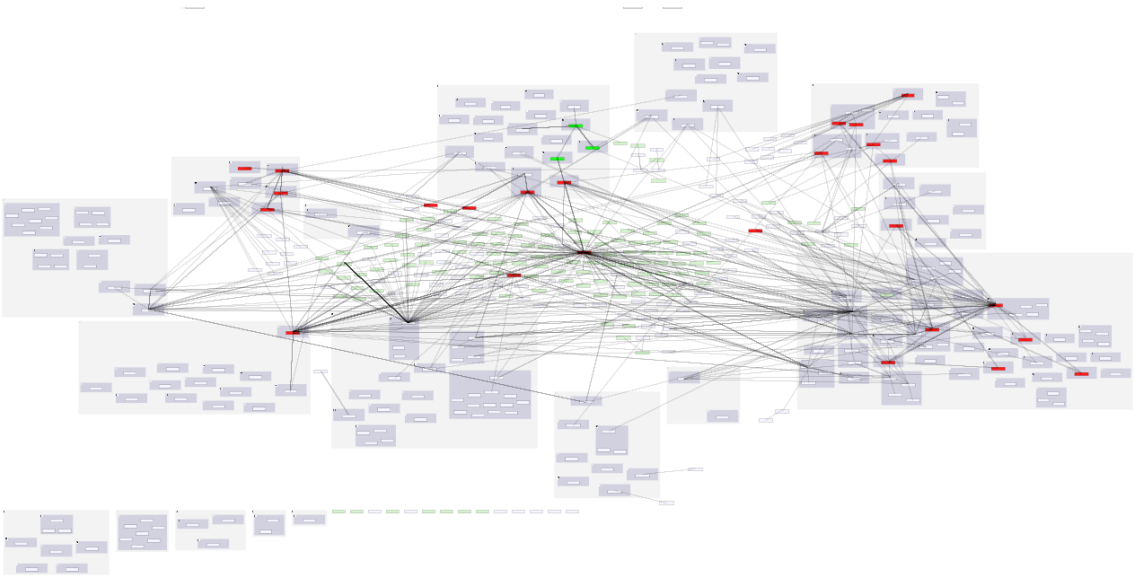


Figure 8.18: Service landscape with a visualization of the synchronous cycle result.

provided. Additionally, the values to determine asynchronous and synchronous communication are provided with different keywords.

The majority of the microservice templates can be applied to the model. However, the validity of the results is low due to the bad model quality. We identified two synchronous dependency cycles within the model (see figure 8.18 the red and green elements). In contrast to the previous case study, we were able to employ the security similarity metric and the utilization degree for a technology. For the later one the SOAP percentage is determined which is currently at 11%. In the first case, only 38 services have different requirements at their interfaces regarding transfer security and encryption. The visualization of this result is provided in figure 8.19. The red services are those ones where the determined ratio is neither 1 or 0, i.e. there exist different security specifications at the interfaces of this service.

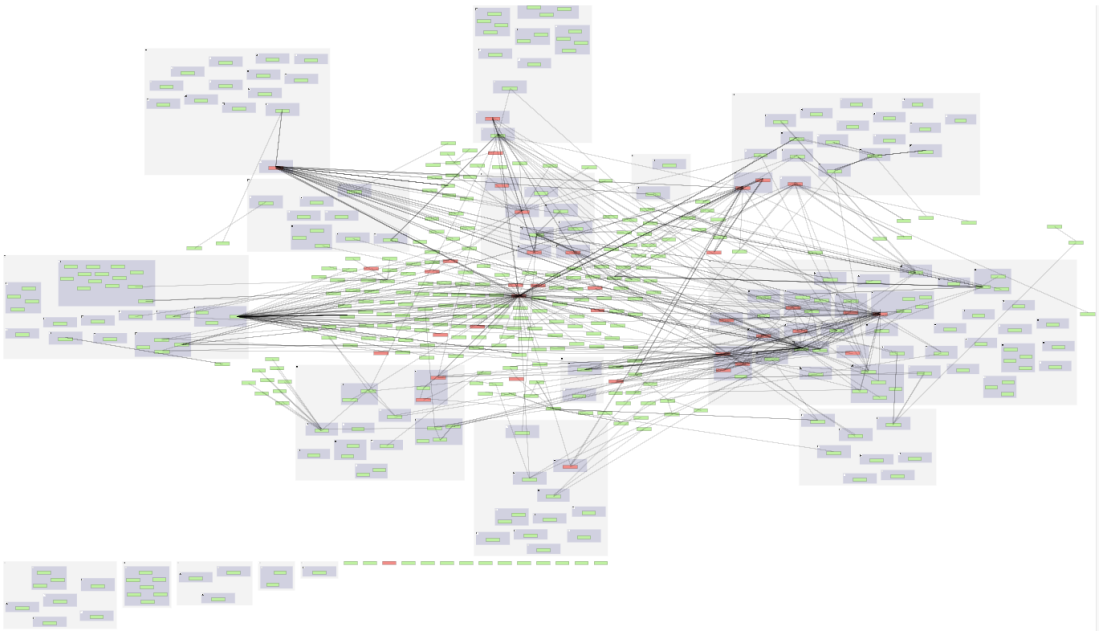


Figure 8.19: Visualization of the security similarity result.

Applicability The majority of the provided templates from chapter 6 are applicable within this case study. In specific the missing property metric was used to verify the existence of a transfer, encryption, data volume and frequency property. An additional metric was added to test for empty string values or the specific value `na`. The missing relation metric was used to retrieve the isolated services without any usage. A customization of this metric was used to determine the internal services without application. To assess the retired or active-deprecated applications that are still in use, two new analysis definitions were created that configure the respective metrics.

Considering the model requirements for microservice assessment, it turns out that the values for the properties data volume and frequency are unsuitable. The templates require a numerical value, whereas within the model only discrete values are provided. Following the templates for M3, M4 indicating their distribution and M7, determining the average size of messages, cannot be applied. Additionally, the discrete values required to identify synchronous and asynchronous interfaces are slightly different. Thus, the respective

templates have to be adapted regarding the expected property value.

Flexibility and adaptability To enable the application of the predefined templates within the model of this case study, several adaptions were required and could be performed with less effort. To cope with the different ways of annotating the synchronous, respective asynchronous, communication, those templates had to be redefined. The expected property value had to be changed from `synchronous` to `online synch`. The structure of the calculation was remained. To verify the absence of property specifications, the provided template could be used but was not sufficient. We had to add further analysis definitions that test e.g. for empty string values. Also the missing relation metric was adapted with an additional property condition in order to restrict the services without applications to the internal ones.

Additionally, we provide a new analysis definition to assess the database usage of services. Using an element metrics, all databases are identified which are accessed by more than one service. The current blueprint demands a one-to-one dependency between services and databases to avoid static data dependencies. 6 of the 20 identified databases violate the blueprint.

Relevance The results from the case study were valuable feedback for the responsible architect. Up to this point, the mentioned Excel file was the foundation for further planning activities and during decision-making. Shortly after our case study, the file was updated and finally replaced with a new service catalog. The microservice assessment can only be interpreted with caution, due to the bad model quality. Nevertheless the results indicate that the organization requires several restructurings to reach their goals of a more homogeneous and loosely coupled landscape. Especially between the different clusters, the goal of minimal dependencies is not fulfilled. Using the defined analyses, the progress regarding homogenization and loosely coupling can be further monitored. Additionally, the model quality analyses are further used to monitor the progress of updating the data. Despite the metrics, we implemented respective views to outline the outdated or inconsistent elements to support this process.

8.5 Discussion

The A2F provides comprehensive analysis functionality for enterprise architecture models. The representation of the model with the GMM enables its generic applicability and the large coverage of the analysis language Arla provides access to several different analysis types. For executing the analyses we utilize a combination of SPARQL and data-flow based analysis. Both approaches are successfully applied in large application scenarios and thus provide a scalable technical foundation for our analysis execution. The graph-based query language SPARQL provides features to answer structural questions and extract model parts. DFA is perfectly qualified to answer behavioral questions, execute recursive analysis definitions and deal with cyclic dependencies. It enables a forward and also backward traversing of the model.

Common practice is a visual inspection of the available EA models or the manual creation of further views. If the relevant views are already available, it is obviously easier to utilize them for decision-making. Within the case studies we observed that the quality of the existing models is often not sufficient and that this fact is not known to the users. Additionally, the consistency of manually created views cannot be ensured. Analysis support enables the assessment of the quality as well as provides reliable views for further processing. Additionally, specific demands like a database usage assessment can be directly evaluated with respective metrics or views.

Major weakness of the proposed framework is the missing support for probabilistic analysis approaches. Probabilistic techniques are widely used within EAM and meet several of the information demands as well as they enable to deal with the uncertainty of the provided information. Additionally, the usability of the provided textual language should be improved or replaces with user interfaces. Especially for the business-related stakeholders, the textual language is not suitable.

In an additional case study we compared the model-based analysis approach to automated tests to determine the effects of interface changes. The case study is described within the bachelor thesis [Wet18] which was supervised by the author of this thesis. To determine potential effects of interface changes within a customer relationship management system, the architecture of the system was represented within AutoAnalyze. Using the provided analysis functionality the effects of a semantic or syntactic change at an interface are determined. The suitability of the model-based approach is compared to the suitability of automated tests. The tests were implemented to ensure the compatibility of changes.

The model-based approach enables the approximation of the effects of syntactic as well as semantic changes. However, the test-based approach provides the actual effects of a change, but only in the syntactic case. The model-based approach has the short coming of maintaining the model and ensuring a sufficient data quality. Whereas within the test-based approach, the provided results depend on the quality of the executed tests.

In the following the proposed analysis framework A2F is finally discussed according to the specified design goals in section 5.1.

Generic applicability The GMM enables a tool and meta model independent representation of the EA data for analysis execution. The adaption process for a specific EA initiative is kept as simple as possible. The concept of element and relation classes, provides a mechanism to define generic analyses and execute them with small effort on a specific model. Only during import definition the mapping of stereotypes to the classes

has to be made once. Additionally, if more detailed semantics about the elements and relations are required, it is possible to use stereotype variable within the analysis templates. These variables have to be mapped to concrete EA model stereotypes before execution.

We were able the represent several different EA models, relying on different EA meta models, with the GMM. These are the three models from the case studies. Additionally, the EA model of a medium-sized manufacturing company, the EA model of automotive manufacturer and those of a public administration were captured with the GMM.

The application of the provided templates in the use cases within the three case studies proves the suitability of this representation format for analysis execution. Within the medium-sized manufacturing company the impact, scope and path analysis are in productive use.

Universal interface With Arla a DSL is provided that enables a uniform interface to common analysis activities during the EAM process. The overall coverage of the A2F regarding existing EA analyses is quite high. More than 60% of the identified analysis types within EAM can be captured with the A2F. An additional 20% of the types can partially be captured with the A2F. Regarding the identified technical and functional dimensions, the A2F covers nearly all. Only for the functional dimension `System` no approach could be identified that is captured or partially captured with the A2F. This is due to the major weakness of a missing support for probabilistic approaches like Bayesian networks and Extended Influence Diagrams. All other technical dimensions are covered or at least partially covered by Arla.

Custom analyses The language supports seven different analysis classes as well as the possibility to integrate custom analysis definition using the DFA approach or individual SPARQL queries. Each analysis class can be instantiated according to the configuration possibilities. This enables the custom definition of views with the scope analysis as well as of custom metrics. Also dependencies can be determined according to the current information demand using the path analysis. This is important for the creation of the often-used business support maps. Finally, the rules for the impact analysis can be customized to be able to cover different impact types like changes or failures. The supported analysis composition enables the definition of more complex analyses and extends the expressiveness of the language.

Declarative analysis definition The language Arla abstracts from all the technical details during analysis definition. The concrete execution procedures are generated form the Arla definition at runtime. For example views can be defined through defining the constraints an element has to fulfill to be part of it. The verification procedure for these constraints is generated from the analysis configuration. Also for the path analysis only the source and target element types have to be provided as well as the considered relation types. The remaining evaluation and conversion are performed by the A2F.

Re-use of analysis definitions The requested re-use of EA analyses is supported through template definition within Arla. We evaluated the template mechanism through an application of the two use cases within the case studies. If the data quality was sufficient, the templates could be easily adapted within the respective case study. Especially templates

like the missing property metric or the missing relation metric are highly reusable.

Re-use is also supported with the provided element and relation classes of the GMM. Analyses can be defined while only referencing to those classes instead of concrete stereotypes. In this case, even no mapping between the stereotypes of the templates and those of the concrete EA model is required. The respective definition can be directly executed.

Additionally, we demonstrated the flexibility of the utilized technologies with a re-implementation of the performance analysis proposed in [JI09]. The beforehand statically defined execution procedure could be converted into a dynamic one using DFA propagation rules. The determination of the performance measures relies only on the provided relation classes and the presence of the required property values. The identifiers for the property values are provided within the analysis configuration. Therewith, the performance analysis is now executable independently from the utilized EA meta model.

Robustness regarding large models SPARQL as well as the utilized DFA approach are well-proved technologies that are capable to deal with large models. Since we restricted the concepts for data representations to RDFS there exists several implementations for scalable reasoners as well as query engines. The DFA approach is already applied in medium-sized as well as very large models.

The largest case study, case study 3, consists of about 1.200 model elements and 650 usage dependencies. Additionally, there are another 1.200 assignment and realization dependencies between the model elements. The execution times for executing the impact analysis, a scope analysis or a metric calculation are below 100ms on an average laptop.

The path analysis, the scope analysis and the impact analysis are currently in use within a very large enterprise architecture. The EA model contains about 2.700 elements and about 6.700 relations. Within the modeling tool, more than 250 different views in terms of diagrams are created for its representation. We measured the analysis execution times in two cases for the most complex analysis, the path analysis, on an average laptop. The path analysis result is used for business support map creation. To retrieve the necessary information one to three path configurations are executed. The results are used to locate the elements within the support map.

In the first case, two path configurations with maximum path size 1 are executed. In total 792 paths were determined in 500 ms on average. In the second case, three path configurations are executed. The first two ones retrieve paths of the length 3, the third one paths of the length 6. In this case in total 76.270 paths are identified in 16 seconds on average.

Robust towards incomplete models Especially the third case study, the backend service landscape, had the problem of insufficient data quality. With the beforehand assessment of model quality in generic terms as well as analysis specific terms, valuable feedback could be provided to the architect. Templates, for which the data was not present, were not included during the assessment of the microservice principles. Additionally, parts of the architecture with obviously outdated data were excluded. The execution procedures for SPARQL and DFA are implemented in a robust way so that missing entries do not lead to errors during evaluation.

Round-trip-engineering The A2F was integrated as a plugin in the modeling tool Innovator. Thereby, not only the model data is exported to execute the analyses, the determined results are also provided back into the modeling tool. In specific the impact analysis is used to apply a coloring to the available diagrams. The result of a scope analysis is used for the generation of new diagrams and the path analysis result is used to generate business support maps. These maps are also represented as diagrams within the tool.

9
Conclusion

9.1 Summary

Within this thesis, we presented a comprehensive approach for EA analysis. Analyzing EA models is an essential part within enterprise architecture management. They increase the understanding of the current architecture and thus, enable the identification of weak points and optimization potentials. The generic applicable approach supports the understanding, the evaluation as well as the comparison of enterprise architecture models.

Analysis support is essential to handle the large and complex nature of EA models as well as to fulfill the information demands of the stakeholders. The proposed architecture analysis framework fosters the assessment of the current architecture as well as the evaluation and integration of planning scenarios. Figure 9.1 depicts the structure of the proposed framework A2F. The three components, *Generic Meta Model*, *Analysis Execution* and *Analysis Definition Language*, represent the main contributions to meet the objectives of this thesis.

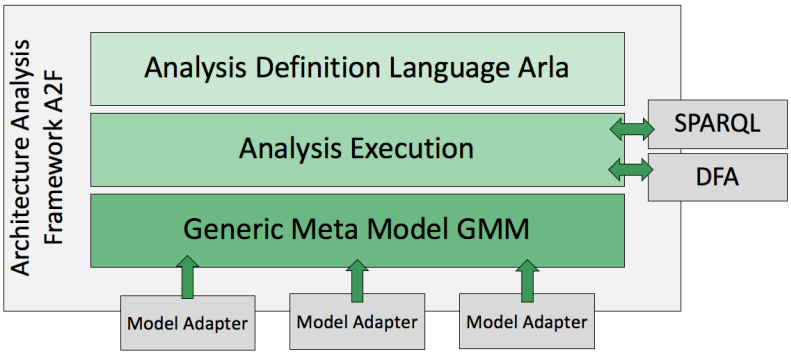


Figure 9.1: Overview of the architecture analysis framework (A2F).

Objective 1: Generic representation

Objective one, a generic representation for EA models that facilitates analysis execution, is achieved with the Generic Meta Model GMM. This representation enables the re-use of the existing EA models within an organization. One or more existing resources can be integrated into one GMM model to create a holistic overview of the architecture. A respective adapter has to be defined once, the actual conversion can be performed without further user interaction. Thus, it is possible to make use of these existing models and increase their utility (challenge one).

For analysis execution, a processable representation of the architecture as a model is required. Within the EA domain there exists a plethora of different meta models that propose concepts for the description (challenge two). A standard representation does not exist and thus, it is essential that the analysis framework can deal with the variety. The proposed Generic Meta Model (GMM) interprets an EA model as stereotyped graph. It is used to describe the model itself as well as the corresponding meta model. An EA model within the GMM consist of `ModelNodes`, `ModelEdges` and `ModelProperties` that represent the elements and relations as well as their properties. The type of these elements is captured with respective `MetaModelNodes`, `MetaModelEdges` and `MetaModelProperties`. `MetaModelEdges` are further distinguished by `MetaModelEdgeConnection` that define the concrete source and target `MetaModelNodes`. Finally, `MetaModelNodes` can be assigned

to a ModelLayer to depict the different hierarchies within the EA model. Therewith, it is possible to represent most of the EA models, more concrete those EA models that are specified in an object-oriented manner.

Additionally, the GMM includes an optional categorization approach for the different element and relation types. Seven different relations classes and four different element classes are provided to extend the generic GMM element with further semantics. The classes depict commonly used concepts within EA models. The seven identified relation classes are: Provide, located at, consumed by, structural dependent of, behavioral dependent of, instance of and generalization. Relation types can be assigned to these classes to enable the correct interpretation within predefined analyses. The proposed element classes are: Human, Process, Application and Infrastructure Component. The classes provide an additional information despite the concrete type of an element. Therewith, analysis definitions can be provided without relying on a concrete meta model.

Objective 2: Generic architecture analysis framework

Objective two, the provisioning of a universally applicable analysis framework that is independent from the utilized meta model, is met through the combination of the different components of the A2F. The Architecture Analysis Framework A2F encompasses three components. One for model storage, one for analysis definition and one for analysis execution. Utilizing the GMM for model storage and Arla for analysis definition as well as the previously presented execution approach fulfills objective two. Arla provides a uniform interface towards the different analysis activities.

Analysis activities within EAM are implemented with various different analysis types. Within [Rau13] we identified in total 40 analysis types within current literature. A comprehensive approach covering different types is missing [NSV14, BMS09, JNL07]. With the Analysis Definition Language Arla we presented a declarative DSL that provides a uniform interface to analysis activities. The language supports eight different analysis classes that are customizable according to the current information demands of the stakeholder. These are the Scope analysis, the Impact analysis, the Path analysis, Metric calculation, the Performance analysis, the Gap analysis, the Adapted analysis and the Composed analysis.

With these eight mentioned classes it is possible to capture 60% of the identified analysis types which addresses the challenge of the various different analysis approaches and the necessity for an integrated one (challenge four). Another 20% of the types can be captured with the definition of a Custom Analysis. This class allows the direct specification of SPARQL queries or the reference to individual data-flow analysis configurations.

To be able to fulfill the varying and changing analysis needs of the stakeholders (challenge three), the analysis classes can be customized. Thereby the user defines 'what' he is interested in, the technical details for the implementation are generated. Re-use of analysis definitions within Arla is supported with two mechanisms. One is the definition of templates. A template provides a generic configuration for a specific analysis, independent from a concrete meta model. Variables are used to reference meta model elements instead of concrete stereotypes. Within an Adapted analysis, a template can be tailored to a concrete EA model through providing a mapping of the variable to stereotypes. A re-definition of the configuration details is not required.

The other one is the support of Arla for the proposed element and relation classes of

the GMM. They can be used within templates and specific analysis definitions. A template, only defined with classes, does not require any mapping for its execution. This enables predefined out-of-the-box analyses while keeping independent form the utilized meta model.

For the execution of the template and analysis definitions two different techniques are employed. Structural requests are transformed into SPARQL queries, a query language for RDF data. Behavioral requests, like the impact of an event, are evaluated with the data-flow analysis. The combination of both techniques facilitates the implementation of each analysis classes in Arla. The concrete execution routines are generated from the Arla definitions, while considering the specific configurations made by the user. Utilizing these techniques provides a scalable execution environment. SPARQL as well as DFA are proven techniques, with reasonable response time also in large models. It is possible to consider cyclic dependencies as well as to deal with missing information in the models.

Objective 3: Employ framework for architecture assessment

To fulfill objective three, the A2F is employed for the assessment of weak points as well as the evaluation of planning scenarios. The A2F is employed to provide analysis support within these use cases. The use cases depict different analysis activities of stakeholders during EAM. Within each use case different views and evaluations are required (challenge five).

The first one is the assessment of weak points within enterprise architectures. Based on a preliminary assessment of the model quality, the architecture is evaluated regarding possible weak points. This is exemplified with the assessment of microservice principles. Based on challenges within practice and typical characteristics of microservice architectures, nine metrics are defined to validate the adherence to the principles. The metrics for microservice assessment as well as the metrics to evaluate the model quality are realized with templates.

The second use case provides analysis support during EA planning, in specific for the evaluation of planning scenarios. A combination of different analyses is proposed to evaluate a planning scenario regarding its conformance to the current principles and goals. The relevant domain of a planning scenario is defined with a gap analysis, providing the differences to the current architecture. The scope analysis retrieves the relevant part of the current architecture that is affected by the planning scenario. An impact analysis approximates potential ripple effects of the changes and thus ensures the consistency of the scenario. Finally, the target architecture is assessed with quantitative measures as well as the generation of views for qualitative assessment. The analysis support is realized using Arla templates which enable the generic application of the evaluation method.

Objective 4: Evaluate the methods and the framework

To fulfill objective four, three case studies as well as a scenario-based evaluation are carried out. They account for the applicability of the framework as well as its coverage regarding existing analysis approaches. We identified twelve scenarios to prove the coverage of Arla regarding the different functional and technical dimensions of EA analyses. Each scenario depicts the implementation of one existing analysis approach from literature. All covered and partially covered dimensions are addressed in at least one scenario. Only the functional dimension *System* cannot be captured with Arla as well as the technical

dimensions depicting probabilistic approaches (Bayesian networks and Extended Influence Diagrams).

For the application within the case studies we provided two implementations of the A2F. One is a partial integration in the modeling tool `Innovator`, the other one is an almost complete realization utilizing the AutoAnalyze modeling product. Within the case studies, we proofed the applicability of the provided templates within the use cases as well as demonstrated the generic applicability of the framework. Each case study utilizes a different meta model and comprises different types of EA elements. The performed analyses provide valuable feedback for the stakeholders which enhances their understanding of the architecture and also points out weaknesses. The analysis support for the evaluation of planning scenarios detected inconsistencies within the previously manually created domain models.

9.2 Future Work

There are two main focuses for future work. One is the improvement of the EA model through considering more data sources and provide better integration possibilities. The other one is the extension of Arla and the analysis execution with further functionality. This includes an extension of the analysis classes regarding their expressiveness and also the creation of a comprehensive collection of different analysis templates.

The quality of the analysis results is highly dependent of the quality of the EA model. Integrating different sources into one model provides a more solid foundation for analysis activities. It also enables the verification of manually created data, e.g. from EA tools, with data about actual dependencies from monitoring tools and data logs. Thus, outdated or missing data can be identified and the model quality can be increased. Future work includes the creation of further model adapters to support those comparisons. Additionally, the mapping possibilities for EA data from different sources have to be extended. Reasoning over the RDF data set or the employment of rules can be applied for this task.

Within future work the data representation with RDFS could be further extended to the use of OWL. This enables more expressiveness, especially regarding relations between the model elements. OWL allows the specification of multiplicities as well as the annotation of inverse relations. In this case, it is possible to validate the quality of the model automatically with reasoning techniques.

The second focus of future work addresses the expressiveness and coverage of Arla and the A2F. Since probabilistic analyses are currently not supported, we will elaborate ways to integrate them. For example, Bayesian reasoning (see e.g. [SBLH06]) can be integrated to derive probabilities of certain events according to their probability distribution. Additional probabilistic techniques allow the consideration of uncertainties regarding the existence of relations, properties and elements (see e.g. [NBE14]). The integration of this techniques enables the coverage of all identified functional and technical dimensions of EA analyses.

To enable more fine-grained analysis definitions by the user, the configurations for path, impact and scope analysis will be extended. In the future, it should be possible to utilize relation classes and stereotype references in combination. Thereby, the stereotype references will overwrite the more generic relation classes. This decreases also the effort for customization of existing analyses. The user can choose the templates that fits best for his scenario and then only have to provide extensions for certain stereotypes that are interpreted differently.

Additionally, to increase the independence of the analysis definition in Arla from a specific meta model, future work will include further consideration of indirect dependencies within analysis configurations. The results of the path analysis or results of a reasoning provide the related elements and not the definition of a concrete relation type or relation class. For example, the expression “realizing component” will not rely on a specific type, but on a dynamic determination of this element. Thus, it is irrelevant if there are one or more intermediate elements between the current element and its realizing component.

Future work will also include the extension of the analysis composition possibilities. Within the case studies, we observed that the characteristics of paths, like their lengths, their total number or the number of paths to one source node, are characteristics that are needed within other analysis classes. Furthermore, it must be possible to assign a set of elements as result for an element. A use case would be the determination of those applications

that realize a business process. Also, the adaption of Arla templates will be extended within future work. This includes support for overwriting a specified property value for comparisons. Additionally, new analysis classes, like an analysis of strongly connected components, or the direct support of date values are planned.

Finally, the provided implementation of the A2F within the modeling tool Innovator is extended with the remaining analysis classes in future work. Thereby, the usability of the proposed language is considered and improved. A user interface is developed to enable the specification of analysis and templates definitions in a more user-friendly way.

Part V

Annex

Bibliography

[AAE16] N. Alshuqayran, N. Ali, and R. Evans. A systematic mapping study in microservice architecture. In *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016.

[ABF+09] 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 *3rd International Workshop on Enterprise Modelling and Information Systems Architectures*, 2009.

[ACS15] P. Andersen, A. Carugati, and M. G. Sørensen. Exploring Enterprise Architecture Evaluation Practices: The Case of a Large University. In *Proceedings of the 48th Hawaii International Conference on System Sciences (HICSS 15)*, 2015.

[AG10a] S. Aier and B. Gleichauf. Application of enterprise models for engineering enterprise transformation. *Enterprise Modelling and Information System Architectures*, 5(1):58–75, 2010.

[AG10b] S. Aier and B. Gleichauf. Applying Design Research Artifacts for Building Design Research Artifacts: A Process Model for Enterprise Architecture Planning. In *Global Perspectives on Design Science Research. DESRIST 2010*, volume 6105 of *Lecture Notes in Computer Science*. Springer, 2010.

[AGSW09] S. Aier, B. Gleichauf, J. Saat, and R. Winter. Complexity Levels of Representing Dynamics in EA Planning. In *Advances in Enterprise Engineering III. CIAO! 2009, EOMAS 2009*, volume 34 of *Lecture Notes in Business Information Processing*. Springer, 2009.

[AKRS08] S. Aier, S. Kurpjuweit, C. Riege, and J. Saat. Stakeholderorientierte Dokumentation und Analyse der Unternehmensarchitektur. In H.-G. Hegering, A. Lehmann, H.J. Ohlbach, and C. Scheideler, editors, *Proceedings of INFORMATIK 2008 : Beherrschbare Systeme - dank Informatik*. Gesellschaft für Informatik, 2008.

[AMMN16] M. Amundsen, M. McLarty, M. Mitra, and I. Nadareishvili. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, 2016.

[ANT14] ANTLR. Another tool for language recognition ANTLR. In `http://www.antlr.org`, 2014. Accessed 26/03/2019.

[APC+15] M. Amaral, J. Polo, D. Carrera, Mohomed I., M. Un-uvar, and M. Steinder. Performance evaluation of microservices architectures using containers. In *IEEE 14th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2015.

[APH10] A. Aryani, I.D. Peake, and M. Hamilton. Domain-based change propagation

analysis: An enterprise system case study. In *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2010.

[Arc18] Architecture Capability Team. NAF Architecture Framework. Technical Report Version 4, NATO, 2018.

[AS11] S. Aier and J. Saat. Understanding processes for model-based enterprise transformation planning. *International Journal of Internet and Enterprise Management*, 7(1), 2011.

[ASML12] F. Ahlemann, E. Stettiner, M. Messerschmidt, and C. Legner. *Strategic Enterprise Architecture Management*. Springer, 2012.

[BBJ+11] S. Buckl, M. Buschle, P. Johnson, F. Matthes, and C. M Schweda. A meta-language for EA information modeling. *Enterprise, Business-Process and Information Systems Modeling*, page 511–525, 2011.

[BCW12] M. Brambilla, J. Cabot, and M. Wimmer. *Model-driven software engineering in practice*. Morgan & Claypool Publishers, 2012.

[Bea19] Phillip Beauvoir. Archi – Open Source ArchiMate Modelling. In `https://www.archimatetool.com`, 2019. Accessed 26/03/2019.

[BEL+07] S. Buckl, A.M. Ernst, J. Lankes, K. Schneider, and C.M. Schweda. A pattern based approach for constructing enterprise architecture management information models. In *Wirtschaftinformatik Proceedings 2007*, 2007.

[BFH+09] S. Buckl, U. Franke, O. Holschke, F. Matthes, C.M. Schweda, T. Sommestad, and J. Ullberg. A pattern-based approach to quantitative enterprise architecture analysis. In *15th Americas Conference on Information Systems (AMCIS)*, 2009.

[BFKW06] T. Bucher, R. Fischer, S. Kurpjuweit, and R. Winter. Analysis and application scenarios of enterprise architecture: An exploratory study. In *10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 06)*. IEEE, 2006.

[BG14] D. Brickley and R.V. Guha. RDF Schema 1.1. W3C Recommendation, 2014.

[BHS+12] M. Buschle, H. Holm, T. Sommestad, M. Ekstedt, and K. Shahzad. A Tool for Automatic Enterprise Architecture Modeling. In S. Nurcan, editor, *IS Olympics: Information Systems in a Diverse World. CAiSE 2011*, volume 107 of *Lecture Notes in Business Information Processing*. Springer, 2012.

[BLO03] L.C. Briand, Y. Labiche, and L. O'Sullivan. Impact analysis and change management of UML models. In *International Conference on Software Maintenance (ICSM 2003)*. IEEE, 2003.

[BMNS09] S. Buckl, F. Matthes, C. Neubert, and C.M. Schweda. A wiki-based approach to enterprise architecture documentation and analysis. In *17th European Conference on Information Systems (ECIS 2009)*, 2009.

[BMS09] S. Buckl, F. Matthes, and C.M. Schweda. Classifying enterprise architecture analysis approaches. In *Enterprise Interoperability. IWEI 2009*, volume 38 of *Lecture Notes in Business Information Processing*. Springer, 2009.

[Boh02] S.A. Bohner. Software change impacts-an evolving perspective. In *International Conference on Software Maintenance (ICSM 2002)*. IEEE, 2002.

[BR88] V. R. Basili and H. D. Rombach. The TAME project: towards improvement-

oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773, 1988.

[BUF+11] M. Buschle, J. Ullberg, U. Franke, R. Lagerström, and T. Sommestad. A tool for enterprise architecture analysis using the PRM formalism. In *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing*. Springer, 2011.

[BvSF+10] W. Bruls, M. van Steenbergen, R.M. Foorthuis, R. Bos, and S. Brinkkemper. Domain architectures as an instrument to refine enterprise architecture. *Communications of the Association for Information Systems*, 27:517–540, 2010.

[BW05] C. Braun and R. Winter. A comprehensive enterprise architecture metamodel and its implementation using a metamodeling platform. In J. Desel and U. Frank, editors, *Enterprise Modelling and Information Systems Architectures - Proceedings of the Workshop in Klagenfurt*. Gesellschaft für Informatik, 2005.

[BWZ17a] J. Bogner, S. Wagner, and A. Zimmermann. Automatically masuring the maintainability of service- and microservice-based systems - a literature review. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. ACM, 2017.

[BWZ17b] J. Bogner, S. Wagner, and A. Zimmermann. Towards a Practical Maintainability Quality Model for Service-and Microservice-based Systems. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*. ACM, 2017.

[Car16] J. Carter. The Essential Project Team Blog - What about OWL? In `https://www.enterprise-architecture.org/blogs/tag/owl/`, 2016. Accessed 26/03/2019.

[CDK11] G. F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Pearson, 2011.

[CHL+13] W. Chen, C. Hess, M. Langermeier, J. von Stülpnagel, and P. Diefenthaler. Semantic Enterprise Architecture Management. In *International Conference on Enterprise Information Systems (ICEIS 13)*, 2013.

[CWL14] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2014.

[DA09] M. R. Davoudi and F. S. Aliee. A New AHP-based Approach towards Enterprise Architecture Quality Attribute Analysis. In *Third International Conference on Research Challenges in Information Science (RCIS 2009)*. IEEE, 2009.

[DB13] P. Diefenthaler and B. Bauer. Gap Analysis in Enterprise Architecture using Semantic Web Technologies. In *15h International Conference on Enterprise Information Systems (ICEIS 2013)*, 2013.

[DB14] P. Diefenthaler and B. Bauer. From Gaps to Transformation Paths in Enterprise Architecture Planning. In *Enterprise Information Systems*, volume 190 of *Lecture Notes in Business Information Processing*. Springer, 2014.

[dBBG+05] F.S. de Boer, M.M. Bonsangue, L.P.J. Groenewegen, A.W. Stam, S. Stevens,

and L. van der Torre. Change impact analysis of enterprise architectures. In *IEEE International Conference on Information Reuse and Integration (IRI 2005)*. IEEE, 2005.

[DBLR+11] M. Della Bordella, R. Liu, A. Ravarini, F.Y. Wu, and A. Nigam. Towards a method for realizing sustained competitive advantage through business entity analysis. In *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *Lecture Notes in Business Information Processing*. Springer, 2011.

[Dep10] Department of Defense, United States. The DoDAF Architecture Framework Version 2.02. In `http://dodcio.defense.gov/dodaf20.aspx`, 2010. Accessed 26/03/2019.

[DGL+17] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. In M. Mazzara and B. Meyer, editors, *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

[Dyn19a] Dynatrace LLC. Dynatrace - Software Intelligenz für die Enterprise Cloud. In `https://www.dynatrace.de`, 2019. Accessed 26/03/2019.

[Dyn19b] Dynatrace LLC. Top Challenges Facing CIOs in a Cloud-Native World. In `https://www.dynatrace.com/cloud-complexity-report/`, 2019. Accessed 26/03/2019.

[Ecl18a] Eclipse Foundation. Eclipse Modeling Framework (EMF). In `https://www.eclipse.org/modeling/emf/`, 2018. Accessed 26/03/2019.

[Ecl18b] Eclipse Foundation. Xtext - Language Engineering for Everyone. In `https://www.eclipse.org/Xtext/`, 2018. Accessed 17/01/2019.

[EFJ+09] M. Ekstedt, U. Franke, P. Johnson, R. Lagerstrom, T. Sommestad, J. Ullberg, and M. Buschle. A tool for enterprise architecture analysis of maintainability. In *13th European Conference on Software Maintenance and Reengineering*. IEEE, 2009.

[EHH+08] G. Engels, A. Hess, B. Humm, O. Juwig, and M. Lohmann. *Quasar enterprise: Anwendungslandschaften serviceorientiert gestalten*. dpunkt.verlag, 2008.

[ELBH18] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann. Evaluation of Microservice Architectures: A Metric and Tool-Based Approach. In J. Mendling and H. Mouratidis, editors, *Information Systems in the Big Data Era. CAiSE 2018*, volume 317 of *Lecture Notes in Business Information Processing*. Springer, 2018.

[Eng17] T. Engel. Evaluation von Microservice-Architekturen: Definition von Metriken und Entwicklung eines Analysetools. Master thesis, University of Augsburg, 2017.

[Eva03] E.J. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, 2003.

[Exe12] Executive Office of the President of the U.S. The common approach to federal enterprise architecture. In `https://obamawhitehouse.archives.gov/sites/default/files/omb/assets/egov_docs/common_approach_to_federal_ea.pdf`, 2012. Accessed 26/03/2019.

[FAB+11] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke. Automation processes for enterprise architecture management. In *15th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 11)*. IEEE, 2011.

[FB08] R. Foorthuis and S. Brinkkemper. Best Practices for Business and Systems Analysis in Projects Conforming to Enterprise Architecture. *Enterprise Modelling and Information Systems Architectures*, 3(1):36–47, 2008.

[FFJ09] U. Franke, Waldo R. Flores, and P. Johnson. Enterprise architecture dependency analysis using fault trees and Bayesian networks. In *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 2009.

[FHBB12] R. Foorthuis, F. Hofman, S. Brinkkemper, and R. Bos. Compliance Assessments of Projects Adhering to Enterprise Architecture. *Journal of Database Management*, 23(2):44–71, 2012.

[FHK09] U. Frank, D. Heise, and H. Kattenstroth. Use of a domain specific modeling language for realizing versatile dashboards. In *Proceedings of the 9th OOPSLA workshop on domain-specific modeling (DSM)*. Helsinki Business School, 2009.

[FJdW97] H.M. Franken, H. Jonkers, and M.K. de Weger. Structural and quantitative perspectives on business process modelling and analysis. In *Proceedings of the 11th European Simulation Multiconference*. Citeseer, 1997.

[FL14] M. Fowler and J. Lewis. Microservices - a definition of this new architectural term. In `https://martinfowler.com/articles/microservices.html`, 2014. Accessed 26/03/2019.

[FML17] P. D. Francesco, I. Malavolta, and P. Lago. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In *IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017.

[Fou18] The Apache Software Foundation. Apache Jena. In `https://jena.apache.org`, 2018. Accessed 26/03/2019.

[Fra11] U. Frank. The MEMO Meta Modelling Language (MML) and Language Architecture. ICB-Research Report 43, University Duisburg-Essen, 2011. 2nd edition.

[Fra14] U. Frank. Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. *Software & Systems Modeling*, 13(3):941–962, 2014.

[GCF+17] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle. Towards Recovering the Software Architecture of Microservice-Based Systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017.

[Gra15] J. Gräupner. Transformation from a product house to a SaaS provider from the viewpoint of Enterprise Architecture Management by the case example MID GmbH. Master thesis, University of Augsburg, 2015.

[Han13] I. Hanschke. *Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise Architecture Management*. Carl Hanser Verlag, 2013.

[HBLE14] H. Holm, M. Buschle, R. Lagerström, and M. Ekstedt. Automatic data collection for enterprise architecture models. *Software & Systems Modeling*, 13(2):825–841, 2014.

[HGR13] S. Harris, N. Gibbins, and A. Riddoch. 3store. In `http://sourceforge.net/projects/threestore`, 2013. Accessed 26/03/2019.

[HHKR16] I. Hanschke, S. Hanschke, A. Kozlovskaya, and M. Rempte. EAM-Tool-Survey 2015/2016, 2016.

[HKP+12] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation, 2012.

[HMPR04] A.R. Hevner, S.T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[HNF+09] O. Holschke, P. Närman, W.R. Flores, E. Eriksson, and M. Schönherr. Using Enterprise Architecture Models and Bayesian Belief Networks for Failure Impact Analysis. In Feuerlicht G. and Lamersdorf W., editors, *Service-Oriented Computing – ICSOC 2008 Workshops. ICSOC 2008*, volume 5472 of *Lecture Notes in Computer Science*. Springer, 2009.

[HRSM13] M. Hauder, S. Roth, C. Schulz, and F. Matthes. Current tool support for metrics in enterprise architecture management. In *MetriKon 2013*, 2013.

[HS17] 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)*. IEEE, 2017.

[IFI99] IFIP–IFAC Task Force on Architectures for Enterprise Integration. GERAM: Generalised Enterprise Reference Architecture and Methodology. In *Handbook on Enterprise Architecture. International Handbooks on Information Systems*. Springer, 1999.

[Ins19] Instana GmbH. Instana. In `https://www.instana.com/`, 2019. Accessed 26/03/2019.

[IOB06] F. Innerhofer-Oberperfler and R. Breu. Using an enterprise architecture for it risk management. In *Proceedings of the ISSA 2006 from Insight to Foresight Conference*, 2006.

[ISO11a] ISO/IEC 9075:2011. Information technology - Database languages - SQL. International standard, 2011.

[ISO11b] ISO/IEC/IEEE 42010:2011. Systems and software engineering - architecture description. International Standard V1, 2011.

[ite18] iteratec GmbH. iteraplan Query Console. In `https://doc.iteraplan.de/display/iteraplan64/Query+Console`, 2018. Accessed 26/03/2019.

[ite19] iteratec GmbH. iteraplan. In `https://www.iteraplan.de`, 2019. Accessed 26/03/2019.

[JBR+99] H. Jonkers, P. Boekhoudt, M. Rougoor, E. Wierstra, et al. Completion time and critical path analysis for the optimisation of business process models. In *Summer Computer Simulation Conference*. Citeseer, 1999.

[JI09] H. Jonkers and M.-E. Iacob. Performance and cost analysis of service-oriented enterprise architectures. In *Global Implications of Modern En-*

terprise Information Systems: Technologies and Applications, pages 49–73. Information Science Reference, 2009.

[JJ05] E. Johansson and P. Johnson. Assessment of enterprise information security-an architecture theory diagram definition. In *Proceedings of 3rd Annual Conference on Systems Engineering Research (CSER 05)*, 2005.

[JJSU07] P. Johnson, E. Johansson, T. Sommestad, and J. Ullberg. A tool for enterprise architecture analysis. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 07)*. IEEE, 2007.

[JLNS07a] P. Johnson, R. Lagerström, P. Närman, and M. Simonsson. Enterprise architecture analysis with extended influence diagrams. *Information Systems Frontiers*, 9(2-3):163–180, 2007.

[JLNS07b] P. Johnson, R. Lagerström, P. Närman, and M. Simonsson. Extended influence diagrams for system quality analysis. *Journal of Software*, 2(3):30–42, 2007.

[JNL07] P. Johnson, L. Nordström, and R. Lagerström. Formalizing analysis of enterprise architecture. In G. Doumeingts, J. Müller, G. Morel, and B. Vallespir, editors, *Enterprise Interoperability*. Springer, 2007.

[JUB+13] P. Johnson, J. Ullberg, M. Buschle, U. Franke, and K. Shahzad. P2AMF: Predictive, Probabilistic Architecture Modeling Framework. In *Enterprise Interoperability. IWEI 2013*, volume 144 of *Lecture Notes in Business Information Processing*. Springer, 2013.

[KA09] S. Kurpjuweit and S. Aier. Ein allgemeiner Ansatz zur Ableitung von Abhängigkeitsanalysen auf Unternehmensarchitekturmodellen. In *Wirtschaftinformatik Proceedings 2009*, 2009.

[KAV05] S. H. Kaisler, F. Armour, and M. Valivullah. Enterprise Architecting: Critical Problems. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS 05)*. IEEE, 2005.

[Kil73] G.A. Kildall. A unified approach to global program optimization. In *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1973.

[Kil16] T. Killalea. The Hidden Dividends of Microservices. *Communications of the ACM*, 59(8):42–45, 2016.

[KMKB14] J. Kienberger, P. Minnerup, S. Kuntz, and B. Bauer. Analysis and Validation of AUTOSAR Models. In *2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 2014.

[KMP11] P. Kazienko, R. Michalski, and S. Palus. Social network analysis as a tool for improving enterprise architecture. In *Agent and Multi-Agent Systems: Technologies and Applications. KES-AMSTA 2011*, volume 6682 of *Lecture Notes in Computer Science*. Springer, 2011.

[Krc05] H. Krcmar. *Informationsmanagement*. Springer, 2005.

[KRRR08] A. Kumar, P. Raghavan, J. Ramanathan, and R. Ramnath. Enterprise interaction ontology for change impact analysis of complex systems. In *IEEE Asia-Pacific Services Computing Conference (APSCC 08)*. IEEE, 2008.

[KW07] S. Kurpjuweit and R. Winter. Viewpoint-based Meta Model Engineering. In *2nd International Workshop on Enterprise Modelling and Information*

Systems Architectures (EMISA 2007). Gesellschaft für Informatik, 2007.

[Lan12] M. Lankhorst. *Enterprise Architecture at Work*. Springer, 2012.

[LB09] Y.T. Leung and J. Bockstedt. Structural analysis of a business enterprise. *Service Science*, 1(3):169–188, 2009.

[LB17] M. Langermeier and B. Bauer. Generic EA Analysis Framework for the definition and automatic execution of analyses. In *International Conference on Enterprise Information Systems (ICEIS 17)*, 2017.

[LB18a] M. Langermeier and B. Bauer. A Model-Based Method for the Evaluation of Project Proposal Compliance within EA Planning. In *22nd IEEE International Enterprise Distributed Object Computing Workshop (EDOCW 18)*. IEEE, 2018.

[LB18b] M. Langermeier and B. Bauer. Evaluating Project Compliance During EA Planning: A Model-based Semi Automatic Method for Enterprise Architecture Planning. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018.

[LDL+13] F. Lautenbacher, P. Diefenthaler, M. Langermeier, M. Mykhashchuk, and B. Bauer. Planning Support for Enterprise Changes. In *The Practice of Enterprise Modeling. PoEM 2013*, volume 165 of *Lecture Notes in Business Information Processing*. Springer, 2013.

[Lea19] LeanIX GmbH. LeanIX - Next-Generation Enterprise Architecture Management. In `https://www.leanix.net/`, 2019. Accessed 26/03/2019.

[LJ08] R. Lagerstrom and P. Johnson. Using architectural models to predict the maintainability of enterprise systems. In *12th European Conference on Software Maintenance and Reengineering*. IEEE, 2008.

[LJE10] R. Lagerström, P. Johnson, and M. Ekstedt. Architecture analysis of enterprise systems modifiability: a metamodel for software change cost estimation. *Software Quality Control*, 18(4):437–468, 2010.

[LJJ+06] Å. Lindström, P. Johnson, E. Johansson, M. Ekstedt, and M. Simonsson. A survey on CIO concerns - do enterprise architecture frameworks support them? *Information Systems Frontiers*, 8(2):81–90, 2006.

[LJW+16] B. Lantow, D. Jugel, M. Wißotzki, B. Lehmann, O. Zimmermann, and K. Sandkuhl. Towards a classification framework for approaches to enterprise architecture analysis. In *The Practice of Enterprise Modeling. PoEM 2016*, volume 267 of *Lecture Notes in Business Information Processing*. Springer, 2016.

[LLP+09] J. Luo, Y. Li, J. Pershing, L. Xie, and Y. Chen. A methodology for analyzing availability weak points in soa deployment frameworks. *IEEE Transactions on Network and Service Management*, 6(1):31–44, 2009.

[LSB14a] M. Langermeier, C. Saad, and B. Bauer. Adaptive approach for impact analysis in enterprise architectures. In Shishkov B., editor, *Business Modeling and Software Design. BMSD 2014*, volume 220 of *Lecture Notes in Business Information Processing*. Springer, 2014.

[LSB14b] M. Langermeier, C. Saad, and B. Bauer. Context-Sensitive Impact Analysis for Enterprise Architecture Management. In *Proceedings of the Fourth International Symposium on Business Modeling and Software Design*, 2014.

[LSB14c] M. Langermeier, C. Saad, and B. Bauer. A unified framework for enterprise architecture analysis. In *18th IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW 14)*. IEEE, 2014.

[MBLS08] F. Matthes, S. Buckl, J. Leitel, and C.M. Schweda. Enterprise Architecture Management Tool Survey 2008. Technical report, Technical University Munich, 2008.

[MID19] MID GmbH. Innovator for Enterprise Architects. In `https://www.mid.de/leistungen/tools/innovator/enterprise-architects`, 2019. Accessed 26/03/2019.

[Min12] Ministry of Defence, United Kingdom. MOD Architecture Framework. In `https://www.gov.uk/guidance/mod-architecture-framework`, 2012. Accessed 26/03/2019.

[MMSS11] F. Matthes, I. Monahov, A. Schneider, and C. Schulz. EAM KPI Catalog. Technical Report v 1.0, Technical University Munich, 2011.

[NBE14] P. Närman, M. Buschle, and M. Ekstedt. An enterprise architecture framework for multi-attribute information systems analysis. *Software & Systems Modeling*, 13(3):1085–1116, 2014.

[New15] S. Newman. *Building Microservices*. O'Reilly and Associates, 2015.

[NFK+12] P. Närman, U. Franke, J. König, M. Buschle, and M. Ekstedt. Enterprise architecture availability analysis using fault trees and stakeholder interviews. *Enterprise Information Systems*, 8(1):1–25, 2012.

[NFT+17] E. Nowakowski, M. Farwick, T. Trojer, M. Häusler, J. Kessler, and R. Breu. Enterprise Architecture Planning: Analyses of Requirements from Practice and Research. In *Proceedings of the 50th Annual Hawaii International Conference on System Sciences (HICSS 17)*, 2017.

[Nie06] K.D. Niemann. *From enterprise architecture to IT governance*. Springer, 2006.

[NJN07] P. Narman, P. Johnson, and L. Nordstrom. Enterprise architecture: A framework supporting system quality analysis. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. IEEE, 2007.

[NSJ+08] P. Narman, M. Schonherr, P. Johnson, M. Ekstedt, and M. Chenine. Using enterprise architecture models for system quality analysis. In *12th International IEEE Enterprise Distributed Object Computing Conference (EDOC 08)*. IEEE, 2008.

[NSV14] D. Naranjo, M. Sánchez, and J. Villalobos. Towards a unified and modular approach for visual analysis of enterprise models. In *18th IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW 14)*. IEEE, 2014.

[NSV15] D. Naranjo, M. Sánchez, and J. Villalobos. Primrose: A graph-based approach for enterprise architecture analysis. In Cordeiro J., Hammoudi S., Maciaszek L., Camp O., and Filipe J., editors, *Enterprise Information Systems. ICEIS 2014.*, volume 227 of *Lecture Notes in Business Information Processing*. Springer, 2015.

[Obj13] Object Management Group. Unified Profile for DoDAF and MODAF. OMG Specification V 2.1, 2013.

[Obj16] Object Management Group. Meta Object Facility. OMG Specification, 2016.

[Obj17] Object Management Group. Unified Modeling Language. OMG Specification V 2.5.1, 2017.

[OLB15] M. Osenberg, M. Langermeier, and B. Bauer. Using Semantic Web Technologies for Enterprise Architecture Analysis. In F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, editors, *The Semantic Web. Latest Advances and New Domains. ESWC 2015*, volume 9088 of *Lecture Notes in Computer Science*. Springer, 2015.

[Ope19a] OpenTracing Specification Council. OpenTracing - Vendor-neutral APIs and instrumentation for distributed tracing. In `https://opentracing.io`, 2019. Accessed 26/03/2019.

[Ope19b] OpenZipkin volunteer organization. Zipkin. In `http://zipkin.io/`, 2019. Accessed 26/03/2019.

[Ora19] Oracle Corporation. Oracle Spatial and Graph RDF Knowledge Graph. In `https://www.oracle.com/technetwork/database/options/spatialandgraph/overview/rdfsemantic-graph-1902016.html`, 2019. Accessed 26/03/2019.

[Ose17] M. Osenberg. Formale Spezifikation von Analyseanforderungen im Enterprise Architecture Management. Master thesis, University of Augsburg, 2017.

[PH05] M. Pulkkinen and A. Hirvonen. EA Planning, Development and Management Process for Agile Enterprise Development. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS 05)*, 2005.

[Piv07] Pivotal Software Inc. RabbitMQ. In `https://www.rabbitmq.com`, 2007. Accessed 26/03/2019.

[PSP12] H. Plessius, R. Slot, and L. Pruijt. On the categorization and measurability of enterprise architecture benefits with the enterprise architecture value framework. In *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation. PRET 2012, TEAR 2012*, volume 131 of *Lecture Notes in Business Information Processing*. Springer, 2012.

[Pul06] M. Pulkkinen. Systemic Management of Architectural Decisions in Enterprise Architecture Planning. Four Dimensions and Three Abstraction Levels. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS 06)*, volume 8, 2006.

[Rau13] J. Rauscher. Analysen in Unternehmensarchitekturen - Ziele, Techniken, Anwendungsbereiche. Bachelor thesis, University of Augsburg, 2013.

[Rau15] J. Rauscher. Anforderungen an und Definition von einer Analysesprache für das Enterprise Architecture Management. Master thesis, University of Augsburg, 2015.

[RDF19] Eclipse RDF4J. RDF4J 2.5.0 released. In `http://rdf4j.org`, 2019. Accessed 26/03/2019.

[RGA07] G. Riempp and S. Gieffers-Ankel. Application portfolio management: a decision-oriented view of enterprise architecture. *Information Systems and e-Business Management*, 5(4):359–378, 2007.

[RHF+13] S. Roth, M. Hauder, M. Farwick, R. Breu, and F. Matthes. Enterprise Architecture Documentation: Current Practices and Future Directions. In *Wirtschaftsinformatik Proceedings 2013*, 2013.

[RLB16] J. Rauscher, M. Langermeier, and B. Bauer. Characteristics of Enterprise Architecture Analyses. In *Proceedings of the Sixth International Symposium on Business Modeling and Software Design*, 2016.

[RLB17] Julia Rauscher, Melanie Langermeier, and Bernhard Bauer. Classification and definition of an enterprise architecture analyses language. In *Business Modeling and Software Design. BMSD 2016*, number 275 in Lecture Notes in Business Information Processing. Springer, 2017.

[Saa10] J. Saat. Zeitbezogene Abhängigkeitsanalysen der Unternehmensarchitektur, booktitle = Multikonferenz Wirtschaftsinformatik 2010. 2010.

[Saa14] C. Saad. *Data-flow based Model Analysis - Approach, Implementation and Applications*. PhD thesis, University of Augsburg, 2014.

[SB11] Christian S. and Bernhard B. The Model Analysis Framework - An IDE for Static Model Analysis. In *Proceedings of the Industry Track of Software Language Engineering (ITSLE) in the context of the 4th International Conference on Software Language Engineering (SLE'11)*, 2011.

[SB13] C. Saad and B. Bauer. Data-flow based Model Analysis and its Applications. In Moreira A., Schätz B., Gray J., Vallecillo A., and Clarke P., editors, *Model-Driven Engineering Languages and Systems. MODELS 2013*, volume 8107 of *Lecture Notes in Computer Science*. Springer, 2013.

[SBLH06] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[SH93] S.H. Spewak and S.C. Hill. *Enterprise Architecture Planning: Developing a Blueprint for Data, Applications and Technology*. QED Information Sciences, 1993.

[Sha17] S. Sharma. *Mastering Microservices with Java 9*. Packt Publishing, 2017.

[SK11] A. Sasa and M. Krisper. Enterprise architecture patterns for business process support analysis. *Journal of Systems and Software*, 84(9):1480–1506, 2011.

[SKR13a] S. Sunkle, V. Kulkarni, and S. Roychoudhury. Analyzable enterprise models using ontology. In *Proceedings of CAiSE Forum*, 2013.

[SKR13b] S. Sunkle, V. Kulkarni, and S. Roychoudhury. Analyzing enterprise models using enterprise architecture-based ontology. In *Model-Driven Engineering Languages and Systems. MODELS 2013*, volume 8107 of *Lecture Notes in Computer Science*. Springer, 2013.

[Sof15] OpenLink Software. About OpenLink Virtuoso. In `https://virtuoso.openlinksw.com`, 2015. Accessed 26/03/2019.

[sof18] soffico GmbH. Orchestra – agile Middleware. In `https://orchestra.soffico.de`, 2018. Accessed 26/03/2019.

[Sof19a] Softplant GmbH. Semantic Enterprise Architecture Man-

agement. In https://www.softplant.de/portfolio-item/semantic-enterprise-architecture-management/, 2019. Accessed 26/03/2019.

[Sof19b] Software AG. Alfabet - Plan your IT to power innovation. In http://alfabet.softwareag.com, 2019. Accessed 26/03/2019.

[SR14] G. Schreiber and Y. Raimond. RDF 1.1 Primer. W3C Working Group Note, 2014.

[ST06] S.H. Spewak and M. Tiemann. Updating the enterprise architecture planning model. *Journal of Enterprise Architecture*, 2(2):11–19, 2006.

[Stu11] H. Stuckenschmidt. *Ontologien: Konzepte, Technologien und Anwendungen*. Springer, 2011.

[Tho15] J. Thönes. Microservices. *IEEE Software*, 32(1):116–116, 2015.

[The13] The Open Group. ArchiMate 2.1 Specification. Open group standard, 2013.

[The17] The Open Group. ArchiMate 3.0.1 Specification. Open group standard, 2017.

[The18] The Open Group. *TOGAF Version 9.2*. Van Haren Publishing, 2018.

[The19] The Open Group. The TOGAF Standard, Version 9.2. In https://www.opengroup.org/togaf-standard-version-92-overview, 2019. Accessed 26/03/2019.

[TNJH07] A. Tang, A. Nicholson, Y. Jin, and J. Han. Using bayesian belief networks for change impact analysis in architecture design. *Journal of Systems and Software*, 80(1):127–148, 2007.

[Top] Top Quadrant Inc. Towards Executable Enterprise Models: Building Semantic Enterprise Architecture Solutions with TopBraid Suite. In https://www.topquadrant.com/docs/whitepapers/WP-BuildingSemanticEASolutions-withTopBraid.pdf. Accessed 26/03/2019.

[TTF79] N.M. Tichy, M.L. Tushman, and C. Fombrun. Social network analysis for organizations. *Academy of management review*, 4(4):507–519, 1979.

[UFBJ10] J. Ullberg, U. Franke, M. Buschle, and P. Johnson. A tool for interoperability analysis of enterprise architecture models using pi-OCL. In Popplewell K., Harding J., Poler R., and Chalmeta R., editors, *Enterprise Interoperability IV*, page 81–90. Springer, 2010.

[vKG03] A. von Knethen and M. Grund. QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. In *International Conference on Software Maintenance (ICSM 2003)*. IEEE, 2003.

[vSSBB10] Marlies van Steenbergen, Jurjen Schipper, Rik Bos, and Sjaak Brinkkemper. The dynamic architecture maturity matrix: Instrument analysis and refinement. In A. Dan, F. Gittler, and F. Toumani, editors, *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, volume 6275 of *Lecture Notes in Computer Science*. Springer, 2010.

[vWWH+11] J. van't Wout, M. Waage, H. Hartman, M. Stahlecker, and A. Hofman. *The Integrated Architecture Framework Explained - Why, What, How*. Springer, 2011.

[W3C13] W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C Recommendation, 2013.

[W3C15] W3C. Semantic Web. In `https://www.w3.org/standards/semanticweb/`, 2015. Accessed 26/03/2019.

[WC12] H. Wan and S. Carlsson. Towards an understanding of enterprise architecture analysis activities. In *6th European Conference on Information Management and Evaluation*. Academic Conferences, 2012.

[Wet18] J. Wetterich. Fallstudie zur Qualitätssicherung der Schnittstellenkommunikation in Microservice-Systemen. Bachelor thesis, University of Augsburg, 2018.

[WF06] R. Winter and R. Fischer. Essential layers, artifacts, and dependencies of enterprise architecture. In *10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 06)*. IEEE, 2006.

[Yen09] V.C Yen. An integrated model for business process measurement. *Business Process Management Journal*, 15(6):865–875, 2009.

[YSD06] E. Yu, M. Strohmaier, and X. Deng. Exploring intentional modeling and analysis for enterprise architecture. In *10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 06)*. IEEE, 2006.

[Zac87] J. A. Zachman. A framework for information architecture. *IBM System Journal*, 26(3):276–295, 1987.

[Zac08] J.A. Zachman. The Concise Definition of The Zachman Framework. In `http://www.zachman.com/about-the-zachman-framework`, 2008. Accessed 26/03/2019.

[ZNL17] U. Zdun, E. Navarro, and F. Leymann. Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns. In Maximilien M., Vallecillo A., Wang J., and Oriol M., editors, *Service-Oriented Computing. ICSOC 2017*, volume 10601 of *Lecture Notes in Computer Science*. Springer, 2017.

Glossary

A2F Architecture Analysis Framework
ADM Architecture Development Method
AHP Analytic Hierarchy Process
Arla Architecture Analysis Language
APM Application Performance Monitoring
AUTOSAR AUTomotive Open System ARchitecture
BBN Bayesian Belief Networks
CEO Chief Executive Officer
CIO Chief Information Officer
CSV Comma-separated Values
CTO Chief Technology Officer
DA Domain Architecture
DFA Data-flow Analysis
DM2 DoDAF Meta Model
DoDAF Department of Defense Architecture Framework
DSL domain specific language
EA Enterprise Architecture
EAM Enterprise Architecture Management
EAP Enterprise Architecture Planning
EID Extended Influence Diagrams
EMF Eclipse Modeling Framework
FEAF Federal Enterprise Architecture Framework
GERAM Generalised Enterprise Reference Architecture and Methodology
ArchiMate GM ArchiMate Generic Model
GMM Generic Meta Model
IAF Integrated Architecture Framework
IDEAS International Defence Enterprise Architecture Specification
IRI International Resource Identifier
IT information technology
KPI Key Performance Indicator
KTH Royal Institute of Technology
MAF Model Analysis Framework
MEMO Multi-prespective Enterprise Modeling

MODAF Ministry of Defense Architectural Framework
MOF Meta Object Facility
NAF NATO Architecture Framework
OCL Object Constraint Language
OWL Web Ontology Language
PA Project Architecture
PRM Probabilistic Relational Models
p-OCL probabilistic OCL
RMI Remote Method Invocation
RDF Resource Descripion Framework
RDFS Resource Descripion Framework Schema
SPARQL SPARQL Protocol And RDF Query Language
SQL Structured Query Language
TOGAF The Open Group Architecture Framework
UML Unified Modeling Language
UUID Universally Unique Identifier
URI Universal Resource Identifier
W3C World Wide Web Consortium
XML Extensible Markup Language

List of Figures

List of Tables

Listings

A

Appendix A

A.1 Full EA model for the RentalCar company

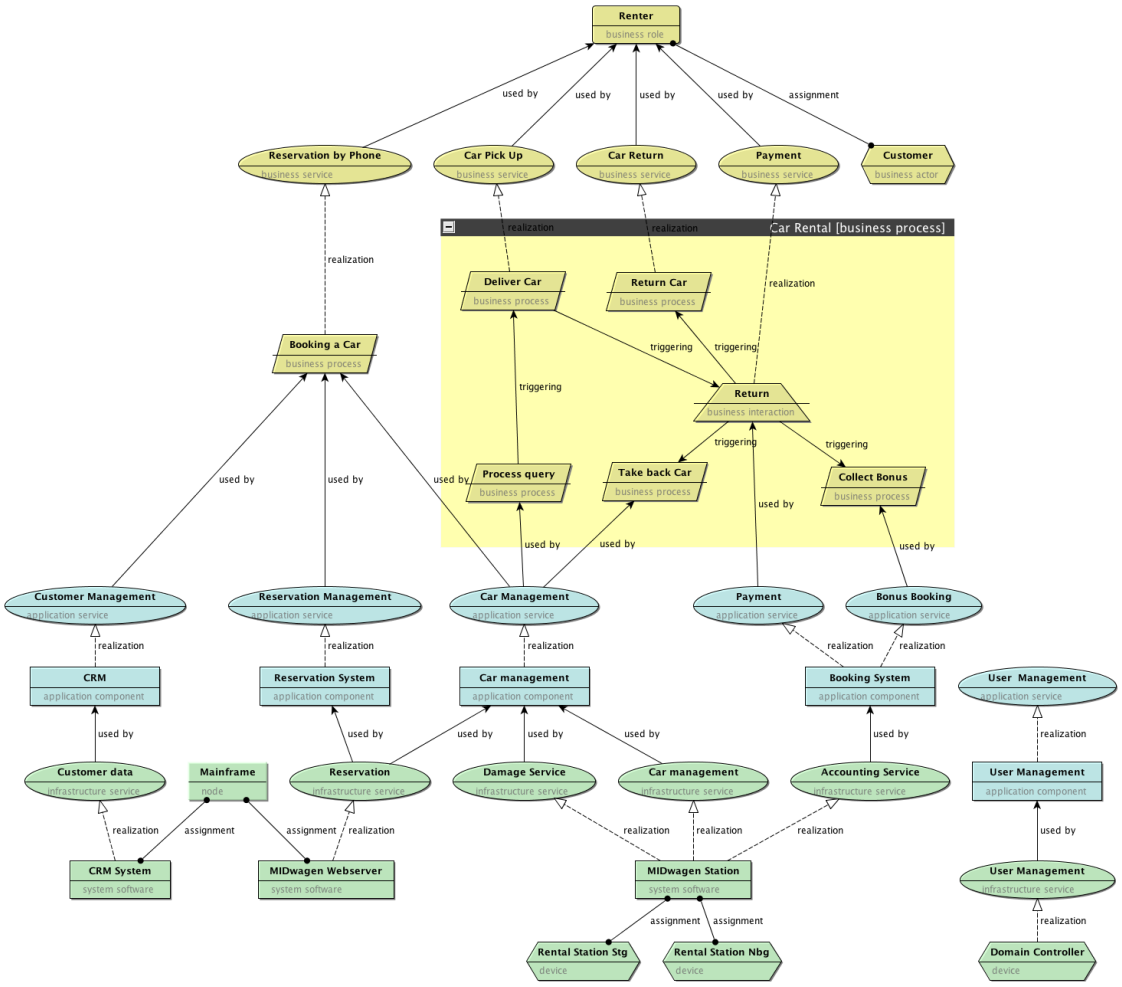


Figure A.1: Full enterprise architecture model for the RentalCar company.

A.2 SPARQL queries for gap analysis

ASK query to determine deletion candidates.

```
1  ASK
2  FROM <currentModelUri>
3  WHERE {
4    <currentElementUri>    ?relation1To   ?modelEdge.
5    ?model             model:edges    ?modelEdge.
6    ?affectedElement   ?relation2To   ?modelEdge.
7    ?model             model:elements ?affectedElement.
8    ?affectedElement   gmm:name       ?affectedElementName.
9    FILTER (?affectedElement != <currentElementUri>)
10   GRAPH <targetModelUri> {
11     ?targetModel     model:elements  ?targetResource.
12     ?targetResource  gmm:uuid        ?targetResourceID.
13   }
14   FILTER EXISTS {
15     ?affectedElement gmm:uuid   ?targetResourceID.
16   }
17 }
```

Listing A.1: Query to determine deletion candidates.

Predecessor proposals based on name and type equality.

```
1  SELECT DISTINCT ?proposedPredecessor
2  FROM <currentModelUri>
3  WHERE {
4    GRAPH <targetModelUri> {
5      <targetElementUri>   gmm:name          ?targetElementName.
6      <targetElementUri>   model:stereotype  ?targetStereotype.
7      ?targetStereotype    gmm:uuid      ?targetStereotypeUUID.
8    }
9    ?currentModel           model:elements     ?proposedPredecessor.
10   ?proposedPredecessor    gmm:uuid           ?predecessorUUID.
11   ?proposedPredecessor    gmm:name           ?targetElementName.
12   ?proposedPredecessor    model:stereotype   ?predecessorStereotype.
13   ?predecessorStereotype  gmm:uuid           ?targetStereotypeUUID.
14   FILTER NOT EXISTS {
15     GRAPH <targetModelUri> {
16       ?targetModel     model:elements  ?targetResource.
17       ?targetResource  gmm:uuid        ?predecessorUUID.
18     }
19   }
20 }
```

Listing A.2: Query to determine predecessor proposals based on name and type equality.

Predecessor proposals according to the source context of a relation

```
1  SELECT  ?proposedPredecessor
2  FROM <currentModelUri>
3  WHERE {"
4    ?currentEdge            model:source      ?currentElement.
5    ?currentEdge            model:target      ?proposedPredecessor.
6    ?currentElement         gmm:uuid          ?currentElementUUID.
7    ?proposedPredecessor    gmm:uuid          ?proposedPredecessorUUID.
8    ?proposedPredecessor    model:stereotype  ?predecessorStereotype.
9    ?predecessorStereotype  gmm:uuid          ?predecessorStereotypeUUID.
```

```
10    GRAPH <targetModelUri>{
11      ?targetEdge       model:source    ?targetElement.
12      ?targetEdge       mode:target     <targetElementUri>.
13      ?targetElement    gmm:uuid        ?currentElementUUID.
14      <targetElementUri>  model:stereotype    ?stereotype.
15      ?stereotype       gmm:uuid        ?predecessorStereotypeUUID.
16    }
17    FILTER NOT EXISTS {
18      GRAPH <targetModelUri> {
19        ?targetResource    gmm:uuid    ?proposedPredecessorUUID.
20      }
21    }
22  }
```

Listing A.3: Query to determine predecessor proposals according to the source context.

Predecessor proposals according to the target context of a relation

```
1   SELECT  ?proposedPredecessor
2   FROM <currentModelUri>
3   WHERE {
4     ?currentEdge            model:target      ?currentElement.
5     ?currentEdge            model:source      ?proposedPredecessor.
6     ?currentElement         gmm:uuid          ?currentElementUUID.
7     ?proposedPredecessor    gmm:uuid          ?proposedPredecessorUUID.
8     ?proposedPredecessor    model:stereotype  ?predecessorStereotype.
9     ?predecessorStereotype  model:uuid        ?predecessorStereotypeID.
10    GRAPH <targetModelUri> {
11      ?targetEdge       model:target  ?targetElement.
12      ?targetEdge       model:source  <targetElementUri>
13      ?targetElement    gmm:uuid      ?currentElementUUID.
14      <targetElementUri>   model:stereotype    ?stereotype.
15      ?stereotype       gmm:uuid      ?predecessorStereotypeID.
16    }
17    FILTER NOT EXISTS {
18      GRAPH <targetModelUri> {
19      ?targetResource    gmm:uuid    ?proposedPredecessorUUID.
20      }
21    }
22  }
```

Listing A.4: Query to determine predecessor proposals according to the target context.