



universität
wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Data Integration and Cleansing for Enterprise
Architecture Analytics

verfasst von / submitted by

Mag. Christoph Moser

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2017 / Vienna, 2017

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on the student
record sheet:

A 084 175

Dissertationsgebiet lt. Studienblatt /
field of study as it appears on the student record sheet:

Dr.-Studium d. Sozial- u. Wirtschaftswiss.
UniStG Wirtschaftsinformatik

Betreut von / Supervisor:

o. Univ.-Prof. Dr. Dimitris Karagiannis

Introduction

Acknowledgements

Now that this work is coming to an end, I would like to thank all of the people who have encouraged me. Without their help and support it would not have been possible to write this doctoral thesis. It is possible to give only some of them particular mention here.

First and foremost, I would like to express my thanks to my supervisors, Prof. Dr. Dimitris Karagiannis and Prof. Dr. Wilfrid Grossmann.

Prof. Grossman introduced me to the world of business analytics and statistical metadata management. Without his advice and guidance this work would not have been possible. I would like to express appreciation and thanks for all the support he has given me during the past years.

Prof. Karagiannis, my doctoral father, I want to thank for the endless hours of discussion guiding me on this long journey. Prof. Karagiannis sparked my interest in meta modelling and enterprise architecture management many years ago. He not only gave me scientific advice that enabled me to conduct this work, he also encouraged me to evaluate some of the discussed concepts in industrial settings. I have to thank him for all the years we have been working together.

My thanks also go to my colleagues Manuel Timotic and Wilfrid Utz with whom I have been working together in various university assignments. I have to thank both of them for sharing their knowledge and advice on building modelling tools and for the many discussions that have served to advance and improve this doctoral thesis.

I thank my colleagues from BOC Group with whom I intensively worked on the Enterprise Architecture Management Suite ADOIT for many years.

Finally, it is here that I would like to thank my family, Sabine and our sons Maximilian and Jakob for their patience, never complaining about the little time I had left for them during finalisation of this thesis and for the inspiration they gave me.

Introduction

Foreword

In the age of digital business, organizations face the challenge of reinventing themselves in order to cope with new market conditions. IT megatrends, such as artificial intelligence, big data, cloud computing, mobile internet, the internet of things and smart manufacturing make possible the delivery of completely new value propositions to the customers. Digitalizing traditional products, providing additional channels for customers and late-stage assembly to meet the demands for mass customization are only some examples. Businesses are in a position to collect data about customer behaviour, customer satisfaction and market conditions obtained from social media platforms. At the same time, customers demand valuable, personalized and digital experiences. Customers use social media platforms to inform themselves about product quality and best-price; they use review sites, the digital versions of word of mouth, to report on the advantages and disadvantages of products.

In this volatile environment, Enterprise architecture management is concerned to keep pace with these trends. Topics such as customer experience, business IT alignment, cost reduction, standardization and reduction of IT's time to market are addressed, striving for better return on existing investment and reduced risk for future investment.

Knowledge of the interdependencies between the organisations main design elements, such as strategic goals, stakeholders, business processes, products and IT resources is being recognised as a key success factor. Enterprise architecture management and more widely the application of enterprise modelling methods are management instruments that support decision making in these challenging environments. At the heart of these initiatives, one typically finds graphical modelling languages, which make possible documentation of knowledge about organisations and their business ecosystems. Once available, these models build a rich foundation for analyzing and optimizing the organizational structures, i.e. the enterprise architecture. Typical analysis scenarios uncover gaps within the EA and make possible dependency analysis as well as analysis of cause/effect relations between the central design elements of an organization and entire business ecosystems. Heterogeneity analysis makes standardization possible, leading to potential cost-savings and subsequent release of budgets, which can be invested in innovative products and services. The acquired transparency supports scenarios such as compliance management to identify misalignments to policies and regulatory regulations (e.g. Solvency II or Basel III).

Introduction

In contrast to previous years, the required analysis does not focus purely on a self-contained model base created by domain experts. It is recognized that models need to be substantiated with operative data, whether from organizational internal data, such as customer transactions and sales data, or external data from social media platforms.

Integrating EA models and EA relevant data is a major challenge. In this context, data quality and data provenance aspects concerning the analysis results are a key requirement. This thesis contributes to the support of EA endeavors in this regard. It presents DICE (Data Integration and Cleansing Environment), a method that supports data integration and cleansing in a structured way. It places a strong focus on the required metadata that provide information about data quality and data provenance aspects. In this way, doubt cast on the accurateness of analysis results can be minimized. Due to its automated approach, data engineers are to a great extent relieved from the burden of manual documentation and can focus on the actual business requirements.

At its heart, the presented method provides mechanisms and algorithms to transform data and metadata concurrently. Due to this approach, data transformations, such as selecting data, merging data and restructuring of data become comprehensible.

DICE does not consider itself an all-encompassing solution to the abundance of possibly required transformation tasks. It is conceptualized as a situational method based on a metamodeling approach. In this way, it can be adapted to actual project situations. With its meta structure it provides the foundational concepts, which can be specialized and adapted.

The thesis uses these mechanisms in the fields of Enterprise Architecture Analytics, a combined approach based on enterprise architecture management and business analytics. To this end, from the DICE core, a situational method for EAA is derived providing specifics to cope with peculiarities of EA data.

Introduction

Abstract

Enterprise architecture management (EAM) is a holistic management approach supporting organizations in effectively achieving their current and future objectives. The enterprise architecture (EA) is a conceptual blueprint of an organization or an entire business ecosystem composed of organizations. It defines the organization's main structural elements and their dependencies on each other. These blueprints are typically represented in the form of architectural descriptions. Taken together, these architectural descriptions form a valuable foundation for driving management decisions.

An often mentioned problem is the data quality of the EA descriptions. Often, self-contained descriptions on varying levels of detail are created by domain experts. A concise overview and comprehensive analysis of the EA is not possible on that basis. EA Frameworks such as TOGAF plead for architecture repositories that hold EA models instantiated from standardized metamodels. However, in practice much additional EA relevant documentation is created and not fed back into these repositories. Typically, even thoroughly curated repositories are not fully up to date and concise. Moreover, in many cases additional information about current and future business performance is required to underpin EA models and to make EA analysis possible. Business Analytics is the management domain focusing on the provision of knowledge extracted from operational data. The amalgamation of EA models and operational data offers the best of both worlds: EA models are substantiated by findings extracted from operative business data and BA data can be structured along the most important design objects of the organization. This approach is referred to as Enterprise Architecture Analytics in recent scientific publications.

With DICE (Data Integration and Cleansing Environment) this doctoral thesis introduces a method for integrating and streamlining EA descriptions for further analysis and for extraction and integration of operational data. DICE is a general purpose method for the support of data preparation in BA endeavors. It is conceptualized as a situational method; it can thus be refined to support domain specifics where needed. To this end, DICE builds heavily on a metamodeling approach, which makes possible the specialization of DICE to support Enterprise Architecture Analytics. Besides metamodeling, DICE is built on concepts from the fields of statistical metadata management, workflow management and data mining.

Introduction

It supports all typical transformation tasks, such as data integration, data selection, data imputation and data cleansing. Thereby, it makes possible the concurrent transformation of data and metadata. In this way, the impact on the data quality through the transformations becomes computable. Moreover, the conducted transformations are accurately documented so that resulting data can be traced back to its sources.

In the context of EAA it is used for cleansing and integration of architecture descriptions as well as for enriching the architecture descriptions with operational data for subsequent analysis and decision making. To this end, DICE is specialized by deriving and assembling new transformation task types, i.e. reusable method chunks able to cope with peculiarities of EA data. Evaluation of existing EA analysis techniques and a concise analysis of the nature of EA data lay the groundwork from which EAA-specific requirements are derived. From these requirements the EAA method chunks are conceptualized.

For evaluation purposes, *DICE* and the situational method *DICE for EAA* are prototypically implemented based on a metamodeling platform. The utility of the method is proven by presenting a use case where typical EA models and EA-relevant datasets are integrated with operational business data.

Introduction

Zusammenfassung

Unternehmensarchitekturmanagement ist ein ganzheitlicher Managementansatz und verfolgt das Ziel, Unternehmen bei der Definition und Umsetzung ihrer Strategien zu unterstützen. Die Unternehmensarchitektur bildet dabei den konzeptionellen Blueprint, welcher die wesentlichen Gestaltungselemente des Unternehmens sowie deren Abhängigkeiten beschreibt. Diese Blueprints werden typischerweise in Form von Architekturmodellen beschrieben. Die Architekturmodelle unterstützen sowohl bei der Entscheidungsfindung zur strategischen Ausrichtung des Unternehmens, also auch bei taktischen und teilweise auch operativen Entscheidungen.

Aktualität und Qualität dieser Modelle ist ein entscheidender Erfolgsfaktor für jede Architekturinitiative. Um die Modelle in strukturierter und auswertbarer Form bereitstellen zu können, empfehlen Architekturrahmenwerke wie beispielsweise TOGAF die einheitliche Nutzung eines Metamodells – einer Art Ordnungsrahmen für die zu beschreibenden Konstrukte. Des Weiteren wird empfohlen, die Modelle in einem Architektur-Repository zu verwalten und aktuell zu halten. In der Realität ist dies allerdings schwer durchzuhalten. In der Regel entstehen zahlreiche weitere Dokumentationen und Architektur-relevante Inhalte, welche nicht in einem derartigen Repository eingepflegt werden und oftmals nicht dem vereinbarten Metamodell entsprechen.

Darüber hinaus werden bei diesem eher klassischen Ansatz zum Architekturmanagement operative Unternehmensdaten, welche bei der Entscheidungsfindung über die zukünftige Unternehmensarchitektur dienlich sein können bzw. Architekturentscheidungen untermauern können, oftmals gänzlich vernachlässigt.

Business Analytics ist die Managementdomäne, welche auf die Analyse operativer Daten fokussiert. Zielsetzung ist es, neue Erkenntnisse basierend auf operativen Unternehmens- und Marktdaten zu gewinnen, um das Unternehmen besser steuern zu können. Die Kombination von Unternehmensarchitekturmanagement und Business Analytics verspricht das Beste aus beiden Welten: Unternehmensarchitekturmodelle können mit operativen Daten plausibilisiert werden und Business Analytics Daten werden entlang der wichtigsten Gestaltungselemente des Unternehmens strukturiert, sodass weitreichendere Zusammenhänge über alle Ebenen des der Organisation hinweg abgeleitet werden können. Dieser kombinierte Ansatz wird neuerdings als Enterprise Architecture Analytics bezeichnet.

Introduction

Mit der Methode DICE (Data Integration and Cleansing Environment) trägt die vorliegende Dissertation dazu bei, Unternehmensarchitekturmodelle und operative Daten integrieren zu können. DICE ist als situative Methode konzipiert, d.h. die Methode kann einfach an projektspezifische Gegebenheiten ausgerichtet werden. Zu diesem Zweck basiert DICE auf einem metamodellierungsgetriebenen Ansatz. Die Methode unterstützt alle typischen Datentransformationen wie z.B. Datenintegration, Datenselektion, Datenimputation und Datenbereinigung. Herzstück der Methode sind Mechanismen und Algorithmen, die diese Datentransformationen nicht nur auf Datenebene, sondern auch auf Metadatenebene ermöglichen. Parallel zu den Datentransformationen werden Metadaten errechnet und mitgeschrieben, sodass die durchgeführten Transformationsschritte vom Endergebnis bis hin zur Datenquelle nachvollziehbar bleiben. Durch das integrierte Qualitätsrahmenwerk, werden zugleich auch die Auswirkungen von Transformationen auf die Datenqualität errechnet und dem Methodennutzer als Instrument zur Entscheidungsfindung zur Verfügung gestellt.

Im Kontext von Unternehmensarchitekturmanagement wird DICE zur Integration von traditionellen Architekturmodellen mit operativen Unternehmensdaten eingesetzt. DICE wird zu diesem Zweck erweitert, sodass spezialisierte Transformationen auf den oftmals graphenbasierten Unternehmensarchitekturmodellen möglich werden.

Zur Evaluierung wurde die Methode basierend auf dem Metamodellierungswerkzeug ADOxx und der Statistikplattform R umgesetzt. Der Nutzen der Methode wird demonstriert, indem aufgezeigt wird, wie typische Architekturmodelle mit operativen Unternehmensdaten integriert werden.

Introduction

Contents

1	Introduction.....	14
1.1	Motivation & Relevance	17
1.2	Problem Statement	18
1.3	Research Questions and Objectives	19
1.4	Research Approach & Research Procedure	20
1.5	Thesis Outline	26
2	Methodological and Conceptional Foundations.....	28
2.1	Enterprise Architectures and Enterprise Architecture Management.....	28
2.2	Architecture Descriptions	31
2.3	Data Mining, Knowledge Discovery in Databases and Business Analytics	35
2.4	Modelling Foundations	38
2.4.1.1	Models & Viewpoints.....	39
2.4.1.2	Modelling Methods.....	39
2.4.1.3	Modelling Language.....	40
2.4.1.4	Modelling Procedure	43
2.4.1.5	Mechanisms & Algorithms.....	44
2.5	Situational Methods and Situational Method Engineering	45
2.6	DIBA and DICE.....	47
2.7	Tying it all together.....	48
2.8	Motivating Example.....	52
3	Related Work	54
3.1	Literature Research Approach	54
3.1.1	Defining Review Scope	54
3.1.2	Conceptualisation of Topic	55
3.1.3	Literature Research & Analysis	57
3.1.4	Summary	67

Introduction

4	DICE – The Centrepiece of the EAA	68
4.1	Suggesting a Method for Data Integration and Cleansing	68
4.2	Summary	75
5	DICE - Method Conceptualization.....	76
5.1	The DICE Modelling Procedure	82
5.2	The DICE Modelling Language.....	89
5.2.1	Behaviour concepts of the DICE metamodel.....	91
5.2.2	Structural concepts of the DICE metamodel	94
5.3	DICE Algorithms and Mechanisms	99
5.3.1	Macro Level: Execution of DICE Workflows	99
5.3.2	Micro Level: Algorithms for Performing Data Transformations	101
5.3.2.1	Initialisation	104
5.3.2.2	Selection	106
5.3.2.3	Addition.....	107
5.3.2.4	Variable Removal	108
5.3.2.5	Reclassification.....	109
5.3.2.6	Consolidation.....	109
5.3.2.7	Restructure.....	111
5.4	Summary	114
6	DICE Quality Framework	115
6.1	Quality Models and Quality Issues	115
6.2	DICE Quality Profile	118
6.3	Interpreting the Quality Profile	132
6.4	Summary	133
7	Application of DICE in the Fields of EAA.....	134
7.1	Data Understanding - EAM-specific Requirements	134
7.1.1	The object-oriented nature of EA data.....	136
7.1.1.1	Catalogues	136

Introduction

7.1.1.2 Matrices	138
7.1.1.3 Diagrams.....	141
7.1.1.4 Data Structures for EA Analysis based on Probabilistic Relation Models...	146
7.1.1.5 Multi-criteria Decision Making Methods	147
7.1.1.6 The Data Structure for Indicator-based EA Analysis	147
7.1.1.7 EA Analysis Based on Network Measurements	149
7.1.2 Amount of Data.....	150
7.1.3 Issues with Structural and Technical Formats	150
7.1.4 Granularity, Generality and Abstractness of EA Data	152
7.1.4.1 Reflexive Relations.....	153
7.1.4.2 Hierarchical Decomposition	154
7.1.5 Logical versus Physical Layers.....	156
7.1.5.1 Types and Instances	159
7.1.6 Time Dimensions	161
7.1.6.1 Architecture Increments	162
7.1.6.2 Versioning of Building Blocks	165
7.1.6.3 Versioning of Architecture Descriptions	167
7.1.7 Planning Scenarios - Alternative Architectures	167
7.1.8 Heterogeneous Metamodels	170
7.1.9 Common Data Quality Issues	171
7.1.10 Summary.....	172
7.2 Extending the DICE Method Base for EAA	174
7.2.1 Restructuring EA Datasets	174
7.2.2 Equivalence Functions	178
7.2.2.1 Syntactic Similarity Analysis	180
7.2.2.2 Semantic Similarity Analysis	184
7.2.2.3 Structural Similarity Analysis	186
7.2.2.4 Attribute-based Similarity Analysis	188

Introduction

7.2.3 Method Chunks Supporting Record Linkage	189
7.2.3.1 Perform Blocking	191
7.2.3.2 Similarity Analysis	193
7.2.3.3 Consolidation of Equivalent Building Blocks	194
7.3 Contraction of Edges and Vertices – Consolidation Strategies	195
7.3.1 Consolidation for Record Linkage	197
7.3.2 Consolidation of Hierarchically Structured Building Blocks	198
7.3.3 Special Case of Reflexive Relations	199
7.3.4 Filtering and Contraction of Entire Paths	202
7.4 Heterogeneous Metamodels and EA Data Schemas	205
7.5 Summary	207
8 Evaluation Based on Prototypical Implementation	209
8.1 Structural Consistency and Efficacy - DICE Prototype Based on ADOxx	209
8.1.1 Architecture	211
8.1.2 Development	213
8.2 Evaluating Validity - Illustrative Scenario Based on the DICE Prototype	220
8.2.1 Input Datasets	220
8.2.2 DICE Process Instance	221
8.2.3 Examination	231
8.3 Summary	232
9 Conclusion and Outlook	234
9.1 Conclusion	234
9.2 Outlook	235
10 Annex	237
10.1 DICE Metamodel	237
10.2 DICE Transformations	241
10.2.1 Initialisation	241
10.2.2 Selection	242

Introduction

10.2.3	Addition	243
10.2.4	Variable Removal	243
10.2.5	Reclassification.....	244
10.2.6	Consolidation of Observable Units.....	245
10.3	Publications by the Author.....	246
11	Bibliography	249
12	List of Tables	264
13	List of Figures.....	265
14	List of Abbreviations	269

1 Introduction

Enterprise Architecture Management (EAM) is a management approach that claims to enable organisations to achieve key business goals. It is a holistic approach covering the entire organisation. Supporting it from the strategic goals via the organisations business architecture, to the technologies that enable the business architecture and business models. EAM strives to manage the organisation's complexity by providing insight into its main "building blocks" and the tight net of dependencies between them. EAM deals with current business conditions as well as the performance of organisations and seeks to support the design of its future states, the so-called target architecture. Target architectures consider inside and outside influences that impact the business. Factors, such as demographic change, "disruptive" technologies, competition and the economic and political environment play an important role when defining an organisation's future direction. Programs and projects are the vehicle for implementing the anticipated target architectures. Fig. 1 illustrates these dependencies.

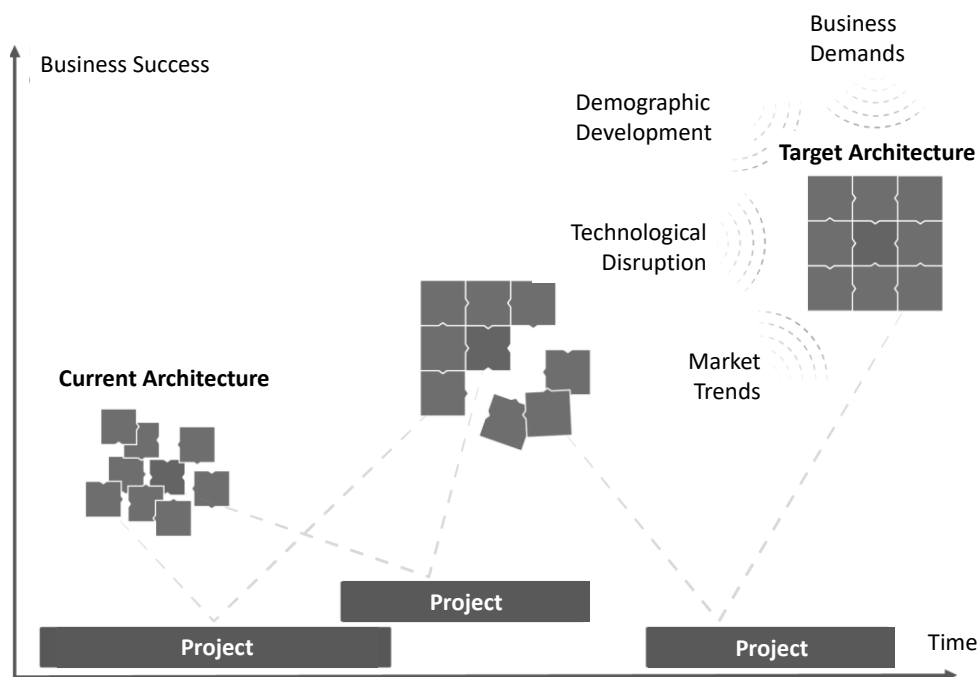


Fig. 1 Continuously evolving as-is and target architectures

At the heart of EA initiatives one typically finds graphical modelling languages that make documentation of knowledge about organisations and their business ecosystems possible (Buckl, Gulden and Schweda 2010). EA Frameworks such as TOGAF (The Open Group 2011), one of the most prominent EA frameworks and the military frameworks DoDAF (DoD

Introduction

2010) and MoDAF (Biggs 2005) define EA metamodels and EA deliverables such as diagrams based on the metamodels. Archimate (The Open Group 2016) goes one step further in that it defines an EA modelling language including recommended stakeholder-oriented viewpoints. Other EA frameworks such as FEAF (Council 1999) and E2AF (Schekkerman 2004a) offer advice on how to define meaningful metamodels by defining the required core elements or important viewpoints and other EA deliverables. Models based on EA metamodels facilitate communication to the various groups of stakeholders on all levels of the EA. These models usually represent context-specific views concerning the organisation and its environment, typically tailored to the needs of the various EA stakeholders. TOGAF proposes a so-called enterprise continuum, a classification schema implemented via an architecture repository to make these EA models available for reuse. Considerable time and effort usually goes into keeping EA models up-to-date. Approaches to data maintenance range from manual housekeeping processes to automated documentation processes.

Business ecosystems have become pervasive in the markets. Organisations act as interconnected and interdependent members of business ecosystems. They “co-evolve” their capabilities and align their investments and project portfolios to create a maximum of value for their clients. Whereas in former times EA initiatives were based on organisation-intrinsic data, nowadays cross-organisational architectures come to the fore. This is not only true for technical aspects such as interoperability needs, e.g. to ensure the exchange of information by connecting applications and IT services via adequate interfaces. All levels of architecture have to be considered. The value chains of the market players have to be aligned on the level of the business architecture, forming so-called value networks. From the standpoint of EA modelling and analysis, it is obvious that EA data collected by the relevant market participants have to be aligned and integrated to support better EA decisions.

EA data and deliverables such as EA models represent valuable input for management decisions on an operational, tactical and strategic level. Mechanisms and algorithms concerned with the extraction of previously unseen and "interesting" information from existing EA descriptions, be it organisation-intrinsic or cross-organisational data, are only seldom in place (Niemann 2006). In accordance with (Fayyad et al. 1996) this thesis defines “interest” as *“an overall measure of pattern value, combining validity, novelty, usefulness and simplicity”*. In this context, data mining, particularly the fields of **Exploratory Data Analysis**, is concerned with the recognition of concealed dependencies between categories of

Introduction

data (Coenen, Goulbourne, and Leng 2004) such as the data residing in an EA repository and any other EA relevant data sources.

To reveal these dependencies, enterprise architecture methods must be extended with data analysis mechanisms. These mechanisms will make possible the extraction of patterns of an organisation's business development and its management of resources in the past. By combining with external data sources such as market and technology trends, predictive models can be used to facilitate decision making based on the EA descriptions. Admixing the EA models with external data, such as industry reference models, industry benchmarks and data from the businesses market will support this endeavour (Neaga and Harding 2005). The backbone of the required BA mechanisms and algorithms are the metamodels and data structures of the EA deliverables (e.g. architecture definition documents) and EA relevant input documents.

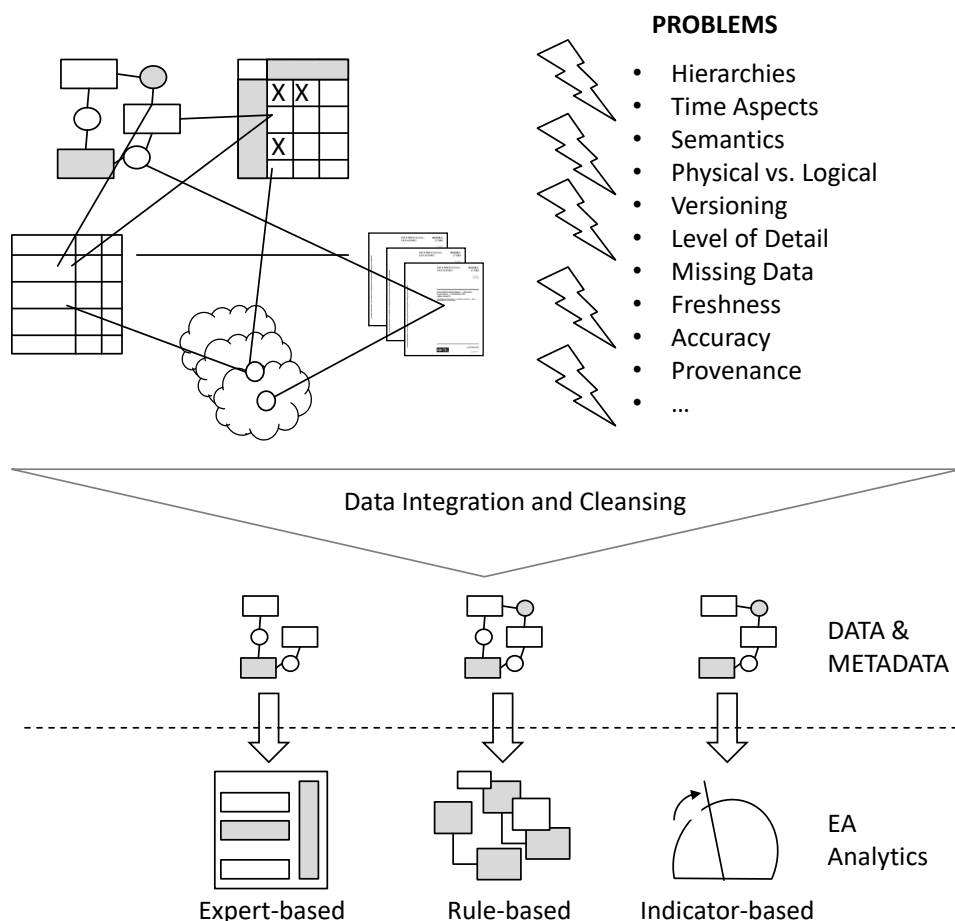


Fig. 2 Preparing EA data for EA decision making

Introduction

Integrating these often self-contained data sources in order to provide a basis for better decision making can be considered a major challenge. Typically, decision makers are not solely interested in the decision documents; the data quality and data provenance aspects are of equal importance. Decision makers need to understand where the data come from and at what level of quality.

1.1 Motivation & Relevance

An abundance of research literature in the fields of enterprise architecture, taking a wider perspective in the fields of enterprise modelling, recognizes that the quality and freshness of the (EA) data is key. EA descriptions must present a correct and current model of the real world to be valuable for decision making in the context of an organisation's strategic orientation and EA-related tactical and operational issues. At the same time, it is stated that usually the quality of available EA data is insufficient (Farwick et al. 2013), (Fischer, Aier and Winter 2007) and (Grunow, Matthes and Roth 2013).

In the face of today's and future challenges these models ideally cover the entire ecosystem that an organisation is part of. These ecosystems evolve and change rapidly with new players entering the ecosystems and others leaving them. It becomes obvious that traditional approaches to EA data maintenance (e.g. manual data acquisition) alone will not be sufficient to satisfy the requirements of data freshness, completeness and accuracy.

As uncovered by the conducted literature review (see section 3), there is little research to tackle these challenges. Existing research focuses on manual and semi-automated organisation-intrinsic maintenance of EA data based on optimised data gathering and maintenance processes. (Fischer, Aier and Winter 2007) and (Moser et al. 2009) propose process patterns for EA data maintenance. (Farwick et al. 2016) elaborate on semi-automated data integration approaches mainly focussing on integration of internally available and mostly IT architecture related data sources such as configuration management systems. (Buschle et al. 2012) propose enterprise service bus implementations as a valuable source for EA documentation. The EA relevant information extracted from the ESB is naturally of a rather technical nature and focuses on application components and their interfaces.

One major drawback of existing automated approaches is that all known applications heavily focus on the IT architecture and are limited to the current (as-is) architecture.

Introduction

Planning data and thus EA data focussing on the *architecture vision* and on target architectures are usually not the focus.

1.2 Problem Statement

Methods are required that make possible the integration of EA-relevant data into a sound basis for EA analysis. As will be discussed, EA analysis typically requires the data in structured form and typically in compliance with a defined metamodel. Looking into EA frameworks and real-life EA initiatives, one can see that a lot of relevant EA data are only available in semi-structured formats. These are typically documents at large adhering to an agreed template. Examples from TOGAF are: architecture contracts, architecture definition documents, architecture requirements specifications and so forth. Furthermore, external data sources, such as regulatory requirements, reference architectures, and sector benchmarks carry EA-relevant data and are ideally taken into account in EA decision making.

Aggravating this situation, data integration and clearance in the context of EA has to deal with manifold data sources and data formats. Once the relevant data such as building blocks and relations between these building blocks have been extracted, one is typically confronted with inconsistent semantics of hierarchies of building blocks, different types of relations between building blocks and complex and transitive dependencies (Kurpjuweit and Aier 2009). Different levels of abstraction, versions and variants of building blocks and EA models as well as the required handling of complex time-aspects add additional complexity.

Besides the typical EA descriptions that allow for structural analysis of dependencies on all levels of the organisation (business view, application view, technology view etc.), operational business data as well as external data, e.g. data obtained from social media platforms have to be considered. Structured along the EA and aggregated to enable business-insights, these data sources play an important role for accurate decision making on all management levels. Evidence can be found in (Potts 2010) who claims to consider structural performance ratios (e.g. based on KPIs such as operating income per unit of staff, profit per transaction and revenue per unit of operating expense) to guide EA endeavours. (Laverdure and Conn 2012) plead for analysis and consideration of external factors such as resource risks to support strategic EA decisions. In their case study they list examples such as “*depletion of oil*

Introduction

reserves, along with a rapidly increasing demand from developing countries” (Laverdure and Conn 2012).

Mastering these challenges not only requires methods and tools for data integration but also a means of tracing back the data integration process in order to fully understand the quality of the EA data sources used for creation of EA deliverables. This quality requirement is well known from the fields of data mining and statistics where measurement and reporting of quality aspects such as relevance, accuracy, interpretability and timeliness of used data sources and data products has always been claimed. In this vein, mechanisms are required to trace back how EA data sources have been created and the impact of data acquisition and transformation steps on data quality: a requirement that sometimes is not even met in thoroughly curated single-source architecture repositories.

To stress the importance of the data integration aspects and the consideration of data sources not intentionally created to support EA initiatives, the method is called DICE for Enterprise Architecture **Analytics**. With reference to business analytics, DICE is intended to support data preparation phases for subsequent EA analysis and EA-based decision making.

1.3 Research Questions and Objectives

This work establishes data integration and cleansing for EA data as a new research area within the fields of enterprise architecture management. It brings into focus strategies and patterns to assemble data integration and cleansing processes. Thereby, the developed method strives to be domain-agnostic, so that it can be applied to any data integration and clearance endeavour. Obviously, such a method cannot cover all possible problem areas. Thus, the method is designed to leverage the concepts of situational method engineering. It can be adapted to the domain or project-specific needs. For the Enterprise Architecture domain, the thesis provides extensions to this method.

Based on the given problem statement the following research questions (RQ) are addressed:

- **RQ1:** How can a sound dataset for EA analysis be established from heterogeneous datasets?
- **RQ2:** How can external data sources and data services be integrated in the context of ad hoc or project-related EA data requirements?

Introduction

- **RQ3:** What main transformation steps are required and how can these transformations be traced systematically to ensure data provenance?
- **RQ4:** What metadata has to be collected? What is the structure of this metadata and how can it be generated automatically or with little human effort?

It is hypothesized that a situational method which makes possible explicit design and representation of executable data integration and cleansing processes enables the generation of EA datasets meeting the quality standards. (Semi-)automated documentation of metadata enforcing data provenance and data quality metrics plays an important role in this context. Thus, the above stated research questions lead to the following three research objectives:

- the design of a universal method for data integration and cleansing that can be tailored to specific problem domains and project-specific needs,
- the provision of a structural analysis of the nature of EA data and
- the design of a situational method for data integration and cleansing for the problem domain of EAM.

To reach these objectives, application of business analytics and data mining techniques is intended to serve the following purposes:

- DM techniques are applied to EA descriptions to build up a concise dataset for (ad hoc) EA analysis.
- DM techniques are used to extract relevant EA building blocks, relations among themselves, and descriptive attributes from operational data (e.g. product catalogues to setup product component models) or from external sources (e.g. regulations the organization has to adhere to).
- DM techniques are used to combine EA data and operational data (e.g. performance values) usually not held in EA models.
- Statistical metadata management techniques are considered to integrate the required data provenance and data quality aspects.

1.4 Research Approach & Research Procedure

The stated research problems concern a complex system where data, processes, human stakeholders and technologies form a tight net of dynamic interdependencies. Central to the

Introduction

problem is a wide range of sometimes unknown data, often residing in semi-structured formats that have to be integrated. The diverse facets of the problem require a pragmatic research approach. “Design science” has been selected as the research approach, which ensures that the defined problem is researched in a systematic way by applying guidelines and roadmaps as well as evaluation criteria throughout the entire research process.

For the thesis the author follows the design science research paradigm of (Hevner et al. 2004). In design science research, besides models: constructs, instantiations and methods are among the viable artefacts. Methods are considered to “*describe viable ways of performing goal-oriented activities in order to solve a real-world problem*” (Bucher and Winter 2008). Obviously, there will not be a “one-size-fits-all” method covering the specific data integration and clearance requirements addressed in the research objectives. The method must be configurable and extendable to organisation-specific and project-specific needs. Methods addressing such requirements are called “situational methods” and stem from the research field of the same name: *situational method engineering*. According to (Kumar and Welke 1992), method engineering is a way of developing and implementing methods. Situational method engineering can be understood as a particular subarea focussing on the development and implementation of project-specific methods. This thesis adopts these principles for the fields of EA data preparation.

The main artefact presented in this research is a situational method for data integration and clearance (DICE) in the fields of EA. At its core the method is generic and can be applied to any data integration and clearance problem. In this research it is refined for the area of EA modelling and analysis. It is not meant to be a substitute for proven methods for collection and provision of EA data. Moreover, it is intended to contribute to these methods in order to create sound data sources for subsequent EA analysis in an efficient way. Comparable with data warehouse strategies, DICE can be based on a supply chain metaphor: EA content is collected, stored and delivered via “data marts” to the consumers for further processing. Other than traditional EA sources such as EA repositories, DICE strives to build up a virtual data set for the given individual problem case. The main requirement of this virtual dataset is to enhance the original datasets (some of them might be extracted from an EA repository) applying sequences of transformation steps, which are invoked and executed on demand. Rather than aiming for provision and storing of a unified view of the EA data (typically represented in a static global schema), the virtual dataset comprises the transformation

Introduction

processes together with the original data sources in order to transform the data in an ad hoc manner to the need at hand. This definition coincides with the definition of virtual datasets in (Stephan, Vckovski and Bucher 1993), who define virtual datasets as datasets holding virtual data that is non-persistent but computed on demand at runtime.

In the context of EA, virtual datasets consist of two or more heterogeneous datasets that need to be integrated. In this context the term heterogeneous emphasises the fact that the input datasets can stem from arbitrary sources and will not follow a global schema. Fig. 3 illustrates the steps to be taken from the initial EA knowledge requirements to the data and metadata ready for EA analysis. The centrepiece is DICE, a process-oriented method for data integration and cleansing, providing the EA datasets accompanied by their metadata such as quality and data-provenance data.

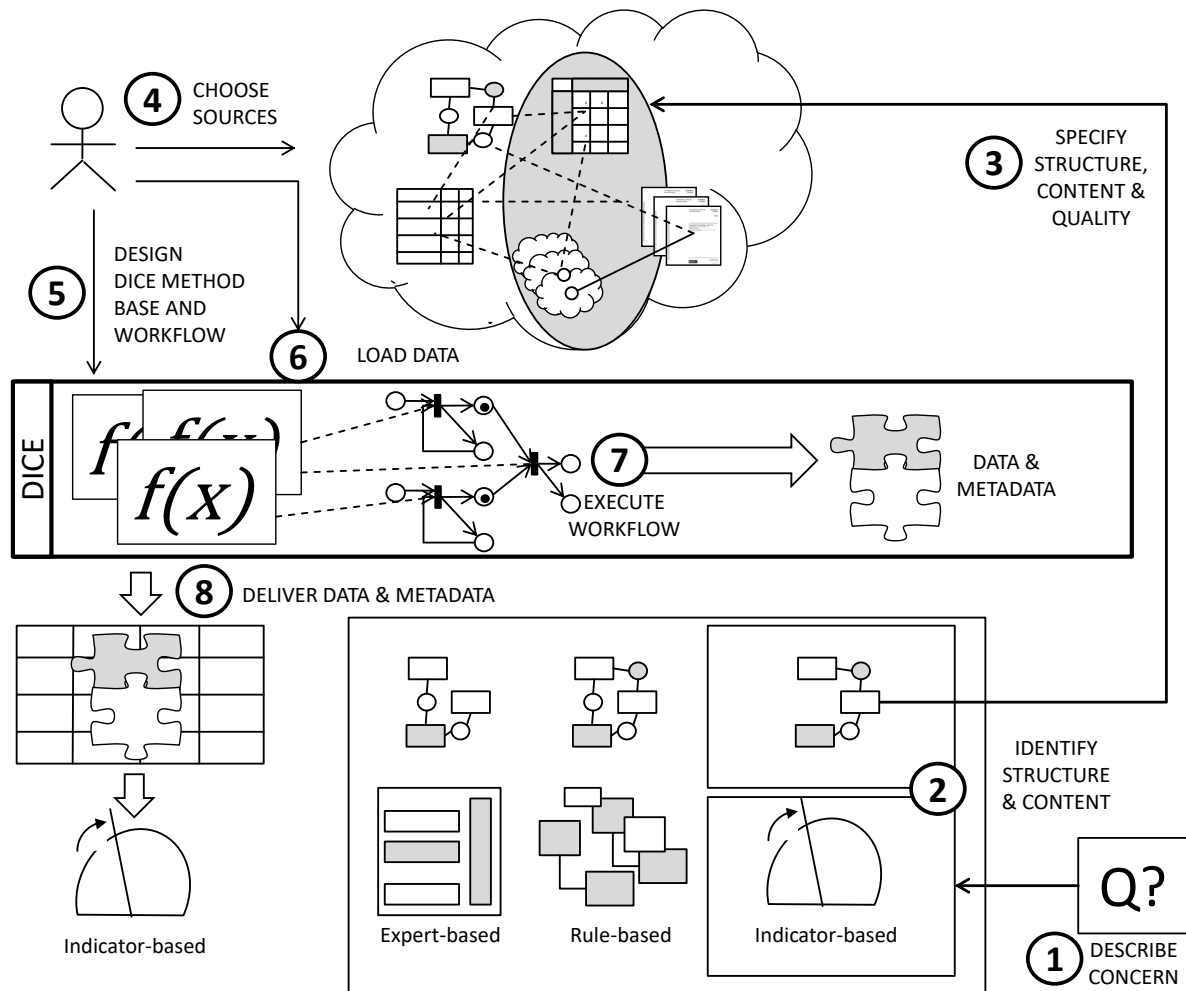


Fig. 3 From heterogeneous datasets to a sound basis for EA analysis

Introduction

As there is no generally accepted EA standard, the thesis is based on TOGAF (The Open Group 2011) and Archimate (The Open Group 2016). TOGAF is the most prominent EA framework. Archimate is a widely adopted modelling language. It is based on the ISO/IEC/IEEE 42010:2011 standard, a standard for providing guidelines for architecture descriptions (ISO/IEC/IEEE 42010 2011). It makes possible the definition of interrelated views on an organisation and its information system's infrastructure. As required, other frameworks such as DODAF (DoD 2010), MODAF (Biggs 2005), FEA (Council 1999) and E2AF (Schekkerman 2004a) are considered. Hence, the results of this contribution are intended to be applicable to EA endeavours based on any EA framework.

Research in the fields of BA provides a wealth of “method fragments”, which can contribute to resolving parts of the given problem. Examples are: techniques for restructuring and harmonisation of data sources, techniques such as part-of-speech-tagging making possible the extraction of data from plain text corpuses, techniques for record linkage, which support the resolution of duplicate data entities, etc. In data mining these methods typically are composed of a clearance and integration process representing the production process necessary to deliver the required data sources for subsequent EA analysis. The definition of the data production processes must be considered as ad hoc, but all design decisions applied to construct the resulting dataset use standard techniques, which have to be assembled in an adequate way. Besides the cleansed and integrated data and the corresponding metadata, the documentation of the required production process itself, the techniques used and the parametrisation of applied algorithms is one of the main deliverables.

Situational Method Engineering is the research discipline that focusses on defining methods that make possible the composition of project-specific methods tailored to specific and most recent needs. DICE is conceptualised as a situational method. At its core, reusable method fragments are held in the so-called method base. Users assemble these method fragments to build their own organization and/or project-specific DICE processes, which meet the demand of data provision and at the same time provide the metadata.

The method base is intended to grow continuously with the requirements by providing a means to add new or refined method fragments to the method base. According to (Harmsen, Brinkkemper and Oei 1994), situational methods can be categorised into three levels: (1) the macro level, i.e. the organisational level, (2) the meso level that is to say, the project level and (3) the micro level, which focuses on concrete method components. DICE for EAA covers all

Introduction

of these levels. On the macro level the method delivers method fragments to build data production workflows utilising continuously reoccurring clearance and integration patterns. On the project level specific EA data sources are integrated and cleansed to build up a basis for EA analysis. Finally, on the micro level, the focus lies on concrete application scenarios, where the data source is restructured, refined and filtered for concrete EA analysis tasks.

Rigor is established by adhering to Hevners principles of design science research (DSR) (Hevner et al. 2004). More specifically, (Peppers et al. 2007) design science research methodology for information systems research that follow these principles is applied. Fig. 4 provides an overview of the research process.

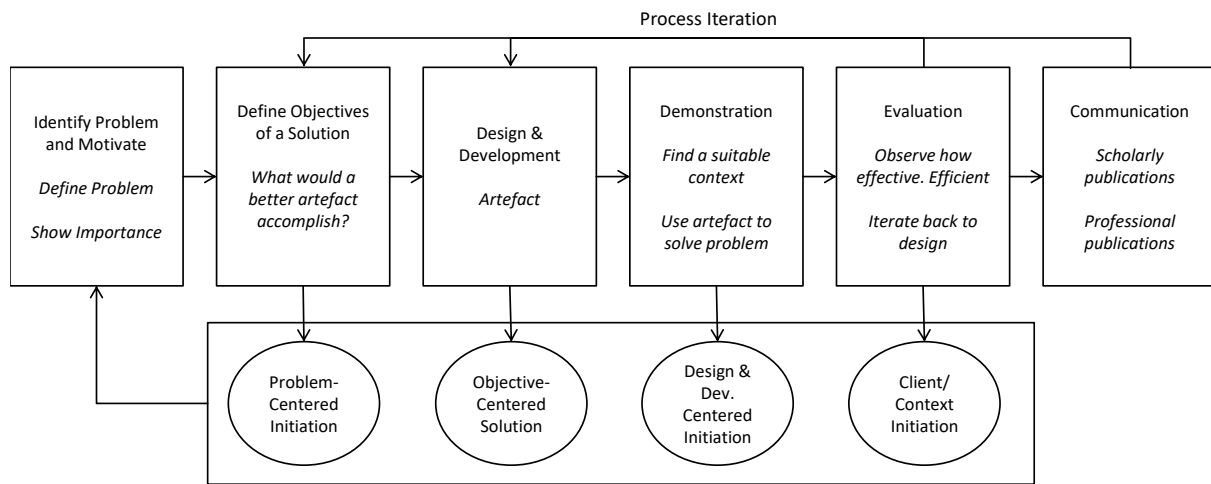


Fig. 4 Design Science Research Methodology Process Model, adapted from (Peppers et al. 2007)

The main steps of this research process also constitute the structure of the work at hand: (1) problem identification and motivation, (2) definition of the objectives of the solution, (3) design and development, (4) demonstration, (5) evaluation and (6) communication (Peppers et al. 2007).

- **Problem identification and motivation** has been addressed in section 1.1 by stating the need for the method and sketching the main requirements.
- The concrete **objectives and requirements** are addressed in section 5, where the DICE method is conceptualized. In section 6.1, from a detailed analysis of dirty data, required quality metadata are derived. Section 7.1 focuses on the concrete problem domain, which is EA. Typical structures of EA data along with transformation needs are discussed and categorized pursuant to the main elements of the DICE metamodel. Consequently, the requirements analysis occurs in two main phases: (1) requirements

Introduction

analysis for the situational method and (2) requirements analysis for applying DICE in the context of EA.

- The **design phase** is framed by design principles for situational methods shaped by (Harmsen, Brinkkemper and Oei 1994) and by the method conceptualization, development and deployment framework propagated by the open models initiative (OMI) (Karagiannis et al. 2016) and (Karagiannis 2015). From the fields of statistical metadata management, concepts for processing metadata guiding the data production process are adopted.
- To **demonstrate** the applicability of DICE for EAA, it is prototypically implemented based on the metamodeling platform ADOxx. The prototype contains all relevant components of Situational Method Engineering (SME) tools such as the method base and the construction toolkit for instantiating and combining the method fragments into a concrete method.
- The methods utility is **evaluated** in a threefold manner. Firstly, the conceptualized DICE metamodel is evaluated in terms of consistency. The main focus lies in the evaluation of consistency of the DICE inherent quality indicator calculation mechanisms. Secondly, DICE as a design science artefact is prototypically implemented. Based on the prototype, DICE is evaluated against a set of artefact evaluation criteria such as the criteria surveyed by Prat et al. (Prat, Comyn-Wattiau and Akoka 2014). These criteria are categorized into the five fundamental dimensions of systems: goal, environment, structure, activity and evolution. Feature-wise requirements are deliberately derived from (Harmsen 1997), who specifies the main components of SME tools.
Thirdly, based on an *illustrative scenario*, the applicability of the method in real-world situations is demonstrated using well-known EA datasets such as the Archimate sample EA available in the Archimate model exchange format as input.
- Finally, **communication** is conducted by this thesis itself, by presenting parts of the work in publications, see section 10.3 and by providing the prototypical implementation including sample models as a reference project on the open models community platform www.openmodels.at.

Introduction

1.5 Thesis Outline

In this section the chapters which make up the thesis, their dependencies and logical flow are presented. Fig. 5 provides an overview.

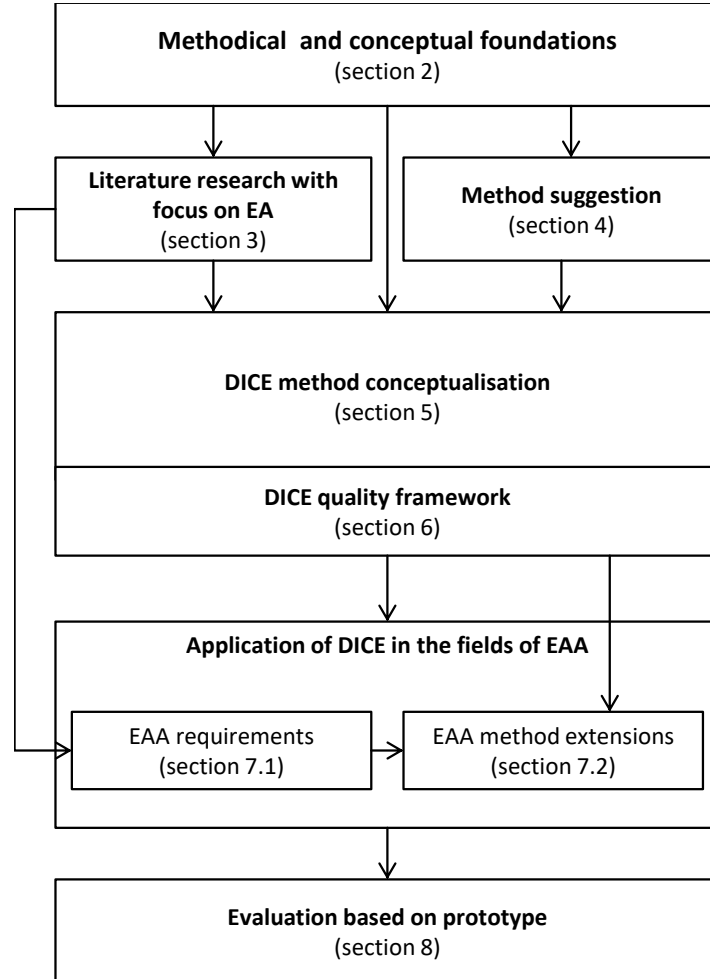


Fig. 5 Thesis Outline

Section 2 discusses the foundational basis of the DICE method with strong emphasis on method engineering approaches, methods from the fields of statistical metadata management and data mining and EA management methods. The literature research places a strong focus on current EA analysis methods and their positioning in the context of business analytics and data mining. The reviewed EA analysis approaches additionally serve as one of the main pillars for the discussion of the nature of EA data in section 7.1. Section 4 introduces DICE, suggesting a possible design solution in the form of a situational method. Section 5 is concerned with the conceptualisation of DICE under consideration of the quality framework introduced in section 6. The DICE quality framework represents a core component of DICE

Introduction

metadata layer. Section 7 introduces EEA as the context for the application of DICE. DICE as a situational method is refined to support the data preparation phases of EA analytics endeavours. To this end, the nature of EA data is analysed and requirements are derived in section 7.1. Subsequently, section 7.2 presents how these requirements can be provided by specialising DICE and extending its method base. Finally, section 8 constitutes the evaluation of DICE based on the prototypical implementation. The method has to fulfil the common requirements of SME tools and the specific requirements in the context of EAM.

2 Methodological and Conceptional Foundations

The foundations of the thesis are based on four main pillars. Firstly, enterprise architecture (management) frameworks and methods that lay the groundwork for institutionalised Enterprise Architecture Management (EAM) conducted by business organisations. Secondly, the theories of modelling and metamodeling, which can be seen as a cornerstone of EAM, providing the means to design, communicate and analyse the EA. Thirdly, methods and concepts from the fields of data mining, business analytics and statistical metadata management are relevant. They are the main building blocks for establishing integrated datasets for EA analysis. Finally, the concepts of situational method engineering lay the groundwork for creating the DICE method, the main artefact of the thesis.

The following section discusses these foundations with an emphasis on their interdependencies.

2.1 Enterprise Architectures and Enterprise Architecture Management

Better return on existing investment, reduced risk for future investment, flexibility for business growth and restructuring, more efficient IT operation and enabling new business models are only some of the value propositions of enterprise architecture management according to TOGAF (The Open Group 2011).

In conformity with the *general theory of systems* and the principles of *systems thinking*, an enterprise can be understood as a complex socio-technical system made up of people, processes and technologies (Dietz 2006). Enterprise systems engineering (ESE) is the discipline that adapts systems engineering concepts to the design of enterprises (Giachetti 2010). EA frameworks such as TOGAF embrace the concepts of ESE by providing methods and tools for designing enterprises. In essence these methods focus on the systemic decomposition of enterprises into their main building blocks for the purpose of analysis of their relationships (Roszczyk 2015).

Hitherto, there has been no generally accepted definition of the term enterprise architecture. One prominent definition, emphasizing the commonalities and characteristics of architectures in general, is provided by the ISO standard (ISO/IEC/IEEE 42010 2011), which defines *architecture* as the

Methodological and Conceptional Foundations

“fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.”

This definition focuses on the structural aspects of a system. It covers the entire system from a holistic point of view. Besides the elements and relationships amongst the system constituting elements, it emphasizes the design principles of the system.

By emphasizing the required change aspects, from analysis of the organization to the effective implementation of organizational change, the definition of the Federation of EA Professional Organizations goes one step further. It defines EA as follows:

“a well-defined practice for conducting enterprise analysis, design, planning, and implementation, using a holistic approach at all times, for the successful development and execution of strategy. Enterprise architecture applies architecture principles and practices to guide organizations through the business, information, process, and technology changes necessary to execute their strategies. These practices utilize the various aspects of an enterprise to identify, motivate, and achieve these changes” (FEAPO 2013).

In conformity with the ISO/IEC/IEEE 42010, this definition of architecture brings into focus the holistic nature of (enterprise) architectures. It considers the core elements of organizations, such as information, processes and technologies. Likewise, the definition emphasizes the importance of principles guiding design and evolution of the architecture. Notably, the definition does not, as in many other EA definitions, clearly distinguish between the practice of “architecturing”, the *“process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system’s life cycle”* (ISO/IEC/IEEE 42010 2011) and the architecture itself.

Gartner defines enterprise architecture as:

“the process of translating business vision and strategy into effective enterprise change by creating, communicating, and improving the key principles and models that describe the enterprise’s future state and enable its evolution” (Lapkin et al. 2008).

With the term “models”, Gartner refers to the documentation aspects of EA. As in many other definitions, models and more generally *architecture descriptions*, are a key delivery to support communication between the various business and IT stakeholders.

Methodological and Conceptional Foundations

Recently, EA research has paid more and more attention to cross-organisational aspects, putting emphasis on so-called business ecosystems, networks of organisations that share and align social and technical resources to create additional value for customers and improve efficiency of all stakeholders. The term *system* in the context of EA does not necessarily refer to single organizations but also to entire networks of organizations. EA endeavors might either consider the complete organization, parts of organizations or entire compounds of organizations (e.g. organized in the form of extended enterprises, virtual enterprises, networked enterprises or supply chains) (Vernadat 2014). The system boundary has to be clearly defined for any EA endeavor. Under these circumstances it becomes obvious that cross-organizational EA endeavors take precedence over the formerly isolated organizational-centric EA endeavors.

Fig. 6 summarizes the most important concepts. The system (of interest) in EA is typically an organization, a business ecosystem comprising multiple organizations or relevant parts of an organization, such as a product line, a service or software. It exhibits an architecture that is expressed in architecture descriptions. Stakeholders who have an interest in the system use parts of these architecture descriptions to gain better understanding of the system for communication purposes and for informed decision making.

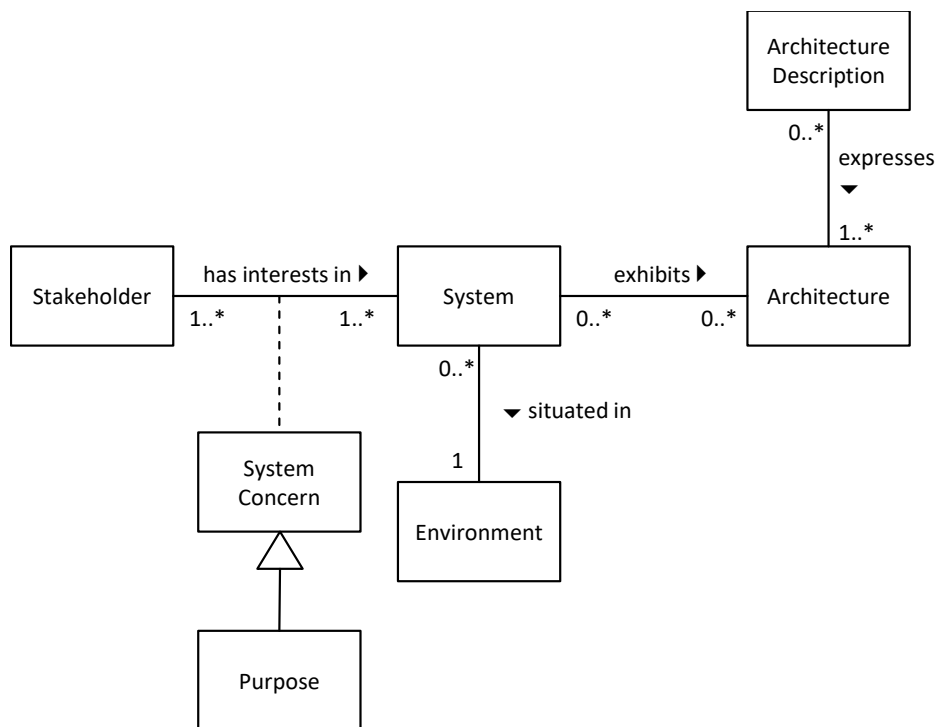


Fig. 6 Context of architecture descriptions (ISO/IEC/IEEE 42010 2011)

Methodological and Conceptional Foundations

There are an abundance of EA frameworks that give advice on how to setup an EA practice. Overviews and discussions about EA frameworks can be found, e.g. in (Leist and Zellner 2006), (Matthes 2011), (Schekkerman 2004b) and (Urbaczewski and Mrdalj 2006). Whereas some of the frameworks place a strong focus on EA deliverables, others place more emphasis on the EA processes (see e.g. the TOGAF Architecture Development Method, ADM).

2.2 Architecture Descriptions

For this thesis, architecture descriptions that can be considered the main work products of any enterprise architecture endeavor are of utmost importance. DICE takes architecture descriptions as one major input and prepares the data of these descriptions for further analysis. Thus, architecture descriptions and its constituent elements need to be examined in more detail.

ISO/IEC/IEEE 42010 is the standard that defines architecture descriptions. In its first versions (and also in the predecessor version IEEE 1471), the standard was intended for requirements definition on architectures of software-intense systems. In its latest version (released in 2011), the scope was widened to also reflect the “*creation, analysis and sustainment of architectures of systems through the use of architecture descriptions*“. The standard introduces a conceptual model as the base structure for architecture descriptions as illustrated in Fig. 7.

Methodological and Conceptional Foundations

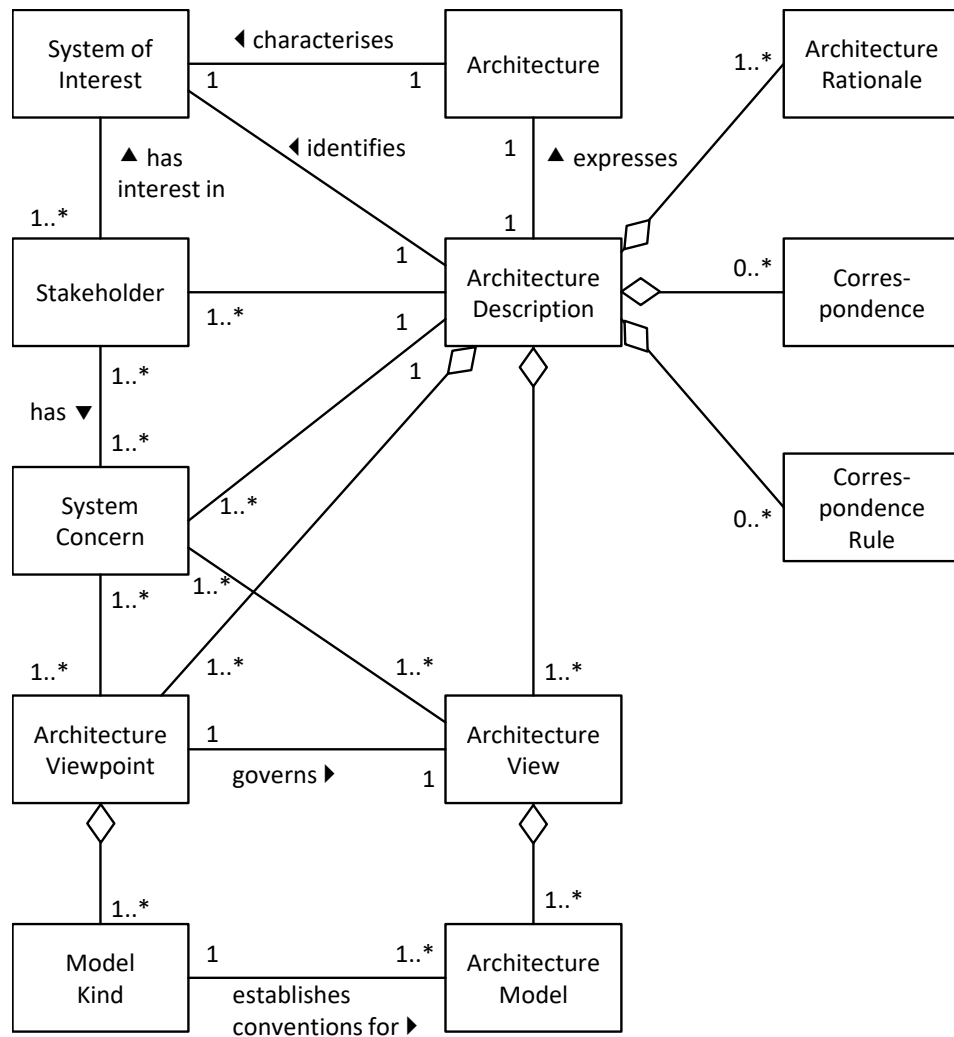


Fig. 7 Conceptual model for architecture descriptions (ISO/IEC/IEEE 42010 2011)

Table 1 presents the most important constructs of the conceptional model, referencing EA and DICE.

Table 1 Concepts for architecture descriptions

Concept	Definition ¹	Reference to EA and DICE
System-of-Interest	<i>“entities whose architectures are of interest.”</i>	Organisations, parts of organisations or entire business ecosystems.

¹ All definitions taken from (ISO/IEC/IEEE 42010 2011).

Methodological and Conceptional Foundations

Stakeholder	<i>“individual, team, organization or classes thereof having an interest in a system.”</i>	Stakeholders are humans who have key roles in, or concerns about, the system. Examples are: people having roles, such as executive, process owner, application owner, technical specialist and enterprise architect.
Concern	<i>“interest in a system relevant to one or more of its stakeholders.”</i>	Concerns allude to system aspects, such as design, development and operations. Qualities, such as flexibility, complexity, reliability, security and customer experience play an important role.
Architecture viewpoint	<i>“work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns.”</i>	Schema or templates for view construction, information that must appear in the architecture descriptions, i.e. models.
Architecture view	<i>“work product expressing the architecture of a system from the perspective of specific system concerns.”</i>	In accordance with TOGAF, a view is defined as a representation of a whole system from the perspective of a related set of concerns. Examples from TOGAF are: business architecture and technology architecture. Examples from DODAF are: All view (AV), Operational view (OV), Systems

Methodological and Conceptual Foundations

		view (SV) and Technical view (TV).
Model kind	<i>“conventions for a type of modelling.”</i>	Model kind specifies the type of architecture model. Examples are: BPMN process models, EPC models, Class diagrams and ER diagrams.
Architecture model	<i>“can be anything: (i) a model can be a concept (a “mental model”) or (ii) a model can be a work product.”</i>	Architecture models represent concrete instances. Examples are: the model of the order process of an organisation, the model of the customer data and organisational charts.
Architecture Rationale	<i>“records explanation, justification or reasoning about architecture decisions that have been made.”</i>	Documentation of architecture decisions.
Correspondence	<i>“defines a relation between architecture description elements.”</i>	Of high relevance for DICE, as correspondences can be of support in the data preparation (e.g. merging of models)
Correspondence Rule	<i>“used to enforce relations within an architecture description (or between architecture descriptions).”</i>	Rules might be defined that enable, e.g. checking consistency between different models or model kinds.

For DICE, the concept of architecture model and model kind, which can be considered as the main sources of EA datasets, is of utmost importance. These can come in a number of different structures, which DICE has to deal with (e.g. diagrams, matrices and catalogues).

Methodological and Conceptional Foundations

Examples of model kinds are TOGAFs artefacts², such as organisation/actor catalogue, application/data matrix and application communication diagram. For a better understanding, Fig. 8 depicts the main model kinds of TOGAF grouped into the phases of TOGAF ADM.

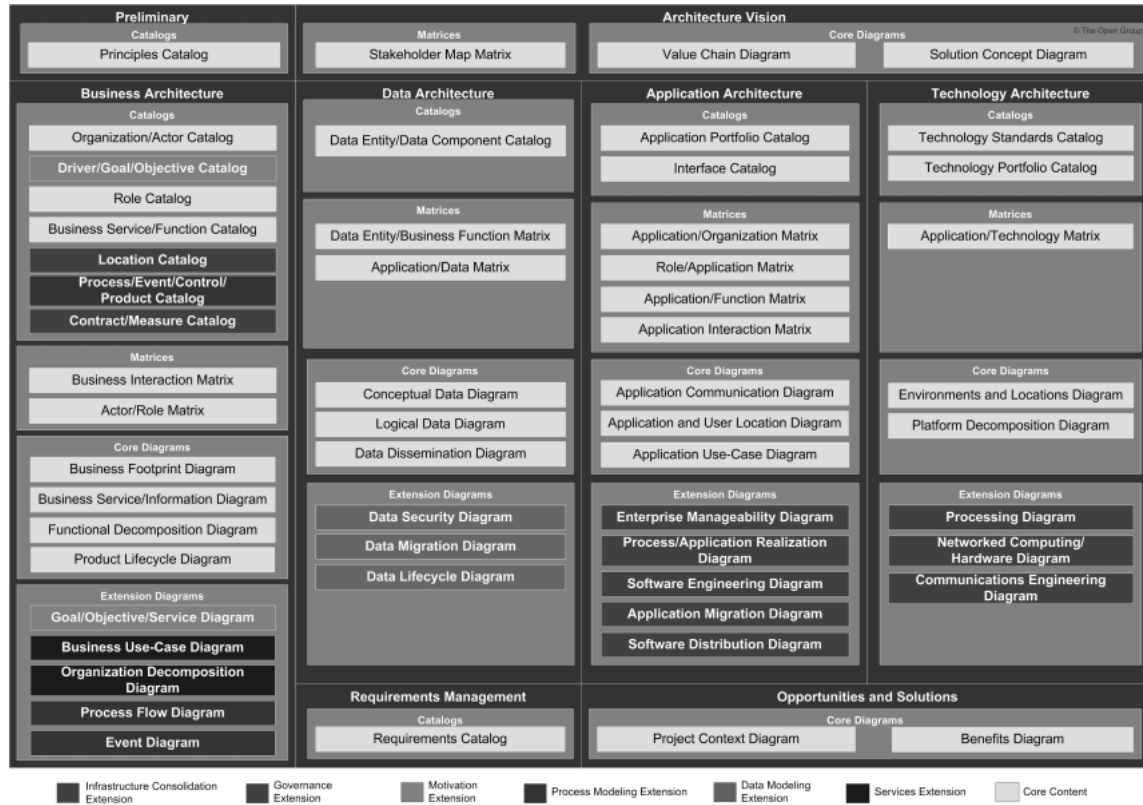


Fig. 8 Model kinds defined by TOGAF (The Open Group 2011)

2.3 Data Mining, Knowledge Discovery in Databases and Business Analytics

Knowledge discovery in Databases (KDD) and data mining (DM) are interdisciplinary domains making possible the extraction of information from large datasets and the transformation of the data into an interpretable structure for subsequent analysis. The main goal is to uncover new information and knowledge (Chakrabarti et al. 2006). According to (Fayyad, Piatetsky-Shapiro and Smyth 1996), data mining represents the required analysis task of the "knowledge discovery" process, i.e. they view data mining as a subdomain within the KDD process. Data integration can be considered as another important subdomain and is a prerequisite step to data mining. According to (Lenzerini 2002), data integration facilitates

² The term artifact appears misleading in this context. The reason for this is that TOGAF embraces, encourages but is not fully compliant with the ISO/IEC 42010:2007 standard.

Methodological and Conceptional Foundations

“combining data residing in different sources and providing users with a unified view of these data”. Based on this unified data structure, formerly unknown and potentially useful information, sets of patterns and relationships between data can be uncovered via data mining. Under these terms, business analytics has been defined as *“a set of technologies and processes that use data to understand and analyse business performance”* (Davenport 2007). With a focus on the transformation process initiated and steered by business analytics, Liberatore and Luo define business analytics as being *“more than just analytical methodologies or techniques used in logical analysis. It is a process of transforming data into actions through analysis and insights in the context of organizational decision making and problem solving.”* (Liberatore and Luo 2010). This definition provides the docking point for *enterprise architecture management*. Whereas business analytics provide the data for organizational decision making, EA is concerned with analysis of the data, planning sustainable target architectures and the steadfast support of its implementation.

Manifold research has been conducted to provide mechanisms to uncover relevant data. Statistical and mathematical techniques, such as genetic algorithms, decision trees, artificial neural networks, induction and visualization (Ghuman 2014) are just a view of the vast number of techniques for extracting the information. In their work, (Zorrilla and García-Saiz 2013) consider data warehouse (DW) techniques, the On-Line Analytical Processing (OLAP) technology, reporting tools and the data mining techniques to be the most essential components of data mining environments. Data mining environments are platforms for conducting the required transformations to convert the data into usable structures for further processing and analytical tasks. The provision of a solid starting position to perform the modelling activities (modelling in the sense of business analytics) is required in order to uncover the previously unseen information. Data have to be accessed, restructured and cleaned before conducting these analysis techniques (Rahm and Do 2000).

In recent years some efforts have been made to standardize methods and tools in the fields of KDD and DM. Prominent KDD frameworks have devoted an important part of their KDD processes to this subject, such as SEMMA (Sample, Explore, Modify, Model and Assess), (Matignon 2007) and CRISP-DM, the Cross Industry Standard Process for Data Mining (Chapman et al. 2000). SolEuNet (Data Mining and Decision Support for Business Competitiveness: A European Virtual Enterprise) is another well-known example. See the work of (Marbán, Mariscal and Segovia 2009) for a comparison of data mining & knowledge

Methodological and Conceptional Foundations

discovery process models. Polls conducted by KDnuggets (www.kdnuggets.com), an often-cited website for business analytics, big data, data mining and data science reveal that the CRISP-DM is the most acknowledged of the DM methods, followed, lagging far behind, by SEMMA.

All of the mentioned DM methods stress that a clear understanding of the existing data is a fundamental prerequisite for receiving reliable statements from the conducted mining endeavours. Thus, statistical metadata as a means of providing statements on the data quality and a feature to trace back data to their source, are considered of utmost importance. Moreover, this metadata will guide the statistical production process providing a sound starting point for the upcoming modelling phases.

As will be discussed in section 4, to be fully effective DICE has to be embedded in a superordinated KDD process such as CRISP-DM. CRISP-DM divides the process of data mining into six major phases: (1) Business understanding, (2) Data understanding, (3) Data preparation, (4) Modelling, (5) Evaluation and (6) Deployment. Fig. 9 offers an overview of the CRISP-DM process.

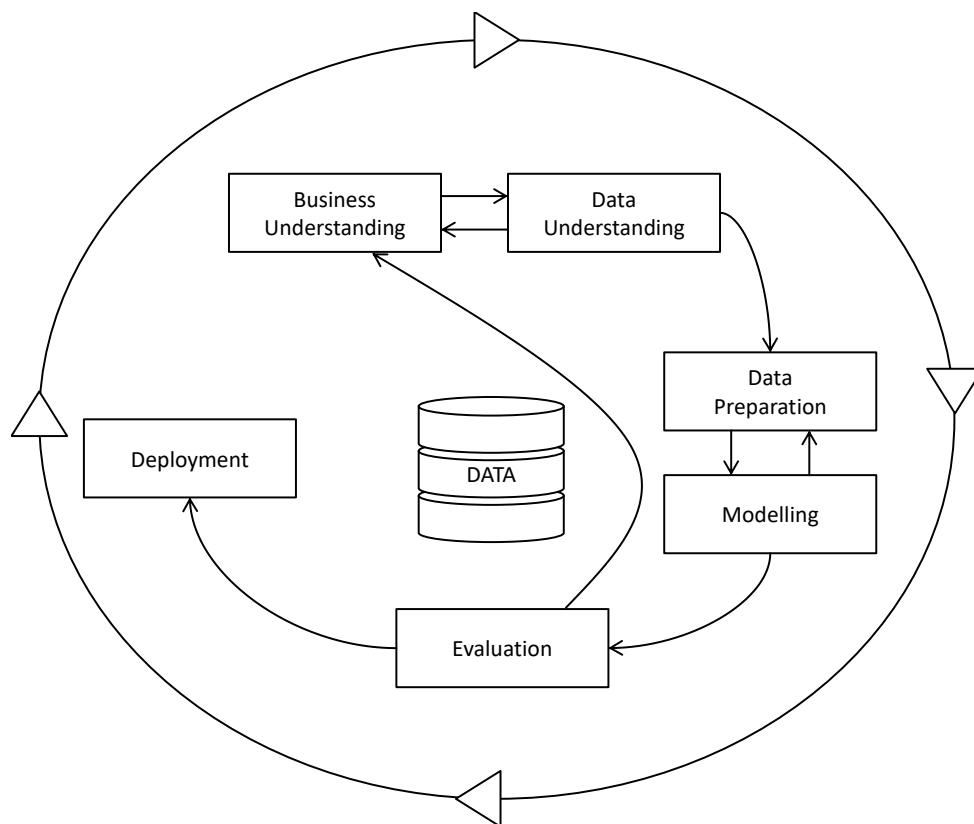


Fig. 9 CRISP-DM process (Chapman et al. 2000)

Methodological and Conceptional Foundations

- **Business understanding:** In this phase the project objectives are defined. It consists of the following tasks: determine business objectives, assess situation (e.g. requirements, risks, costs and benefits), determine data mining goals and produce a project plan.
- **Data understanding:** This phase focusses on collection and review of the available data. It comprises the following tasks: collect initial data, describe data, explore data and verify data quality.
- **Data preparation:** In this phase relevant data are selected and cleansed. Important tasks are: select data, clean data, construct data, integrate data and format data.
- **Modelling:** This phase concerns data manipulation and drawing conclusions. It comprises the tasks: select modelling technique, generate test design, build model parameter settings and assess model.
- **Evaluation:** In this phase the model is evaluated and conclusions are drawn.
- **Deployment:** In this phase conclusions are applied to the business. Additionally, the final report is created and delivered.

The DICE method supports foremost phases 1-3. Having a clear view of the business requirements (phase “business understanding”), the available datasets are examined (phase “data understanding”) and prepared for further analysis (phase “data preparation”). Data preparation tasks will include model/data enhancements and model/data integration tasks, such as dimensionality reduction, cleaning, noise/outlier removal accompanied by required mechanisms for documenting data provenance and generation of data quality measures.

Modelling and evaluation phases (phase 4 and 5) will require standard DM methods, such as artificial neural networks, decision trees, rule induction, genetic algorithms and nearest neighbour but also specialized enterprise architecture analysis techniques.

Where required, the prototypical DICE environment (DICE, see section 8) can be used to deploy a repeatable data mining process (phase 6).

2.4 Modelling Foundations

Modelling methods serve as the backbone of the present work for two reasons: (1) models are the overall accepted vehicle to represent enterprise architectures and (2) DICE itself comprises a modelling method.

Methodological and Conceptional Foundations

In the following sections modelling foundations are introduced. For a better understanding the required elements are discussed with a focus on EA modelling. However, the concepts introduced are important for both: (1) modelling the EA and (2) modelling of the required data integration and cleansing transformation tasks including the documentation of acquired metadata.

2.4.1.1 Models & Viewpoints

The concept of models, model kinds, views and viewpoints have been discussed in section 2.2. The definitions of the therein introduced ISO/IEC/IEEE 42010 standard can be matched with the main elements of modelling methods defined by Karagiannis and Kühn (see section 2.4.1.2).

With the advent of metamodeling platforms, e.g. ADOxx and Metaedit+, see (Karagiannis and Kühn 2002) for an overview, many organisations decided to build a specific metamodel for their EA endeavours or to adapt and tailor one of the standard modelling methods or languages. In many cases, EA stakeholders even chose to use a self-developed modelling method, which is often not formalised or standardised, even within the organisation. The main reason for using these “home grown” modelling languages is: inflexible and non-appropriate modelling languages for a given modelling task or simply a lack of knowledge of the appropriate modelling methods (Szegheo 2000).

Thus, in many cases EA descriptions will not adhere to standards, definitions and recommendations, e.g. introduced by ISO/IEC/IEEE 42010. EA descriptions will come in manifold formats and structures, in many cases not thoroughly instantiated from viewpoints and model kinds. However, this makes the case for data integration and cleansing of EA data (including EA models but also additional EA relevant data) even more important and challenging.

2.4.1.2 Modelling Methods

In line with (Karagiannis and Kühn 2002), a modelling method comprises: (1) a modelling language, (2) a modelling procedure and (3) mechanisms and algorithms representing the model-processing functionality.

Methodological and Conceptional Foundations

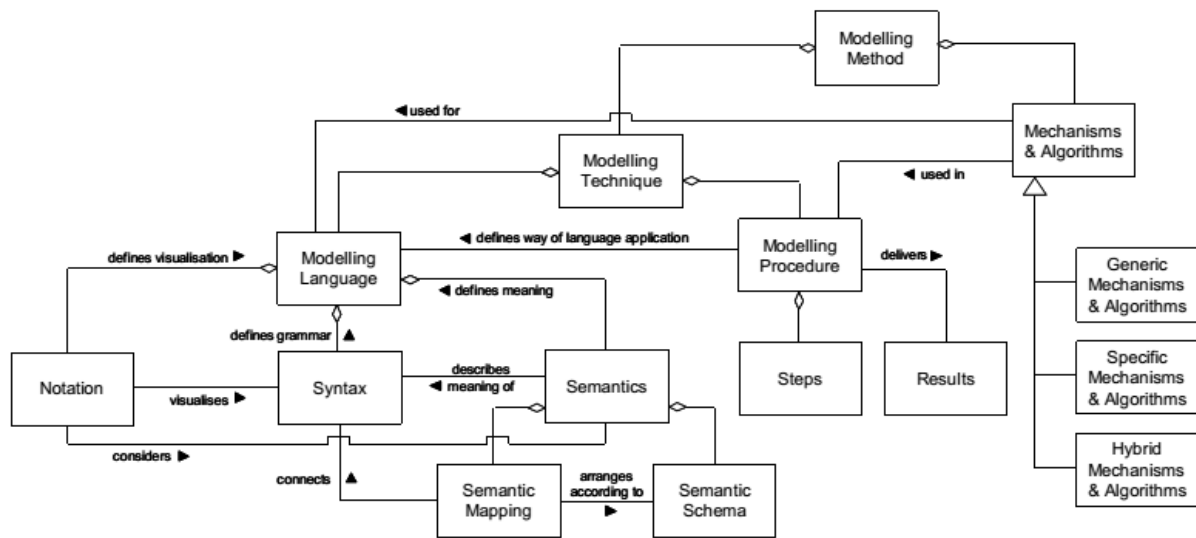


Fig. 10 The main elements of a modelling method

Fig. 10 provides an overview of these core elements of modelling methods:

- the modelling language comprises a modelling notation and a metamodel providing both a language grammar and a vocabulary,
- a modelling procedure comprising the required steps for setting up and maintaining a model base and
- mechanisms/algorithms, which in the context of EA modelling, are targeted to support quantitative evaluations and report generation based on the model base.

In the following sections these core elements are discussed in detail.

2.4.1.3 Modelling Language

The modelling language is specified by syntax, semantics and a (visual) notation. The syntax defines the grammar of the modelling language. In other words, it determines the elements of the modelling language and regulates their usage. Semantics defines the meaning of the constructs of the syntax. Via semantic mapping, the syntactical constructs are associated with their meaning, which is defined in a semantic schema.

The syntactical elements of a modelling language can essentially be reduced to the following concepts: (1) modelling classes (in the context of EAM: e.g. business process, application component and technology component), (2) association classes that define relations between two modelling classes (e.g. association, generalisation) and (3) attributes which represent

Methodological and Conceptional Foundations

properties that specify class semantics (e.g. name, cost and performance) (Visic et al. 2015). These core concepts often addressed as the *meta-elements* of modelling languages are formally discussed by OMG's (OMG 2015) meta object facility (MOF), which uses classes and associations to define the (abstract) syntax (i.e. the metamodel) of a modelling language. Besides MOF, there are manifold meta2models establishing similar primitive meta-elements; for an overview see (Kern, Hummel and Kühne 2011).

Metamodels can be understood as models of models. They are typically used for defining EA concepts provided in EA modelling languages. For an example of a metamodel in the fields of EA see Fig. 11, which illustrates a simplified version of the Archimate metamodel. For the detailed specification of the Archimate metamodel, see (The Open Group 2016).

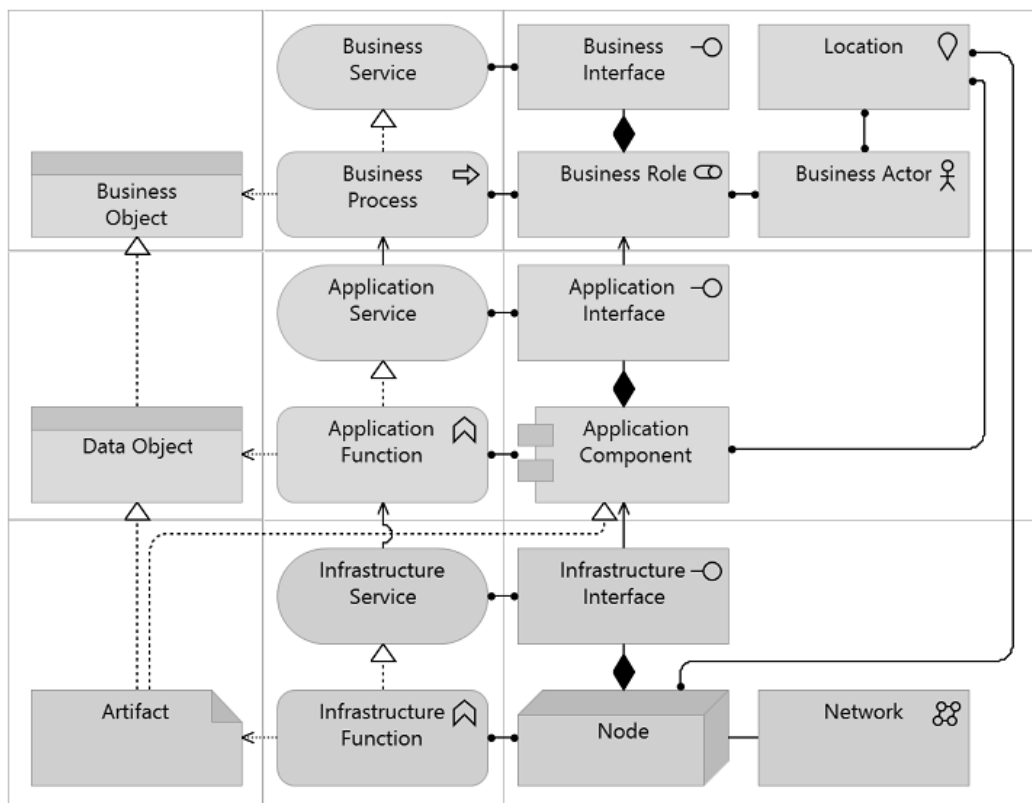


Fig. 11 Simplified Archimate Metamodel (Iacob and Jonkers 2006)

Business organisations are systems with a high degree of structural complexity and dynamics in their behaviour. Enterprise architectures are intended to represent these complex systems from different angles (EA viewpoints). To represent the architectures or relevant parts of the architectures, *models* are used. They are intended to help in mastering complexity, as they allow for abstraction of aspects that are irrelevant. These models are typically based

Methodological and Conceptional Foundations

on modelling languages that are graphical or textual languages. Modelling languages support the visualisation, specification, construction and documentation of systems and their constituent artefacts. In the context of EA, the available constructs (modelling classes) for creating models are the core elements of organisation, called building blocks (in accordance with TOGAF). The spectrum of building blocks organised within these models ranges from strategies and goals, to organisational structures and processes, to concepts of the information system architecture and underlying technological layers (Kurpjuweit and Aier 2009). These elements typically form a vast web of interconnections.

Enterprise Architecture Modelling Languages (EAML) are specialised modelling languages that are intended to represent enterprise architectural structures as well as the behaviour and organisation of enterprises. Examples of EAMLs are: the MEMO modelling languages (Frank 2002), UEMML (Anaya et al. 2010) and most notably Archimate (The Open Group 2016). General purpose modelling languages such as the Unified Modelling Language (UML) are often used for enterprise modelling, see e.g. (Fatolahi and Shams 2006) and (Zrnec, Bajec and Krisper 2001). However, as proclaimed by developers of specialised EAMLs, see e.g. (Frank 2002), general purpose languages are not specially designed for this purpose; thus, they show semantic shortcomings when used for enterprise modelling.

Archimate, as one of the most prominent EA modelling languages, embraces the ISO/IEC/IEEE 42010 standard. It comes with a set of predefined viewpoints oriented towards typical concerns of EA stakeholders. Additionally, it provides a way to define new viewpoints as needed. The centrepiece of Archimate is its metamodel that was designed to cover all important EA practical modelling tasks and at the same time stay compact. It supports a layered concept by comprising business layer, application layer and technology layer in its core. Supporting a service-oriented philosophy, the lower layers provide services which are used by the higher layers. The Archimate metamodel offers three types of concepts classified into aspects: (1) active structure elements, which perform behaviour (e.g. organisation), (2) behaviour elements representing activities (e.g. business process) and (3) passive structure elements accessed by behaviour elements (e.g. business object). Fig. 12 offers an overview of the structure of Archimate.

Methodological and Conceptional Foundations

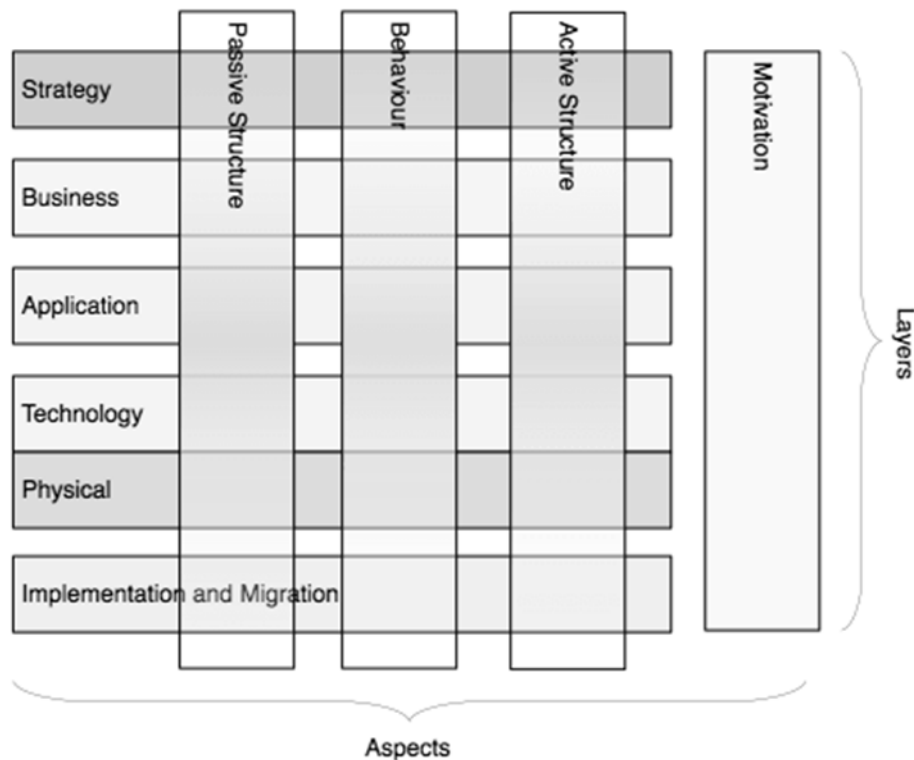


Fig. 12 Structure of Archimate 3.0 (The Open Group 2016)

Archimate models have to be constructed in conformity with the Archimate metamodel. Archimate models are represented as graphs. The vertices in the graphs represent building blocks (instantiated from the metamodel). Examples are: business process, application component and technology component. The edges depict the relations between the building blocks. Examples of relation types are: composition, aggregation and assignment.

2.4.1.4 Modelling Procedure

The modelling procedure represents the recommended steps for creating a model. The modelling procedures are usually deducted from the applied enterprise architecture frameworks and their suggested procedures. Evidence can be seen in the application of the concepts for architecture descriptions promoted by the ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 42010 2011), which recommends clear definition of the concerns of stakeholders, required viewpoints and views before starting the modelling work. The process for creating architecture models must be considered as a non-trivial task. Modelers have to examine (the relevant scope of) the enterprise and determine its structural elements, the building blocks.

Methodological and Conceptional Foundations

The structural elements have to be classified applying the concepts (modelling classes) provided by the metamodel in order to create the corresponding instances and the relations constituting the EA models (Florez, Sánchez and Villalobos 2014). Models are typically created manually by subject matter experts (e.g. enterprise architects) or stakeholders contributing to an EA endeavour, such as process owners, application and technology owners.

Concrete process proposals and patterns for modelling and maintenance procedures for EA models can be found in the work of (Fischer, Aier and Winter 2007) and (Moser et al. 2009). Fischer et al. present formal descriptions of a process and the organisational setting necessary to maintain EA content. Moser et al. define EAM process patterns and exemplarily discuss EA process patterns such as:

- centralised manual data acquisition/maintenance,
- decentralised manual data acquisition/maintenance,
- automatic data acquisition/maintenance and
- architecture control by applying a release workflow.

The modelling procedure does not only involve the creation and maintenance of the models. Additionally, and of utmost importance are: analysis, simulation and communication of the models.

2.4.1.5 Mechanisms & Algorithms

Mechanisms and algorithms provide the functionality for performing the (machine-) processing of models. Mechanisms and algorithms can come in various forms. In many cases they are applied to the model base for model evaluation purposes. Examples of algorithms are: simulation algorithms for business processes, cost calculations applied to EA models, visualisation mechanisms such as automated heatmapping applied to EA visualisations, etc.

(Karagiannis and Kühn 2002) identified three different classes of mechanisms and algorithms: (1) generic mechanisms and algorithms, which are defined on meta-level and can be applied to any modelling language, (2) specific mechanisms and algorithms, which pose prerequisites for the model-base and respectively for their underlying metamodels to be conducted successfully and (3) hybrid mechanisms and algorithms, which need to be configured for concrete modelling languages.

Methodological and Conceptional Foundations

With DICE, this thesis focusses strongly on mechanisms and algorithms. Conceptualised as a situational method, it focusses on data mining mechanisms/algorithms supporting data integration and cleansing. A strong focus is placed on EA models and EA-relevant data. However, at its core DICE is intended to work for any data preparation endeavour.

2.5 Situational Methods and Situational Method Engineering

Due to its holistic view of the enterprise, the sheer volume on EA(M) concepts covers a broad range of topics on all levels of the enterprise. Under the umbrella of EA strategic concerns, operative challenges are addressed. Under these premises, a "one-size-fits-all" EA method covering all of these topics, is not plausible.

“Depending on project type and context type, different methods – or at least different configurations or adaptations of a method are needed.” (Aier, Riege and Winter 2008a).

Situational method engineering addresses this problem through the construction or composition of new methods from existing methods. Situational Method Engineering is defined as, *“the discipline to build project-specific methods, called situational methods, from parts of the existing methods, called method fragments* (Brinkkemper 1996)”.

These methods are tailored to specific project needs. The term situational method was coined by (Harmsen 1997), who identified the need for a standardised approach to the composition of tailor-made methods to address specific organisational circumstances. Stating that *“There is no method that fits all situations”* (Harmsen 1997), Harmsen coined the phrase *“controlled flexibility”* in the context of situational methods. Situational methods are intended to be adaptable to specific project requirements in a controlled manner. At the centre of situational methods, one can find the so-called model base, holding reusable and proven “method fragments” to be composed into tailor-made methods by applying formally defined guidelines. The overall process for configuring situational methods comprises four major steps:

- characterization of situation, which is the step involving requirements gathering for the method to be constructed,
- selection of method fragments, which involves the selection of method fragments meeting the previously identified requirements,

Methodological and Conceptional Foundations

- assembly of method fragments, which involves tailoring of the selected method fragments and assembling the chosen fragments to establish the new “situation-specific” method and
- project performance, where the new situation-specific method is applied.

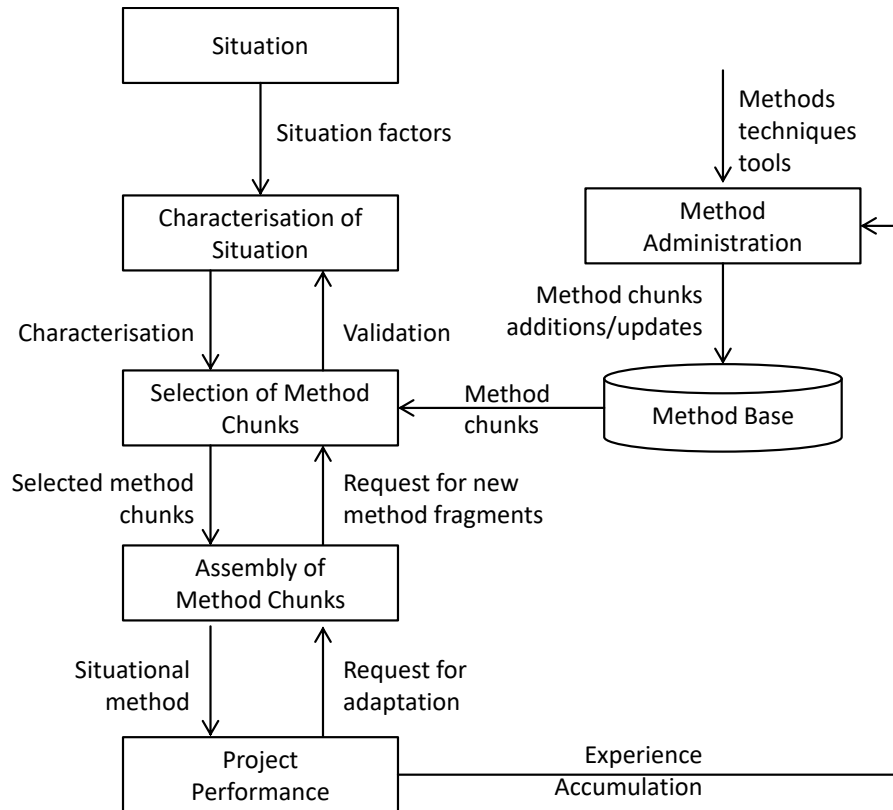


Fig. 13 The process for configuring a situational method, marginally adjusted from (Harmsen 1997)

Method fragments are the atomic elements of method construction. In DICE method chunks (compositions of method fragments) are represented in the form of transformation tasks. Transformation tasks take datasets and metadata describing the datasets as an input and generate restructured, cleansed and/or integrated datasets as well as appropriate metadata on the conducted transformations as an output. Via selection of atomic transformation tasks and by integrating these transformation tasks into an executable data preparation workflow, complex data transformations can be performed. At the same time, required metadata on these transformations are generated automatically as far as possible. Fig. 14 illustrates the structure of transformation tasks, which is discussed in more detail in section 5.

Methodological and Conceptional Foundations

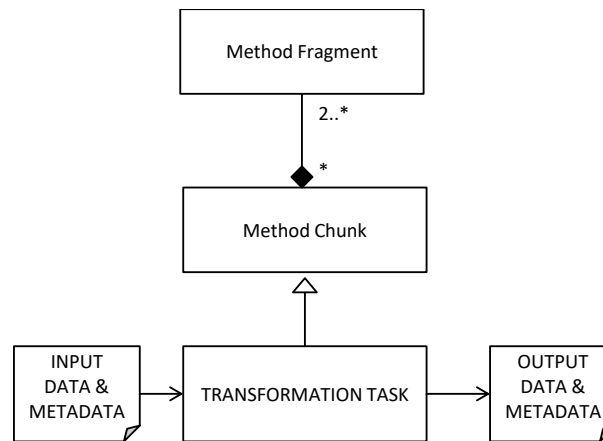


Fig. 14 Method fragments in DICE

Method chunks are the reusable entities that comprise method fragments. At minimum, a method chunk consists of two method fragments, one product and one process fragment, see 5.2 for details. Examples of method chunks in DICE are transformation task patterns which can be configured for their execution. They come with a set of setting options (parameters) and algorithms for data transformation. When executed (in context of an executed data preparation workflow), all required data manipulations are conducted in a standardised manner.

2.6 DIBA and DICE

This thesis must be understood as a continuation and development of the Data Integration in Business Analytics method (DIBA) (Grossmann 2009). DIBA is a domain agnostic BA method for data integration and cleansing. DIBA is heavily based on concepts of workflow management, statistical master data management and data mining. It comes with a basic metamodel for representing metadata to ensure traceability of data integration steps. The main purpose of DIBA is to visually support creation and documentation of the data integration and cleansing processes and to relieve data engineers from the burden of manual documentation of the undertaken DM steps in the phases of data preparation.

In their work, (Papageorgiou et al. 2001) opt for a KDD process accompanied by definition and documentation of metadata supporting the production of traceable BA results. They propose algorithms for the calculation of the metadata based on typical atomic transformation tasks, such as selection, merge and groupby. (Grossmann 2009) seizes these concepts in DIBA and proposes to integrate such algorithms into data transformation processes with the

Methodological and Conceptional Foundations

aim of automatically delivering not only the transformed datasets but also quality indicators and other metadata.

In 2015 a first prototypical implementation of DIBA was accomplished based on the metamodeling platform ADOxx (www.adoxx.org) and the statistical platform R (www.r-project.org). In the course of this, DIBA was renamed DICE. The acronym stands for Data Integration and Cleansing Environment. For more detailed information see the work of (Grossmann and Moser 2016).

With DICE, data engineers can focus on the production of the required datasets. Data preparation services take care of production and interpretation of the statistical metadata. The statistical metadata guide the production process by delivering quality KPIs to support decision making regarding the data preparation processes. In the course of this, each (to be) performed transformation task is evaluated in terms of its quality impact on the transformed dataset(s) and delivers the required metadata and quality KPIs. Based on these KPIs, the data engineer can decide whether a taken transformation task will finally be performed or has to be withdrawn. Thus, early in the data preparation process, the data engineer recognises if the current status of the workflow fulfils the requirements or if additional datasets have to be taken into account. In their work, (Grossmann and Moser 2016) present a prototypical implementation of DICE, substantiating the feasibility of DICE. In the work of (Eltinge, Biemer and Holmberg 2013), a quality approach for the improvement of the entire statistical production process is discussed. The key aspects of this approach are recognised by DICE, making possible the evaluation of the performance of the entire data integration and clearance workflow itself. DICE is the centrepiece of this thesis. That is why in section 4 a detailed introduction to DICE is provided.

2.7 Tying it all together

Recalling the research objectives, in order to set up a sound basis for EA analytics, this thesis seeks to analyse typical EA data and define a method for data cleansing and integration in the fields of EA.

As EA content comes in manifold shapes and styles (see section 7.1), there will not be a one-size-fits-all solution to resolving these research objectives. The thesis builds on DICE, which can be interpreted as a method that is built on concepts of situational method

Methodological and Conceptual Foundations

engineering. DICE combines techniques from the fields of workflow management, metadata management and data mining. The main method chunks facilitated by DICE are *transformation task patterns*, which can be composed into executable data preparation workflows. Transformation task patterns comprise the same elements as modelling methods. From a conceptual point of view, they represent method function blocks consisting of the same basic components as entire methods: modelling language, modelling procedures, mechanisms and algorithms. The modelling procedure involves the configuration of single transformation tasks and the composition of these tasks into executable workflows. Mechanisms and algorithms are required for performing the required data and metadata manipulations. The modelling language is used for modelling the transformation tasks and for depicting the metadata (via dedicated language constructs or by referencing data sources holding the data).

A typical initial situation where DICE might be applied, is when information on the enterprise architecture is required for driving management decisions. Questions to be answered could be as simple as the following: How many applications does the organisation run? Which of our applications hold critical customer data? What applications are operated on outdated technologies? Having a central EA repository in place, which contains all the required building blocks and dependencies, these questions are easily answered. However, it is not very likely that the EA repository, if in place at all, will contain all relevant concepts and relationships for all possible cases. The reality of organisations shows that EA content is often scattered throughout the organisation. EA content such as models is often created ad hoc and not maintained in an adequate manner.

Thus, for this thesis DICE is positioned at the intersection of EA data acquisition and BA data preparation phases. In this sense two spheres have to be considered, namely:

- the domain specific sphere of EAM, which is the sphere where DICE is applied and
- the data mining (DM) sphere, which comprises the data integration and cleansing functions.

This structure allows flexible application of DM methods (DM sphere) to any application domain. At the core of this thesis, DICE is applied to enterprise architectures (EA sphere).

Fig. 15 offers an overview. Both spheres comprise the three main building blocks of modelling methods: modelling language, modelling procedure and mechanisms & algorithms.

Methodological and Conceptional Foundations

The EA sphere represents the application domain providing the datasets for data mining endeavours, hoping to learn about the EA, to uncover previously unseen knowledge or to simply provide an adequate dataset for typical EA analysis. For these purposes the EA sphere delivers EA contents (in the form of formal models but also any other structured or semi-structured EA-related information) describing the architecture of an organisation. This content typically holds information on main EA building blocks, such as the business capabilities, business processes, roles and actors, organisational units and locations, the applications, and their software components as well as dependencies between these building blocks. In other words, the EA content represents the datasets to be cleansed, integrated and analysed in the BA sphere.

As an example, due to compliance reasons, an organisation needs to understand which of their applications hold customer data. EA models based on viewpoints such as Archimate's "Application Behaviour Viewpoint", see (The Open Group 2016) are a valuable source of information for extraction of the required data. Additional information might be found in data catalogues, software architecture models, etc. To analyse the existing EA descriptions, which are typically spread over various models and often described in different modelling languages, data integration and data cleansing tasks have to be performed.

Fig. 15 depicts the interplay between the EA sphere and the DM sphere needed to facilitate this kind of analysis. Enterprise architecture modelling languages (EAML) have already been introduced in section 2.4.1.3. In the DM sphere, a modelling method is required to design the data preparation processes and to hold the metadata generated when conducting the DM mechanisms and algorithms.

Methodological and Conceptional Foundations

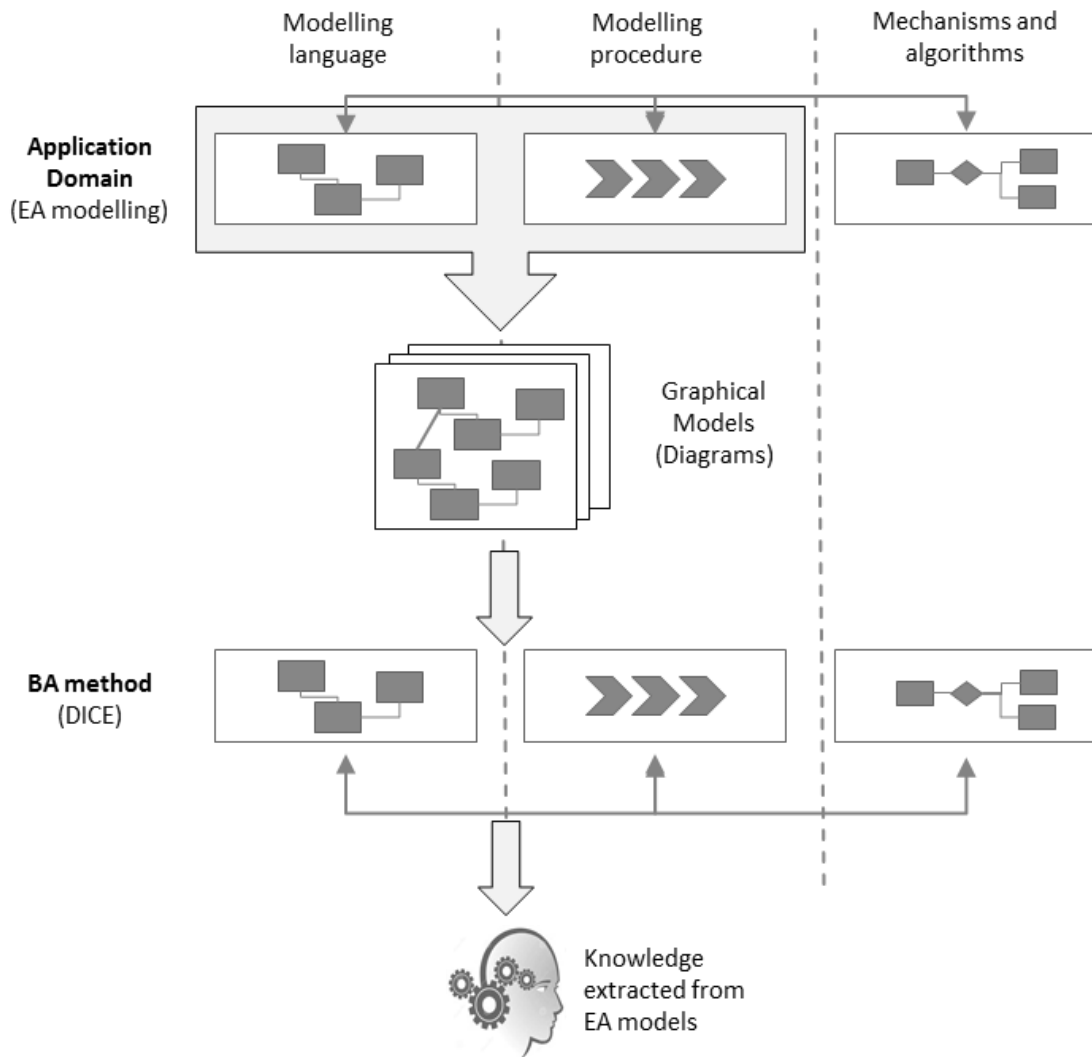


Fig. 15 Interaction of BA and EA

As with the modelling language, the modelling procedure must be considered from both the EA sphere and the DM sphere. The EA sphere covers the procedures for creating the EA models. The DM sphere requires a modelling procedure for creating the DICE processes. These processes are framed by the phases and tasks stipulated in CRISP-DM, the DM reference framework. As explained earlier, in order to be fully effective, the DICE modelling procedure has to be embedded in a superordinated KDD process such as CRISP-DM (Chapman et al. 2000).

The dyadic structure of DM and EA spheres also applies to the “mechanisms and algorithms”, the third of the main elements of a modelling method. The EA specific mechanisms focus on bringing the EA models under control (e.g. versioning mechanisms, release workflows etc.) and the algorithms which are required to run EA analysis, such as EA

Methodological and Conceptional Foundations

simulation, dependency analysis, view generation and what-if evaluations. These mechanisms are positioned to infer new knowledge from existing EA models.

Mechanisms and algorithms of the DM sphere are required for executing the DICE workflows and for performing the data transformation and cleansing tasks. Two types of algorithms are required (Grossmann and Moser 2016). The first type executes the DICE workflow itself. It processes the transformation tasks in the intended sequence. The second type of algorithm, the transformation algorithm, constitutes the processing logic of individual transformation tasks. Simultaneously these transformation tasks perform the transformation on the datasets (the EA models) and alter the related metadata.

All of the core elements of the DM sphere are organised within a situational method. As sketched in section 2.5 (details will be discussed in section 5.2), the set of transformation tasks is organised within the DICE method base. The transformation tasks are configurable and serve as the method chunks of DICE. Instructions concerning appropriate usage and configuration of the transformation tasks and on assembling them into a DICE process have to be covered in the DM modelling procedures.

2.8 Motivating Example

Enterprise architecture is much about communication. Thus, many of the typical EA deliverables are in the form of diagrams, matrices that allow better communication of envisioned target architectures, architectural gaps to be closed and impact on architectural changes.

Relevant data often reside in different formats and different data sources. Even where EA repositories are in place, it is unlikely that those hold all information needed to create the required models for subsequent analysis. Typically, operational data such as customer transactions and process instance data are, if ever taken into consideration, not part of these repositories. In the course of creating decision papers, data quality and data provenance aspects are consequently not considered, which makes the results assailable.

In the course of this thesis a guiding example is introduced. It is based on a scenario where enterprise architects collect and integrate data from internal and external sources of the organisation with the aim of providing a sound EA dataset for subsequent decision making.

Methodological and Conceptional Foundations

In the guiding example, data on business processes, applications and technologies reside in different formats and stem from arbitrary internal and external sources. The example is used to explain the developed mechanisms for data integration and cleansing. It can be seen as a typical application of the method. DICE does not strive to implement a sound EA repository. Rather, it strives for case-specific support for EA-related endeavours based on the existing EA information, which typically resides in manifold data sources.

3 Related Work

The following section is dedicated to the presentation of related work and discusses it in the context of this thesis. Before discussing related work, the literature research approach and its targets will be discussed.

3.1 Literature Research Approach

The subsequent literature research follows the guidelines of (Vom Brocke et al. 2009) to guarantee rigor in the literature search process. The applied research process stems from (Svejvig and Andersen 2015) and is an adaptation of the process initially proposed by (Baker 2000). It was also used in a similar setting creating a literature review of EA analysis/evaluation techniques by (Andersen and Carugati 2014).

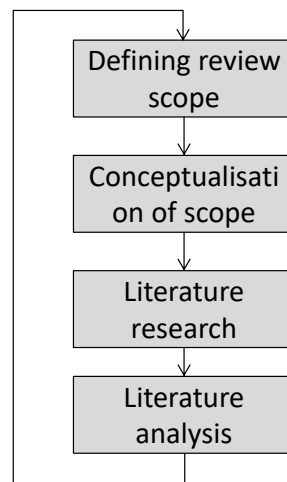


Fig. 16 Main steps of the literature research (Andersen and Carugati 2014)

3.1.1 Defining Review Scope

In the initial step the research scope was defined by stating the main research questions to be answered by the literature research:

- What are the current practices for EA analysis and evaluation and do they pose quality requirements on their underlying EA data? Are these requirements formally defined?
- Is there research that combines EA practices with methods from the fields of data mining and related research fields such as business analytics to support EA-based decision making?

Related Work

- Are data mining techniques used to improve EA data and vice-versa?

The structured literature review was intended to be comprehensive with the aim of covering all relevant research literature within the defined scope. This holds true especially for research literature in the context of EA analysis and evaluation and all relevant literature combining techniques from EA and data mining.

3.1.2 Conceptualisation of Topic

Following Bakers suggestion, to consult *“those sources most likely to contain a summary or overview of the key issues relevant to a subject”* (Baker 2000), the literature review was based on existing literature reviews such as the work of (Simon, Fischbach and Schoder 2013) and (Mykhashchuk et al. 2011) providing broad overviews of existing research in the fields of EA and overviews of EA analysis/evaluation approaches conducted by (Buckl, Matthes and Schweda 2009) and (Andersen and Carugati 2014). From these sources the relevant search terms were derived.

Documents containing one of the following keyword combinations were considered relevant:

$$\{ \text{“enterprise architecture”} \} \times \\ \{ \text{“analysis”} \mid \text{“integration”} \mid \text{“quality”} \mid \text{“data mining”} \mid \text{“KDD”} \mid \text{“business analytics”} \mid \\ \text{“business intelligence”} \}$$

The reasons for deciding on these keywords are briefly discussed below:

- The key word {“enterprise architecture”} was chosen as the core key word. Additionally, in the initial setting of the literature review, key words, such as “business architecture”, “enterprise modelling” and “IT architecture” were used, as EAM is a broad discipline and relevant methods and concepts might not necessarily be presented under the term “enterprise architecture”. However, it turned out that these key words were not effective, as many of the search hits were irrelevant to the topic or already identified by the search containing “enterprise architecture”.
- The keywords {“analysis” | “evaluation”} were used in order to identify scientific literature discussing EA analysis/evaluation techniques. Whereas “evaluation” commonly refers to a systematic determination of a subject’s worth, significance or

Related Work

condition, “analysis” is the process of separating a complex subject into its component parts to gain a better understanding of it (Merriam-Webster 2007). In research both terms are often used synonymously, thus, both keywords were considered. Matching publications are of interest for two reasons: firstly, to check whether analysis/evaluation techniques from the fields of DM have been applied to the EA content and secondly, to check how and if these approaches consider and handle insufficient EA data quality.

- The keyword {“integration”} was used to discover which EA publications consider integration of EA content (e.g. integration of architecture models). The majority of the identified research dealt with integration of EA building blocks itself (e.g. integration of software systems) but not with integration of EA content/descriptions. This is why the initial set of keywords was extended with the keyword combination “enterprise model integration”, the term commonly used for integrating models of different modelling languages (Kühn et al. 2003). Utilizing this keyword combination a plethora of approaches and methods could be identified.
- The keyword {“quality”} aimed at the detection of research focusing on quality aspects of EA contents of any type in general and specifically on EA models. Again, a lot of research dealing with the quality of EA itself (and not with EA descriptions) had to be sorted out.
- Finally, the keywords {“business analytics” | “data mining”| “KDD” } were used to identify research that connects methods from the research fields of EA and DM. As will be discussed in the literature review, only a few publications were identified. Most of the publications matching these keywords dealt with defining enterprise architectures to support business analytics and data mining implementation but not with combining the practices for creating better/new EA content.

The present literature review does not focus solely on top journals and conferences but also takes industry EA publications (such as articles from the journal of EA) into account, keeping in mind that not all publications are of equal rigor (Ngai and Wat 2002). This decision was taken because non-academic EA publications have been judged to be of significant influence in existing EA literature reviews, see (Mykhashchuk et al. 2011), (Simon, Fischbach and

Related Work

Schoder 2013) and (Tamm et al. 2011). Thus, by consideration of peer reviewed IS journals only, highly relevant practice-oriented literature might have been excluded.

As primary search engine, Google Scholar (<https://scholar.google.at>) was used. As a second source, u:search, the search engine of the university of Vienna was used. Both of these search engines serve as meta search engines and base their searches on search engines and digital libraries such as the ACM Portal (<http://portal.acm.org>) and the IEEE XPlore Digital Library (<http://ieeexplore.ieee.org>).

Per search, the first hundred search results were considered. Relevant research results were listed. From these articles, backward searches (reviewing the references of the identified articles) and forward searches (reviewing sources that cited the initial articles) were conducted. For forward searches, the Google Scholar feature “cited by” was used, insofar as a research paper could be found via Google scholar. Papers considered relevant were listed and the procedure was repeated for these papers.

For all papers, at least title and abstract were read in order to judge their relevance with reference to their contribution in the fields of analysing/evaluating EAs or parts thereof. Articles considered relevant were fully examined.

The literature review was performed throughout the years 2015, 2016 and 2017 and considered EA articles from the early beginnings of EA, from the mid-1980s with the first recognised EA frameworks “Zachman” and “PRISM”, see e.g. (Kotusev 2016) to March, 2017.

3.1.3 Literature Research & Analysis

According to (Närman et al. 2008), the mechanisms provided by common enterprise modelling frameworks (such as TOGAF, DODAF and MODAF) for analysing data are barely sufficient. Thus, inferring new knowledge from existing EA models can be considered a major challenge. In their work, Närman et al. present an EA analysis procedure consisting of the following steps: (1) define scenario, (2) determine properties of interest, (3) modelling scenarios using a metamodel, (4) analyse the scenario properties and (5) make a decision. Thus, before analysing the EA, the EA (or subsets of EAs) have to be modelled using a metamodel. This requirement is common to almost all EA analysis approaches, as will be elaborated in more detail in the subsequent sections.

Related Work

In their work, (Buckl, Matthes and Schweda 2009) categorise EA analysis approaches using a classification schema. The schema is based on the dimensions: *body of analysis* (structure, behaviour statistics and dynamic behaviour), *time reference* (ex-post vs. ex-ante), *analysis technique* (expert-based, rule-based etc.), *analysis concern* (functional vs. non-functional) and *self-referentiality* (none, single-level and multi-level). Presenting their AHP-based (Analytical Hierarchy Process based) approach, Razavi et al. (Davoudi, Aliee and Badie 2011) add an additional criteria, namely *source of analysis* (EA models vs. EA content).

In the following section the current and prominent EA analysis approaches are briefly introduced and examined regarding requirements they pose to the underlying model base.

Numerous approaches to EA analysis are based on ***probabilistic relational models (PRM)*** and formalisms based on Bayesian network statistics. These approaches make possible the analysis of EA (scenarios) in the context of a broad range of system qualities, such as data quality (Närman et al. 2008) and (Närman et al. 2011), modifiability (Lagerström, Johnson and Höök 2010), maintainability (Ekstedt et al. 2009), reliability (Närman et al. 2014), security risks in general (Sommestad, Ekstedt and Johnson 2010) and aspects of cyber security (Sommestad, Ekstedt and Holm 2013). A hybrid approach taking several of these system qualities into account is presented in (Närman et al. 2010). The PRM formalism in these analysis techniques makes it possible to deal with uncertainty when computing the values of qualities. Similar to the above analysis techniques, (Franke et al. 2009) analyse enterprise architectures by applying Bayesian belief networks to fault tree analysis evaluating reliability and reusability qualities. All of these approaches require a concise model base as an input. Moreover, not only dependencies between the various EA elements have to be defined but also direct relations between quality attributes must be in place in order to analyse their causal dependencies. This requirement is clearly specified in the work of (Buckl et al. 2011), which proposes extensions to MOF (Meta Object Facility) in the form of a meta-language to support PRM analysis.

Johnson et al. (Johnson, Nordström and Lagerström 2007) recommend **Architecture Theory Diagrams (ATDs)** and the application of Dempster–Shafer theory, a general framework for reasoning with uncertainty.

Another type of EA analysis can be grouped under the umbrella decision aid/decision support systems such as the **multi-criteria decision making method (MCDM)** *Analytic*

Related Work

Hierarchy Process (AHP), where decision makers split their decision problem into a hierarchy of separately appraisable sub-criteria. In the context of EA, these sub-criteria are represented by EA quality attributes such as maintainability and interoperability, which itself might be further composed. Applications of AHP in the context of EA are discussed in (Davoudi, Aliee and Badie 2011) and (Davoudi and Sheikhvand 2012). The architecture (scenarios) to be assessed do not necessarily need to be documented based on strictly-typed models. That is why (Davoudi and Sheikhvand 2012) categorise their AHP-based approach as “EA content”, as opposed to “EA models” in the category “Source of analysis”. The multi-property utility evaluation is another representative of a MCDM. It defines a utility function over a set of qualities to evaluate a particular scenario. Again, independent qualities such as maintainability and interoperability are considered by describing their utility in the form of a utility function. As with the AHP-based approaches, these analysis techniques do not necessarily need a fine-grained model-base. In contrast, the utility-based approach proposed by (Österlind et al. 2013) can be applied to meta-object facility compliant EA models only.

(de Boer et al. 2005) use an XML schema representing an enterprise architecture meta-model, which makes description, analysis and simulation of EA models possible. XML elements are used to define structural information of the EA. Utilising XML and AML (ASCII Markup Language) parsing tools, they execute static analysis algorithms on the EA models. Dynamic behaviour is modelled and analysed applying state machine semantics. Of course, this approach requires a sound and concise model base structured in XML (and AML files). The EA analysis approach of (Iacob and Jonkers 2006) presents a similar approach applying **quantitative analysis techniques** to Archimate models. For this purpose, they added quantified attributes, such as workload parameters, cost and performance characteristics to the concepts and relations of the Archimate modelling language. The approach supports computational EA analysis by top-down propagation (of workload characteristics) and bottom-up propagation (of costs and performance characteristics). Similar to this approach, (Florez, Sánchez and Villalobos 2014) propose the concept of SAMBA (Specialized or extended ArchiMate Metamodel for Business Analysis), which is an extension to the Archimate Metamodel for analysis purposes.

Ontology-based analysis techniques are another important category. They support EA analysis through application of computational inference and querying mechanism. Manifold research has been conducted in these fields. Examples from the broad set of articles are

Related Work

(Caetano 2016), (Hinkelmann, Maise and Thönssen 2013), (Antunes et al. 2013) and (Sunkle, Kulkarni and Roychoudhury 2013).

As another important category of EA analysis approaches, **KPI-based analysis techniques** must be considered. Publications in these fields include the works of (Vasconcelos, Sousa and Tribolet 2007), (Brückmann et al. 2009), (Monahov et al. 2012), (Singh and van Sinderen 2015), (Vasconcelos, Sousa and Tribolet 2015) and (Addicks and Appelrath 2010). All of them have in common the use of EA models instantiated from an EA metamodel as a basis. The KPIs are calculated solely from model-intrinsic data.

EA analysis based on **network analysis** is discussed in (Aier and Winter 2009) where a graph-based clustering algorithm is presented, which makes possible the identification of domains within organizational and application architectures. In his master thesis, (Schoonjans 2016) applies typical metrics from the fields of graph theory to EA models, which are transferred into directed graphs (such as in social networks) for further analysis. A literature review of EA network analysis with a broader definition of EA as compared to this thesis can be found in (Santana, Fischbach and Moura 2016).

Finally, there is research with emphasises on the dynamic behaviour of enterprises. Besides an abundance of work conducted in the fields of business process management, see e.g. (Van Der Aalst, Ter Hofstede and Weske 2003) for an overview, there is little research on **simulation techniques** dedicated to EA. One noteworthy work on this topic has been done by (Glazner 2011), who presents a hybrid simulation technique based on system dynamics, agent-based and discrete event simulation algorithms. Also, in the work of (Manzur et al. 2015), a model-based approach is introduced for simulation of EA models with the aim of executing and assessing EA scenarios. For this purpose, the Archimate modelling language has been enriched with dynamic properties (e.g. statistical distributions on usage behaviour of clients) to allow for simulation.

Revisiting these “prominent approaches” with respect to their underlying information base has shown that none of the approaches tackle data quality problems prior to the analysis. Almost all of the EA analysis approaches require a model-base instantiated from an EA metamodel that is in conformity with the requirements posed by the EA analysis algorithms. The ontology-based approaches do not require a metamodel but have to adhere to ontology languages based on either first-order logic or on description logic. Prominent examples of

Related Work

ontology languages are: the Web Ontology Language (OWL), the Resource Description Framework (RDF) and the RDF Schema (RDFS). For an overview of ontology languages see (Staab and Studer 2013). Thus, the only true exception can be seen in the multi-criteria decision making methods (e.g. AHP) where scenario descriptions are required, but decision making does not require formally described EA models as a base for algorithms. Another important finding is that the required quantitative or operational data concerning the EA elements have to be imputed manually by the EA experts. Where to take the required data from is, (if at all) only vaguely discussed in the articles.

Table 2 summarises the results.

Table 2 Overview and assessment of EA analysis approaches

Analysis Type	References	Model-based approach	Application of BA techniques
Probabilistic relational models	Excerpt: (Närman et al. 2008), (Närman et al. 2011), (Lagerström, Johnson and Höök 2010), (Ekstedt et al. 2009), (Närman et al. 2014), (Sommestad, Ekstedt and Johnson 2010), (Sommestad, Ekstedt and Holm 2013). (Närman et al. 2010)	Yes	No
Architecture Theory Diagrams (ATDs) / Dempster-Shafer theory	(Johnson, Nordström and Lagerström 2007)	Yes	No
Multi-criteria decision making methods (MCDM)	(Davoudi, Aliee and Badie 2011), (Davoudi and Sheikhvand 2012), (Österlind et al. 2013)	Yes	No
Quantitative analysis	(de Boer et al. 2005), (Iacob and Jonkers 2006), (Florez, Sánchez and	Yes	No

Related Work

techniques	Villalobos 2014)		
Ontology-based analysis techniques	Excerpt: (Caetano 2016), (Hinkelmann, Maise and Thönssen 2013), (Antunes et al. 2013) and (Sunkle, Kulkarni and Roychoudhury 2013)	Yes	No
KPI-based analysis techniques	(Vasconcelos, Sousa and Tribolet 2007), (Brückmann et al. 2009), (Monahov et al. 2012), (Singh and van Sinderen 2015), (Vasconcelos, Sousa and Tribolet 2015), (Addicks and Appelrath 2010)	Yes	No
Network analysis techniques	(Aier and Winter 2009), (Schoonjans 2016), (Santana, Fischbach and Moura 2016)		
Simulation-based analysis techniques	Glatzner (Glazner 2011), (Manzur et al. 2015)	Yes	No

Turning to the second core area of this literature research with a focus on integration aspects of BA and EAM, it became apparent that there is very little literature available. Most of the identified literature is concerned with applying methods of enterprise architecture management to establish business analytics and business intelligence capabilities within an organisation. The joint use of EA and BA to improve management decisions is only addressed in four articles.

As early as 2005, (Neaga and Harding 2005) proposed in their work *“An Enterprise Modelling and Integration Framework based on Knowledge Discovery and Data Mining”* to complement enterprise modelling methods with methods from the fields of KDD and DM in order to improve decision making based on enterprise models. The authors emphasize the

Related Work

communalities of enterprise modelling and DM, as both techniques “*are building models of the whole or parts of the enterprise [...] mining models are directed to logically fit or overlap with enterprise models, except that they are obtained by knowledge discovery*”. However, although recognising these communalities, their work is limited to extending enterprise modelling methods with knowledge and mining views to support modelling and planning of integrated KD&DM systems.

(Barone et al. 2010) recognise the benefits of integrating enterprise modelling and business analytics and present an approach to integrate business intelligence mechanisms with their *Business Intelligence Model (BIM)*. The idea is that business people define their knowledge requirements in business terms based on goal modelling techniques (inspired by strategy maps and balanced scorecards), and these specifications are mapped with a minimum effort onto the technical BI implementation schemas. Although a broader integration with EA approaches is not explicitly discussed, the proposed BIM can be seen as the strategy model of an EA. Thus, it serves as the docking point for additional EA views and viewpoints. Concrete algorithms and mechanisms on how to actually integrate the enterprise models with BI tools, how to use the BIM to “automatically” map business structures defined in the BIM to the operational data are not discussed and are left open for future research.

(Stravinskienė and Gudas 2011) conducted an analysis of enterprise modelling methods in respect to their suitability for supporting DM endeavours. Their work on enterprise knowledge modelling and data mining integration strives to define principles for the integration of knowledge components (represented in the form of enterprise models) into data mining processes by using the EA models as input for “*more rational data mining queries*” (Stravinskienė and Gudas 2011). The authors propose a mapping of components of EA concepts and data mining process steps in order to streamline the mining process and the configuration of the required mining algorithms. The approach is illustrated by using the *enterprise meta-model (EMM)*, see (Gudas, Lopata and Skersys 2005) for details and a sketched data mining process. The approach is intended to be agnostic to the given enterprise modelling method. However, the presented mapping remains high-level, and details on how to actually implement this mapping are only discussed superficially.

(Fill and Johannsen 2016) present in their knowledge-based approach, required mechanisms to integrate enterprise modelling and data analysis. Emphasis lies on interaction and exchange of data between enterprise models and methods for analysing big data. In their prototypical

Related Work

implementation, they demonstrate how prepared data are loaded into a modelling tool supporting the RUPERT (Johannsen and Fill 2014) modelling method, from where the data are processed to graphically support decision making by enriching the models defined in RUPERT. The data preparation process and data quality issues are not covered by their approach.

Finally, (Veneberg et al. 2014) propose their method “Enterprise Architecture Intelligence Lifecycle (EAIL)” for combining enterprise architecture descriptions and operational data. They suggest either enriching EA models with BI data to support model-based enterprise architecture analysis or integrating the EA descriptions into BI tools for better structuring of the operational data and to support their retrievability and interpretability. In the second case, EA structures would serve as the metadata for classifying the operational data and for creating interrelations among themselves. The proposed lifecycle consists of the phases: (1) explore, (2) match, (3) enrich, (4) visualize, (5) decide & change and (6) evaluate. In their conclusion, the authors see the required technical model and data transformations as a critical step, which has so far not been addressed. Furthermore, the need to investigate the applicability of existing EA and DM analysis mechanisms is adverted.

In respect to enterprise model integration, a great many approaches could be identified. Solutions comprise hard-coded model transformation mechanisms, framework-specific annotations (via metadata), multi-view methods, approaches based on controlled vocabulary and ontologies. For an overview of model transformation tools, see (Gomes, Barroca and Amaral 2014), (Lúcio et al. 2014) and (Czarnecki and Helsen 2003). Fig. 9 depicts the general ideal process of model transformation. What all of these approaches have in common is that input models instantiated from a source metamodel are transformed into target models that conform to a target metamodel. Transformation rules are required to specify the mapping rules. Fig. 17 shows an overview of typical concepts involved in model transformations.

Related Work



Fig. 17 Model transformation terminology, adapted from (Syriani, Grayand Vangheluwe 2013)

(Falleri et al. 2008) propose an approach that automatically detects mappings between two metamodels and uses them to generate an alignment between those metamodels instead of developing ad hoc model transformations. Their approach is based on concepts from the fields of schema matching and ontology alignment. Domain-specific approaches with a focus on transformation of enterprise models are discussed, e.g. in (Kühn et al. 2003) and (Vernadat 1996). Worth particular mention is the work of (Zimmermann et al. 2013), who present with their *service-oriented reference enterprise architecture* an approach dedicated to the integration of enterprise architecture models. The approach is based on correlation analysis and offers a systematic integration process. (Moser, Fürstenau and Junginger 2010) present an approach to the integration of EA models with business process models, which again requires

Related Work

a bidirectional mapping. The focus of their work is on how to organise the model exchange. They place a strong focus on organisation-specific objectives and conditions but do not consider BI mechanisms for integration. Worth particular mention is the research of (Roth 2014). In his dissertation on „Federated Enterprise Architecture Model Management“ Roth presents an approach for collaborative model integration. The approach deals with the continuous integration of object-oriented models held in different repositories. Strong emphasis is put on techniques for conflict resolution (model and metamodel inconsistencies). Roth introduces a software solution to organize the conflict management. Conflicts are passed to users (the particular owners of the objects) for semi-automated resolution. The approach assumes a unidirectional synchronization between the data held in the EA repository and all other relevant information sources. Its main purpose is to support the federated EA model management and to build up a concise EA model base by integrating building blocks from the different sources. The EA repository acts as the sink source integrating EA-relevant data in conforming to a defined target metamodel. In DICE the concept is different. Datasets from arbitrary sources are adapted, integrated and cleansed as needed without consideration of structural “limitations” in the first place. DICE for EAA is a specific application of DICE in context of EAA. It strives for creating situational datasets, as a basis for subsequent BI-driven analysis. Rather than focusing on the maintenance of a concisely populated EA repository, datasets are created for specific knowledge needs at hand, which cannot be foreseen in advance and answered by standard mechanisms of EA repositories, or rather by EA descriptions/models adhering to a predefined metamodel. To this end DICE for EAA strives for equipping EA data with operative business data such as customer data, product data and any transactional business data.

Applying data mining techniques, e.g. supporting the model transformations with unsupervised transformation techniques, have not been identified by the conducted literature review. The only exception can be seen in the work of (Gill and Qureshi 2015) who present in their approach to “*adaptive enterprise architecture modelling*” techniques for metamodel integration based on DM techniques such as applying similarity analysis to concepts within different enterprise (meta) models with the aim of model integration.

3.1.4 Summary

Compared to the extensive research available on enterprise architecture management, the subfield of EA analysis can be seen as the orphan child of EA. Nearly all of the identified EA analysis approaches require formally created EA models as a basis. It is well recognised that the quality of EA models is of immanent importance for any EA endeavour. Existing approaches that strive to improve the data quality mainly focus on improving the data quality of models held in architecture repositories, by either optimising the processes for manual data acquisition, clearly costing higher data maintenance and governance efforts, or on automated integration-features that integrate EA data from external sources such as CMDBs. From the observed research literature, only (Addicks and Appelrath 2010) integrated data quality and metadata aspects into their approach for indicator-based evaluation of application components in the context of EA. However, the approach is only roughly sketched and focuses only on data freshness and population coverage aspects. In addition, (Iacob and Jonkers 2006) consider model integration and model normalisation (transforming the models to comply with the required structure for analysis) as an important step but do not explain the required transformation measures.

Recently, approaches to integrate EA and BI practices have been put forward. Under designations such as “Enterprise Architecture Intelligence” and “Enterprise Architecture Analytics”, initiatives arise striving to combine the management domains of “Enterprise Architecture Management” and “Business Analytics” respectively in order to combine EA models with operational data. Although the benefits of integrating EA with operational data have been identified, research in these fields is still in its infancy.

DICE as a method for data integration and cleansing contributes to these initiatives by providing the means to integrate all relevant EA models and descriptions, and at the same time, provides metadata such as quality indicators on the resulting EA datasets.

4 DICE – The Centrepiece of the EAA

This section is strongly based on the work of Grossmann and Moser (Grossmann and Moser 2016), who introduced DICE as a method for data integration and cleansing in the context of business analytics scenarios. DICE is, as a continuation of Grossmann's work on DIBA (Grossmann 2009), a data integration method for the support of business analytics endeavours.

4.1 Suggesting a Method for Data Integration and Cleansing

Most DM techniques demand input datasets in a horizontal layout, i.e. a table in which each row represents an observable unit and each column stands for a variable (Ordonez and Chen 2012), (Rahm and Do 2000), (Klösgen 2002). Examples are DM techniques, such as association rules, clustering, classification, regression analysis and principal component analysis (Han and Kamber 2000), (Ordonez 2010).

Knowledge about the data quality and data provenance is a key requirement when applying the data for decision making in business activities. Thus, along with the actual data (e.g. provided in the form of management reports), explicit descriptions that provide insight into data quality, data sources and data structures are required. Such data descriptions are commonly referred to as metadata. Metadata include information for producing, understanding, querying and retrieving statistical work products. DICE adopts the concept of metadata management by leveraging a metadata-model based on concepts stemming from the fields of statistical metadata management (Grossmann 2015). Against this background, DICE considers datasets as **composite analysis objects** (Grossmann and Moser 2016) composed of observable units, variables and properties that represent populations of observable units. Fig. 18 provides an overview of the structure of these data objects and their interplay in UML notation.

DICE – The Centrepiece of the EAA

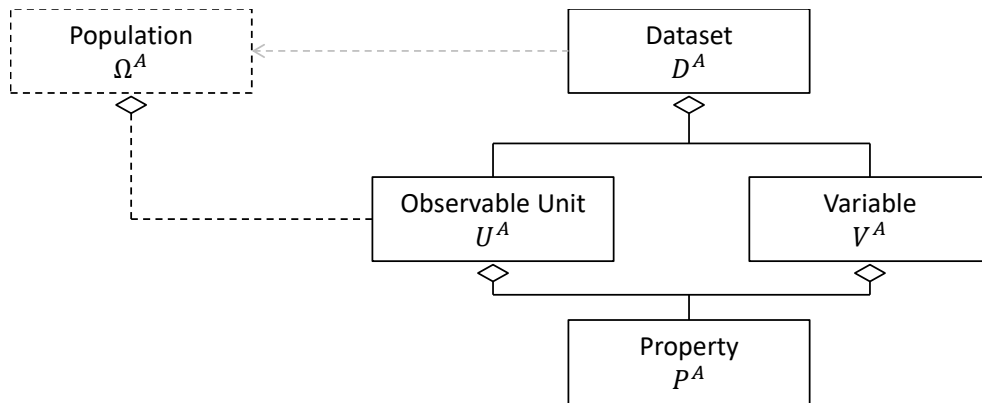


Fig. 18 Composite Data Analysis Object

Observable units U^A are the entities for which empirical information is collected or derived. In a typical enterprise environment, elements, such as products, customers and transactions are perfect examples of observable units. In the context of enterprise architecture management, elements, such as business processes, application components and technology components are typical examples. Ω^A represents the population that is defined by the set of observational units. Typically, a distinction is made between *target* population (aka scope), representing the real-world population and the covered population which is represented within the dataset (aka statistical population or sample population). The dataset D^A holds the observable units and their properties. For each of the observable units, properties P^A are available. These properties are categorised into variables V^A (i.e. attributes). The typical representation of a dataset D^A in business analytics is a table. The rows correspond to the observable units, the columns represent the variables and the cells represent the properties. The properties are observed and described by values assigned to the properties, i.e. the values are stored within the table's cells. The covered population is typically implicitly represented only by the set of observable units, i.e. the rows of the table.

Fig. 19 illustrates an example. It exemplarily depicts an Application Portfolio Catalogue, a typical EA artefact, see (The Open Group 2011). In the example each row holds data of an application that is an observable unit within this dataset. The column “ID” holds the unique identifier of the applications. The “ID” and the subsequent columns “Owner”, [...], “Operating System”, “Operating Costs” and “Production Date” represent the variables. The cells of the table represent the properties which hold the values that distinguish the applications.

DICE – The Centrepiece of the EAA

Observable Unit	00123	FinApp	[...]	Windows Server 2013	123	Population
	01020	CosMos	[...]	Red Hat Enterprise Linux 7.x	230	
	01029	Orinocco	[...]	SUSE Linux 4.2	17	
	[...]	[...]	[...]	[...]	[...]	
Property						

Fig. 19 Example of a composite analysis object – data level

Formally the composite analysis object can be denoted as follows:

$$O^A = [D^A, U^A, V^A, P^A].$$

The superscript A refers to the data level. Note, as the population is only implicitly represented in the dataset, it is not part of the formal definition of the composite analysis object on data level. The population is made up of the set of observable units that are described by a set of properties as follows:

$$U = \{P_u \mid u = 1, 2, \dots \mid u \mid\}.$$

and a set of variables described by a set of properties

$$V = \{P_v \mid v = 1, 2, \dots \mid v \mid\}.$$

A property belongs to exactly one observable unit and to one variable of the dataset:

$$D = \{P_{uv} \mid P_{uv} \in U_u \vee P_{uv} \in U_v\}.$$

From the above definition follows:

$$D = U = V$$

i.e. the dataset is made up of the same properties as the set of observable units and the set of variables.

Of course, for the unambiguous description of a dataset, additional information called metadata is required. Often quoted examples for metadata are: source information, semantic definitions of the observable units and the variables and data quality indicators. In this vein, each of the introduced concepts finds its pendant on the meta-level. Fig. 20 illustrates the basic structure.

DICE – The Centrepiece of the EAA

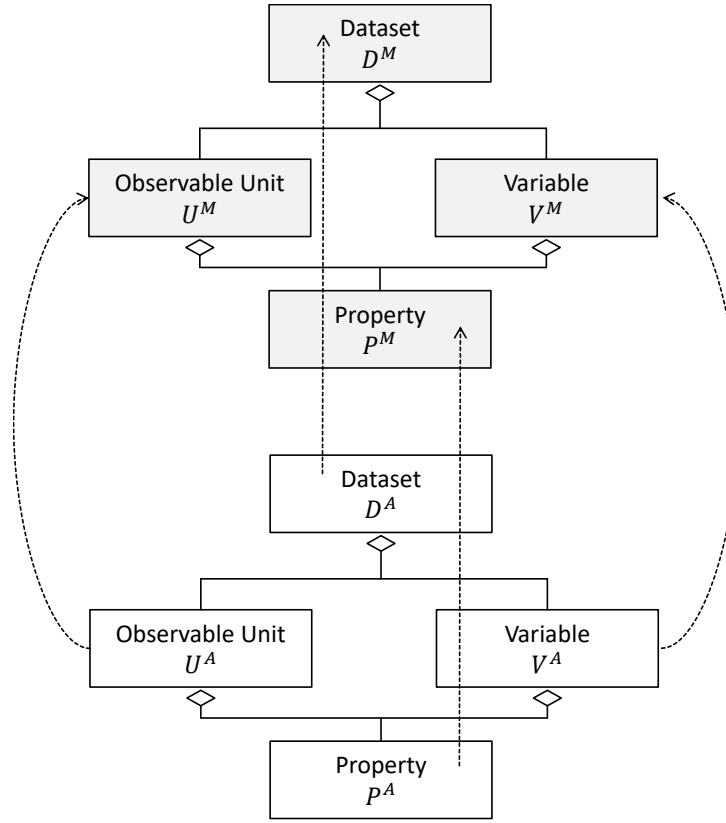


Fig. 20 Data-level concepts and related metadata concepts

Formally the metadata level is represented as follows:

$$O^M = [D^M, U^M, V^M, P^M].$$

The superscript M refers to the meta level. The metadata object U^M holds descriptive information about observable units. Typical examples are: the semantic definition of the observable unit and quality data per observable unit. The metadata D^M holds summary descriptions of the dataset, such as data profiling contents, summative quality metrics over the entire dataset, administrative information, such as source and access information, security issues and data owner etc. Also, information on the population (Ω), such as the unambiguous definition of the covered population, the size of the real-world population and the required coverage are part of the metadata. In this context, time aspects, spatial aspects and any other criteria make possible the specification of the set of contained observable units. The metadata object V^M carries the definitions of the variables, their value domains and measure units. These definitions are important for specifying data quality aspects. P^M provides information about quality aspects regarding single properties. Examples of quality aspects for properties

DICE – The Centrepiece of the EAA

are: missing values, compliance with the required datatype and freshness of the property values etc.

Take the example of the “Application Portfolio Catalogue” depicted in Fig. 19. By solely examining the dataset, one can conclude that the observable units are the application components. However, one cannot reliably preclude that the applications are the observable units. The “operating systems” (contained in the fourth column) could also represent the observable units of the dataset. Therefore, the observable unit has to be declared unambiguously within the meta object U^M . A similar problem arises with the population. Without clear specification, one can only guess whether the given dataset covers the entire organisation, applications of single subsidiaries of the organisation or any other subset of the set of the organisation’s application components. For each of the dataset’s variables, value domains and measure units need to be defined. Take for example the variable “Costs of Operations”. Without information on the currency and the timeframe, the costs related to the actual costs remain insufficiently interpretable. Thus, value domains and measure units have to be clearly defined. Looking at this simple example, it becomes obvious that there are a lot of metadata required to guarantee interpretability and processing of the data.

Fig. 21 shows the previous example data object annotated with some of its metadata.

Variable Name	ID	Name	[...]	Operating System	Cost	Source Information	
Variable Type	[Integer]	[String]	[...]	[String]	[Integer]		
Measure Unit	--	--	--	--	TEUR		
						Quality Measure	
00123							
FinApp							
[...]							
Windows Server 2013							
123							
01020							
CosMos							
[...]							
Red Hat Enterprise Linux 7.x							
230							
01029							
Orinocco							
[...]							
SUSE Linux 4.2							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							
[...]							

Fig. 21 Example a data object and annotated metadata

In DICE, metadata are structured into four perspectives: (1) semantics, logistics, process and quality metadata. Section 2.4.1.3 offers a detailed overview of these perspectives by introducing the DICE metamodel.

DICE – The Centrepiece of the EAA

Representing the data in the form of data objects and associated metadata objects is the core of the DICE method, which supports the manipulation of the data objects in the context of data preparation phases. Typical manipulations for the datasets are tasks for reformatting data, for selecting data, for constructing data (in case of missing values) and for integrating data from different sources.

In accordance with (Grossmann and Moser 2016), these manipulations are represented as transformations of the number of input objects $O^{A,(i)}$, $1 \leq i \leq k$ into a number of output objects $O^{A,(o)}$, $1 \leq o \leq p$:

$$T^A : [D_i^{A,(i)}, U^{A,(i)}, V^{A,(i)}, P^{A,(i)}] \mapsto [D^{A,(o)}, U^{A,(o)}, V^{A,(o)}, P^{A,(o)}]$$

Depending on the type of applied transformation tasks, various elements of the data objects are affected. Take the example of performing a selection transformation on a given input data object. In this case, an output data object is created; the dataset and its meta objects are likewise changed. By performing the selection, the set of observable units is reduced to the observable units matching the selection criteria. This of course has a direct impact on the population of the dataset, while other metadata objects of the data object such as the variables and their value domains remain unchanged.

Data integration is an example of another but more complex transformation. In the case of data integration, two input data objects are integrated into one output data object. The resulting structure of the output object depends on the type of integration applied. (Grossmann 2009) differentiates between two archetypes of integration transformations: *horizontal* and *vertical integration*. In the case of horizontal integration, the variables along with their associated properties of two formerly independent data objects are merged into one data object. The resulting data object will contain the superset of variables and hence the superset of their properties. The number of observable units and the covered population remain unchanged. In contrast, vertical integration describes the integration of two datasets composed of the same variables but containing different or an overlapping set of observable units. In this case, metadata objects do not change in regard to their variables, whereas the population will change. For a better understanding Fig. 22 illustrates the two archetypical integration transformations.

DICE – The Centrepiece of the EAA

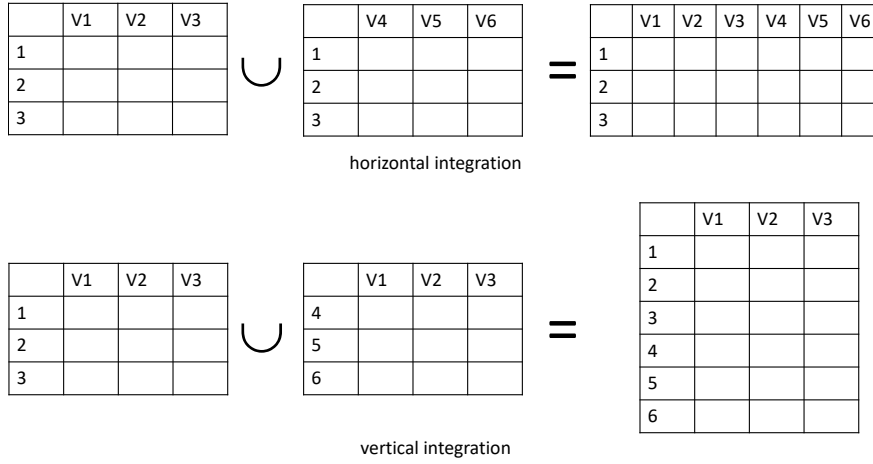


Fig. 22 Horizontal vs. vertical integration of two datasets

In summary, performing transformation tasks requires altering the data on data level and concurrently on metadata level. In any case, metadata has to be considered as an input. In a simple case of transformation, the metadata of the input object are merely extended. This occurs, for example, when calculating summative values from a dataset. However, often metadata from the input data object will be required in order to make a decision about adequate transformation steps. An example of such a case is data editing. Data conventions, represented in the form of edit rules on metadata level, might pose constraints to property values on data level, which have to be considered when performing editing operations. An example of an even more challenging case is when computation on the data level requires input from computation on the metadata level. An example can be seen in data integration on schematically different datasets. In such a case, similarity measures are typically calculated on schema-level (thus, based on metadata), and based on these computations the transformation on data level can be performed. From these examples one can conclude that data and metadata are closely interlinked. DICE recognises this fact and comes with features which make possible the alteration of the data and the metadata objects concurrently. Formally the metadata transformations can be described as follows (Grossmann and Moser 2016):

$$T^M : [D^{M,(i)}, U^{M,(i)}, V^{M,(i)}, P^{M,(i)}] \mapsto [D^{M,(o)}, U^{M,(o)}, V^{M,(o)}, P^{M,(o)}]$$

Fig. 23 illustrates an entire chain of transformations schematically. The chain of transformation is called DICE process and consists of the three transformation tasks: T1, T2 and T3. Each of these steps takes a composite data object as input, transforms data and metadata concurrently and passes on the generated output data object to the next

DICE – The Centrepiece of the EAA

transformation task. At any position within the DICE process, data quality can be assessed and data provenance is transparent.

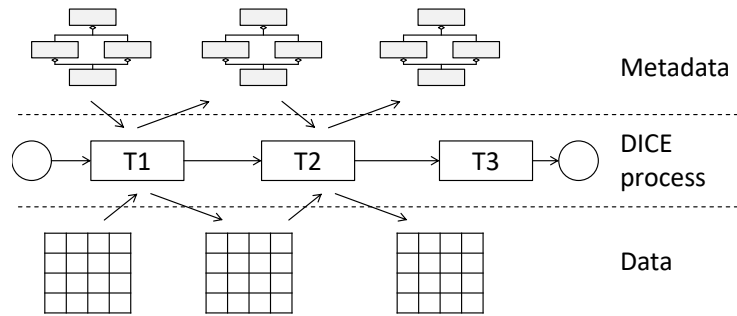


Fig. 23 Simultaneous transformations for data and metadata objects

4.2 Summary

DICE contributes to data preparation endeavours by providing means to alter data and their metadata simultaneously. It considers data not as pure datasets but as composite data objects consisting of the data values and their metadata.

DICE is designed as a domain agnostic method for data preparation that can be specialised for the situational needs of a given BA endeavour. Section 5 introduces its main concepts following the principles of: (1) agile method engineering and (2) situational method engineering.

5 DICE - Method Conceptualization

In their preliminary work on DICE, (Grossmann and Moser 2016) have established the foundational concepts of the method based on a meta-modelling approach. In general, the design and implementation of DICE is framed by the Agile Modelling Method Engineering framework (Karagiannis 2015), which constitutes the following phases: create, design, formalise, develop and deploy/validate. In (Grossmann and Moser 2016), the authors place a strong focus on the overall concepts of DICE but give only little guidance on how to apply DICE in a (situational) project environment. The following section places emphasis on this aspect of DICE.

DICE is strongly oriented towards the principles of situational method engineering (see section 2.5). The method overview provided (see section 2.6) reveals that there is no one-size-fits-all method to support the data preparation phase of a BA endeavour. Applying the principles of situational method engineering as a foundation provides two vital advantages: (1) DICE can be continuously extended with new features in a controlled manner and (2) method users can compose their individual situational methods from DICE.

At the core of situational methods one typically finds the *method base*, which is a repository holding reusable method chunks. Each of these method chunks is instantiated from a meta-model, which is denoted as meta-structure throughout this thesis.³ In simple terms, method chunks are the main building blocks for assembling a situational method. Rather than defining a method completely from scratch, a method engineer composes his/her method from the available method chunks.

It is commonly agreed that methods in general comprise *process* and *product aspects*. This also holds true for method chunks. In DICE, method chunks consistent with the definition of (Henderson-Sellers and Ralyté 2010), consist of one process fragment and one or more product fragments. The method fragments are the atomic elements of the method. Under these terms, DICE (or more generally, any method) can be structured into an assembly of reusable method chunks. Wherein (Ralyté and Rolland 2001):

³ To avoid confusion we intentionally avoid the common term “meta-model” in the context of the situational method, as this term is already used to denote meta-models as one of the core components of a modelling language.

DICE - Method Conceptualization

- a method itself is understood as a method chunk on the top level,
- a method consists of method chunks and
- method chunks are assembled from product and process fragments.

Product fragments denote the product that is created by the method chunk. More broadly, (Henderson-Sellers, Gonzalez-Perez and Ralyté 2008) consider both the input and the output (product) to be specified in the form of a product fragment. In consequence, a process fragment denotes the steps (sometimes also called “guidelines”) required to transform the input products into the output products. In the following, these definitions are applied to DICE. Fig. 24 provides an overview of the main concepts and dependencies. DICE draws its method fragments from various management disciplines:

- from the fields of **process and workflow management**, concepts are derived to design DICE workflows graphically and to execute them ,
- **statistical metadata management** provides concepts for the definition and management of relevant metadata,
- the research field of **data mining** provides countless techniques and algorithms to support data transformations; additionally, reference models are available (such as CRISIP-DM) for performing the data preparation and setting up entire DM initiatives
- and research from **quality management** and statistical quality management in particular provide frameworks for defining adequate quality measures.

The application of DICE is structured in five phases. In the first phase (1) relevant parts of these methods are identified, extracted and taken to the method base. (2) The method parts are structured to fit into the meta-structure of DICE. For this purpose the original method parts are classified into process and product fragments and adapted/enhanced where needed. Process fragments are made up of guidelines and algorithms, whereas the product fragments describe input and output data object structures. In the process of method chunk assembly (3) the reusable method chunks are constructed from the method fragments. Of utmost importance in DICE, are method chunks representing transformation task types that hold all the features for the required data transformations. Method chunks relevant for data integration and cleansing are taken to the method level and refined where required (4). From the method base they are instantiated and assembled into concrete DICE workflows (5). Fig. 24 provides an overview of these five phases.

DICE - Method Conceptualization

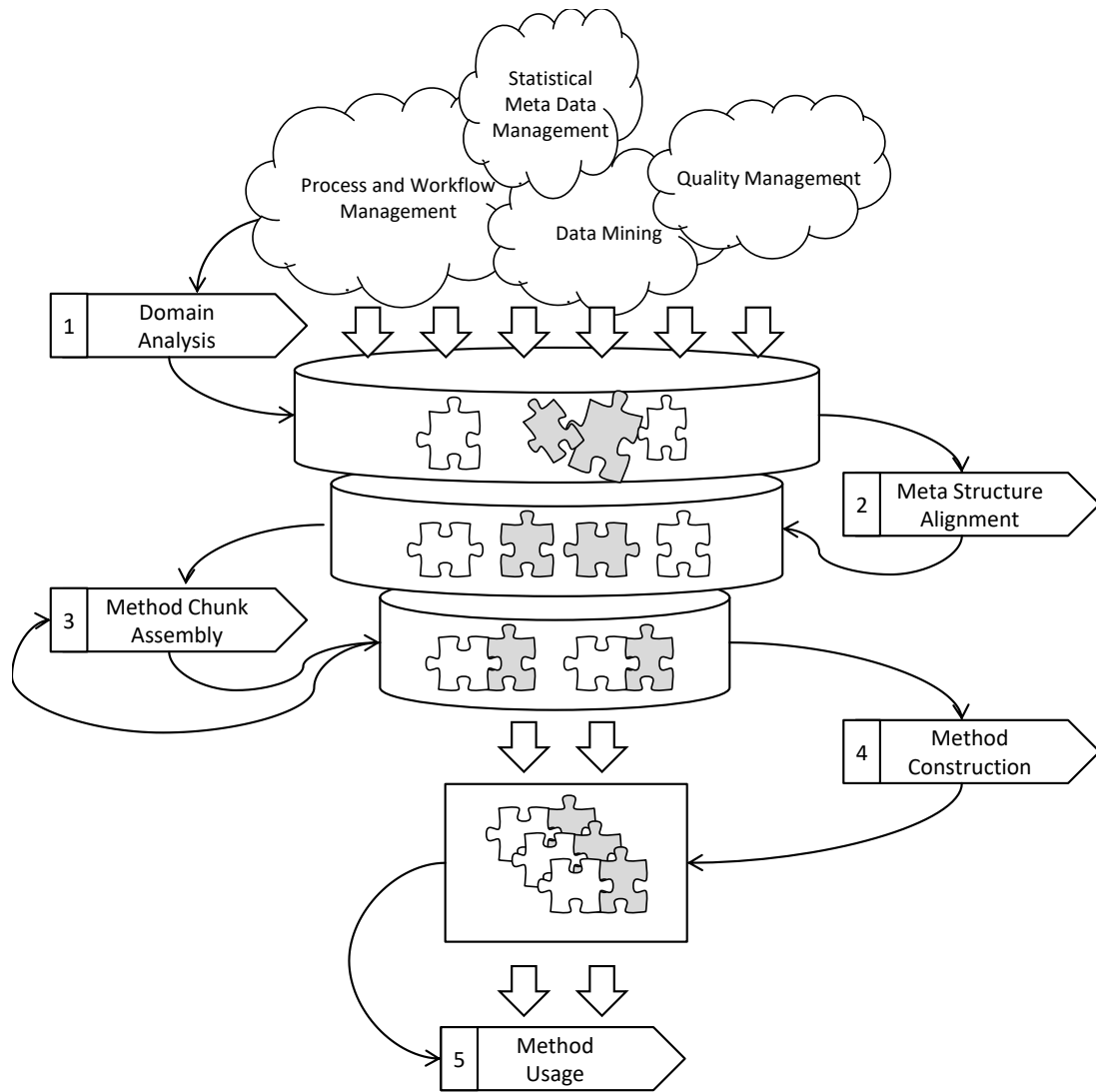


Fig. 24 Core components of situational methods

(Rolland and Prakash 1996) argue that the granularity and abstraction levels of method chunks are: context, trees and forests of trees where contexts represent atomic method chunks that can be put into a hierarchy (trees and/or forests). The top most important method chunk in DICE is the *transformation task type*. Transformation task types are structured in a refinement hierarchy where from the abstract top level transformation task subordinate transformation tasks are derived via specialisation. *Initialisation*, *selection*, *addition*, *variable removal*, *reclassification*, *consolidation* and *restructuring* are the main transformation types that can be further specialised or assembled as needed (see section 5.3.2 for the definitions of the transformation task types). Fig. 25 shows the refinement hierarchy of the transformation task types. For a better understanding the transformation task type “Consolidation” is exemplarily

DICE - Method Conceptualization

detailed. Consolidation is the atomic transformation task required for record linkage, i.e. for merging of observable units. The standard consolidation transformation matches equivalent observable units based on discrete matches. In these cases, an “all-or-nothing” principle (Dusetzina et al. 2014) is applied, i.e. the compared properties of the two observable units fully match. An example is the matching of observable units with the same ID or exactly the same name. In contrast, “fuzzy” consolidation represents a specialised consolidation task that uses similarity analysis to identify equivalent observable units. Observable units that match to a certain extent (a certain degree of similarity) are consolidated. By further specialisation, additional transformation tasks might be instantiated using more specialised functions for matching similar observable units. See section 7.2.2 for a detailed discussion of similarity functions applicable for identification of equivalent observable units in EA descriptions.

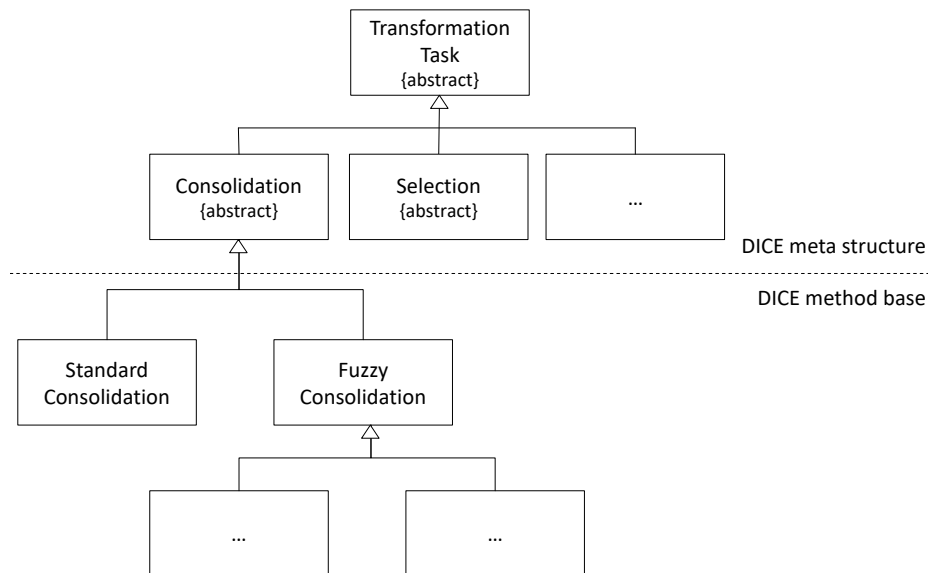


Fig. 25 Hierarchical breakdown of method chunks through specialisation/refinement

All of the levels hold method chunks. Method chunks on the top level are abstract and in contrast to concrete chunks, cannot be taken directly into a situational method. They need to be refined (specialised) first.

Applying the notion of modelling method as introduced by (Karagiannis and Kühn 2002), in DICE method chunks (respectively transformation task types) are made up of:

- modelling language parts: attributes, notation and semantics,
- procedure parts: the guidelines to parametrize tasks, and most importantly,
- algorithms for performing the transformations on the data objects.

DICE - Method Conceptualization

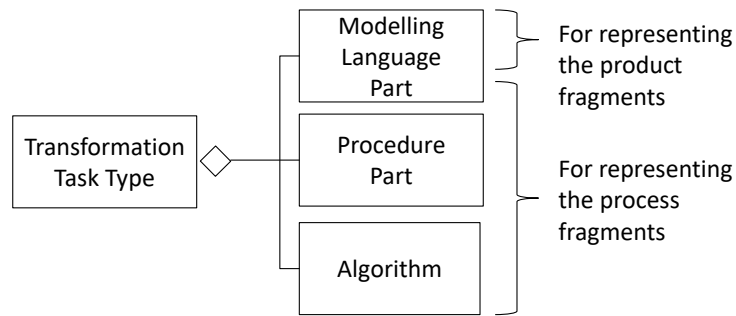


Fig. 26 Constitutional elements of transformation task types

Adopting OMG's multi-level hierarchy for SME, the DICE structure can be represented as shown in Fig. 27.

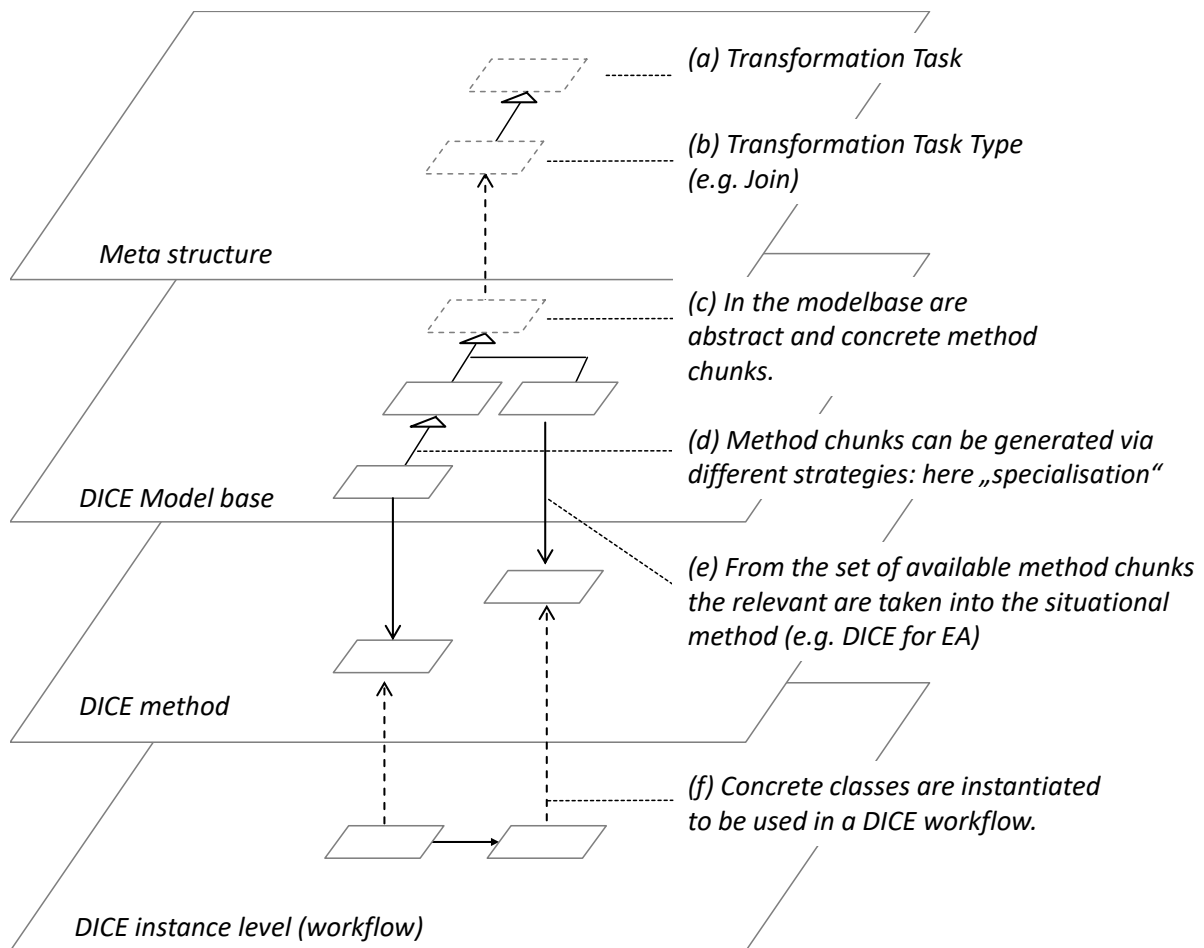


Fig. 27 Multi-level hierarchy of DICE

In DICE, transformation task types are refined hierarchically (a), (b), until they serve as reusable method chunks (c) within the method base.

DICE - Method Conceptualization

The method base consists of abstract and concrete classes. Its classes are instantiated from the meta-structure classes. DICE supports multiple strategies (d) to create method chunks, in conformity with (Ralyté, Rolland and Deneckère 2004) most of which require metamodeling features:

- **Assembly-based strategy:** assembly of existing method chunks by means of aggregation, decomposition and refinement of existing method chunks.
- **Extension-based strategy:** enhancing existing method chunks with novel features, e.g. by adding algorithms for calculating additional meta-data or by extending the set of attributes of classes in the product part for calculation of quality metrics.
- **Paradigm-based strategy:** instantiating new method chunks from the meta-structure.
- **Ad hoc:** creating methods and method chunks from scratch without instantiation from the meta-structure.

Only concrete modelling classes can be taken into DICE methods such as the DICE method for enterprise architecture analytics. On the bottom level, concrete instantiations of transformation tasks are positioned, put into sequence, connected to each other via “flow” relation and parametrised; they build up an executable DICE workflow.

An example of such an instance is a transformation task “Integrate Application Portfolios”, which takes two concrete “data objects” (such as the ones depicted in Fig. 19) as input and generates one integrated output data object. To this end, on model (instance) level at least one transformation task of type “addition” has to be parametrised; the transformation task requires a “name” and the two input data objects have to be specified. Depending on the required type of integration, subsequent transformations have to be applied. For example, in the case of an *inner join*, a subsequent selection transformation is applied in order to keep only observable units contained in both initial datasets.

In (Mirbel and Ralyté 2006), the authors propose a two-step approach for method engineering: (1) an assembly-based approach for constructing the method and (2) a so-called roadmap-based approach for refining the method by providing predefined method chunk configurations. With the concept of DICE subprocess, DICE offers a similar concept. The above example of combining the two transformation tasks (of type *integration* and *selection*) can be seen as a perfect example of such a subprocess. In this case, the subprocess serves as a

DICE - Method Conceptualization

reusable pattern (method chunk) composed of two atomic method chunks. The subprocess can be put into the method base. It serves as a pattern which can be reused whenever an inner join transformation is needed.

In the following sections, the constituent parts of the DICE method are introduced in detail. The first part (section 5.1) covers the DICE modelling procedure and discusses how DICE models are created, the roles which are typically involved and the interplay with the processes of situational method construction.

5.1 The DICE Modelling Procedure

The DICE method can be applied in diverse BA situations. In (Grossmann and Moser 2016), the authors suggest embedding DICE into data mining (DM) and knowledge discovery (KD) processes to unfold its full potential. The Cross Industry Standard Process for Data Mining (CRISP-DM) (Chapman et al. 2000), introduced in section 2.3, is one prominent example of such a process, or more precisely, of such a process-oriented reference model.

In combination with CRISP-DM, a DICE workflow represents a process instance hierarchically deducted from the superior CRISP-DM levels: (level 1) represents so-called *phases*, (level 2) is made up of *generic tasks* intended to cover all possible steps of a DM endeavour and (level 3) provides *specialised tasks* providing concrete input on techniques and how to apply them in specific situations. *Process instances* (level 4) reflect the instance level as enacted in a concrete BA endeavour. In this vein, the DICE workflows represent the 4th level and the DICE procedure is part of the 3rd level.

The phases and tasks of the DICE procedure model (3rd level) are presented in an idealised sequence. Depending on the given situation, tasks will likely be conducted in a different sequence, will be omitted completely and will definitely be shaped to the actual requirements. In many cases the tasks will need to be conducted over and over again with different parametrisation of the transformation tasks and by using additional or different data sources as an input.

“Modelling a DICE workflow, defining a sequential order of the required data transformation tasks, must be understood as analytical and creative work” (Grossmann and Moser 2016). Applying a reference model such as CRISP-DM, a data engineer specialises the generic recommendations from the reference model until reaching the specific level where the

DICE - Method Conceptualization

actual DICE workflow is designed. In the following sections the DICE procedure is introduced framed by the CRISP-DM phases.

Before starting with the actual data preparation and applying DICE, the phases *Business understanding* and *Data understanding* have to be accomplished. The first phase, *Business understanding* addresses the definition of the business objectives of the BA endeavour and requires assessing the situation with a focus on available resources, risks and given constraints. Based on these findings, the most important step of the phase, namely the definition of the Data Mining objectives and the data mining success criteria has to be conducted. Success might be defined by a certain level of predictive accuracy that must be achieved in the final delivery. With regard to the work product delivered through the data preparation, success criteria are defined together with the quality criteria deliberately derived from the DICE meta structure. Fig. 28 summarises the steps of the business understanding phase with emphasis on the DICE relevant process steps.

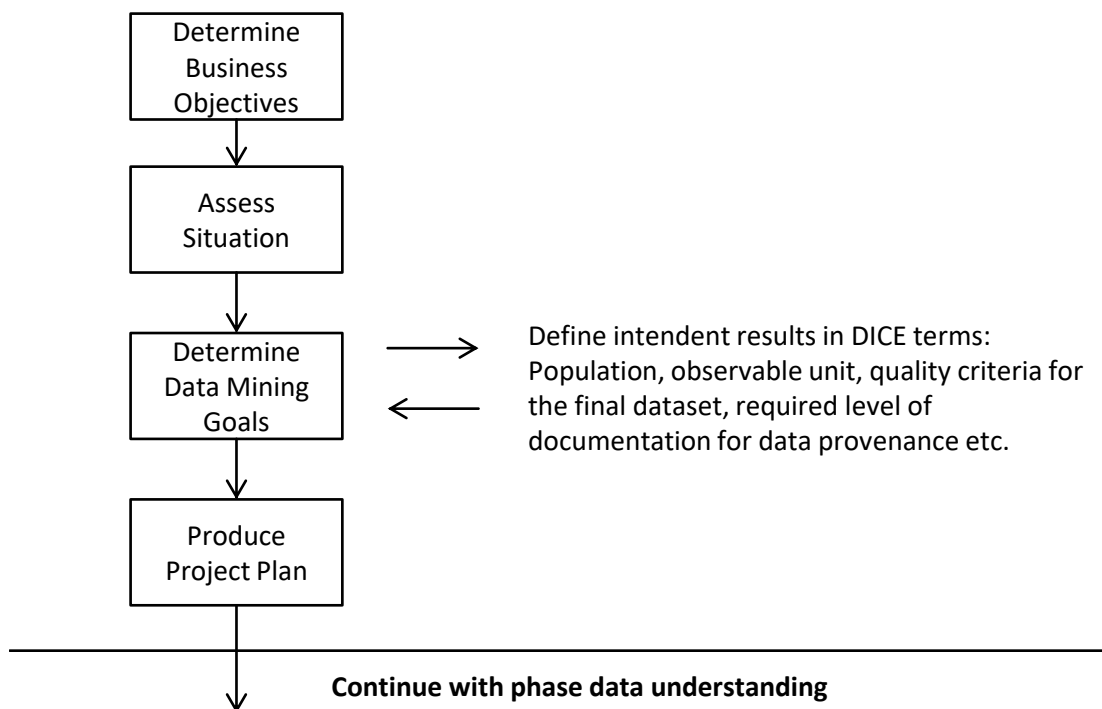


Fig. 28 Modelling procedure – Phase “Business understanding”, adapted from (Chapman et al. 2000)

In the **second phase** “Data understanding”, the source datasets are collected and explored. Conducting an initial analysis, the data analyst obtains an impression of the data quality of the input sources.

DICE - Method Conceptualization

The data analyst creates initial DICE workflows comprising transformation tasks allowing loading of the data and restructuring of the data as needed. In most cases, datasets in tabular format will be applicable (the common format of business analytics). Of utmost importance in this phase is the creation of composite data objects from the given sources. Thus, not only the datasets themselves but also metadata describing the sources have to be collected and characterised applying the structure of the composite data objects introduced above. This step is called *initialisation*. Any time a new source dataset is considered, it has to be initialised. Thus, all metadata have to be calculated from the input dataset and manually added where not automatically ascertainable.

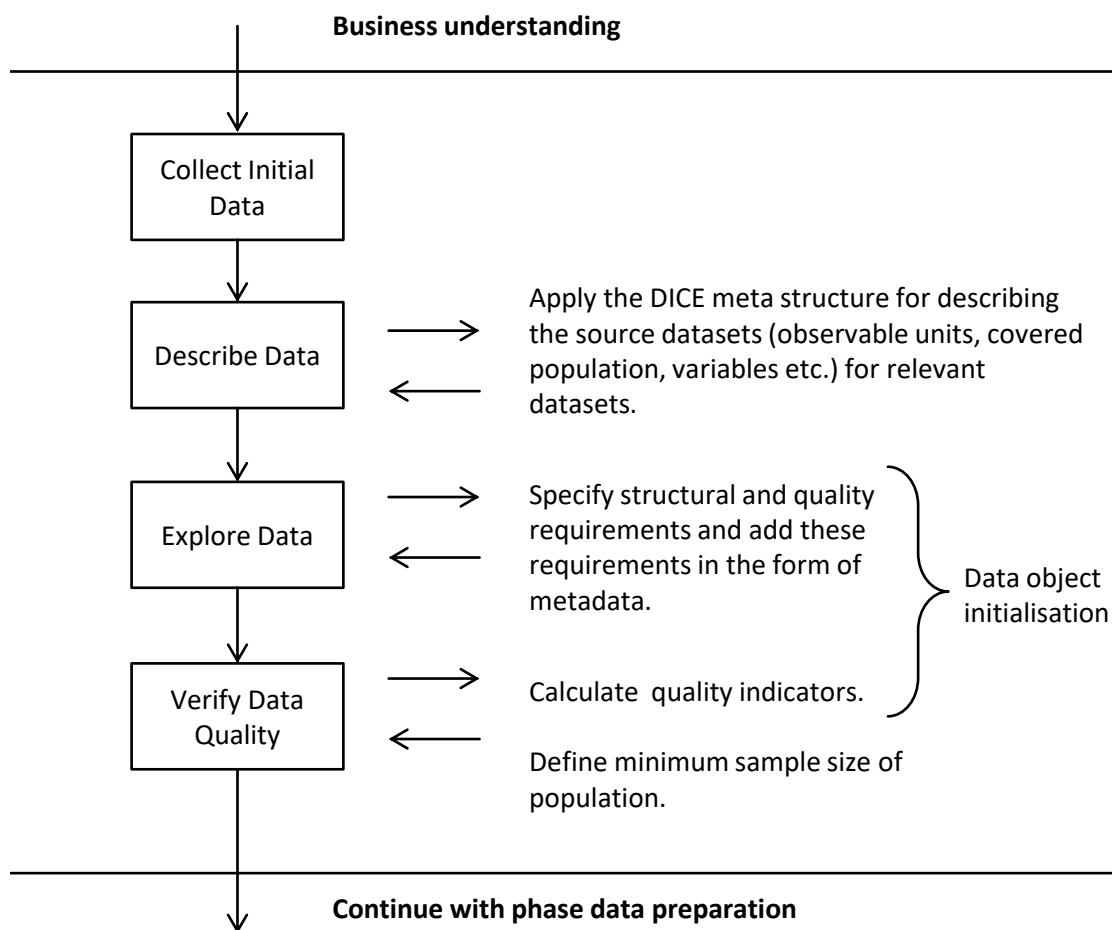


Fig. 29 Modelling procedure – Phase “Data understanding”, adapted from (Chapman et al. 2000)

For example, a typical dataset could be one holding application data as the application portfolio catalogue in Fig. 19. After loading the dataset, the variables have to be clearly specified. Each column represents one variable. The variable names and their semantic meaning have to be specified. Examples of variables in the given dataset are: ID, application

DICE - Method Conceptualization

name, operating costs and operating system. The variable “ID” could be considered as a unique identifier of the represented observable units.

By semantically interpreting the content of the dataset it becomes clear that the observable units are application components. Although somehow obvious in this case, one could also assume that operating systems represent the observable units in the given dataset. The population is not clear at all. It could be all application components of the entire organisation, the application components of a specific organisational unit or any other subset of application components. Without additional information the population represented in the dataset cannot easily be determined. Thus, additional metadata, which might not be extractable from the given input sources, has to be collected and imputed.

The same holds true for estimations of the data quality. DICE considers three types of quality data:

- **Instant quality indicators** can be calculated directly from the given property values. Examples are completeness, referential integrity etc.
- **Requirement-based quality indicators** are not that straight forward and need additional input for their determination. An example is the calculation of data freshness, which obviously requires a threshold, specifying the acceptable age of the data properties (derived from the last update timestamp).
- **Assessment-based indicators** cannot be calculated from the given dataset and have to be appraised when initially loading the dataset. (Berka et al. 2016) provide a catalogue of nine scored questions for such a quality appraisal. Example questions are: Is the variable important for the data source owner? What is the average time span in which data are adjusted in case of changes? Are the data revised on entry? Are there any kind of technical input checks applied to the data?

Detailed information on these metadata types is discussed in section 5.2 where the entire metamodel with all DICE meta objects is presented. Anticipating this, it can be stated that quality and structural requirements are formalised in the form of processing metadata and that DICE provides mechanisms for the automatic appraisal of property values against these requirements.

Another important step in this phase is the determination of the required population size: the population coverage. Determining the coverage (respectively the sample size) is critical.

DICE - Method Conceptualization

Oversized sample populations may waste time and resources, while samples that are too small often result in inaccurate results. Thus, the sample population has to be chosen in such a way that it makes inferences about the target population possible. In the literature, an abundance of techniques for the determination of sample sizes is discussed, see (Dupont and Plummer 1990). In essence, the required population size (sample size) is calculated from the following four determinants: (1) population size, (2) margin of error, i.e. confidence interval, (3) confidence level and (4) the degree of variability in the variables being measured (standard of deviation), see (Israel 1992). The required population size can be calculated as follows:

$$D^{\text{PROC} > \text{RequiredPopulationSize}} = \frac{Z_{1-\alpha/2}^2 p(1-p)}{e^2}$$

where Z^2 is the abscissa of the normal curve that cuts off an area α at the tails, e is the tolerable error for the sample mean (required level of precision) and p is the estimated proportion of an attribute that is present in the population (Israel 1992).

The third phase of the DICE procedure is **data preparation**. The initial DICE workflow is extended in this phase. Transformation tasks, such as select data, restructure data and consolidate data are applied with the aim of generating a sound dataset for the upcoming data modelling and evaluation phases.

DICE - Method Conceptualization

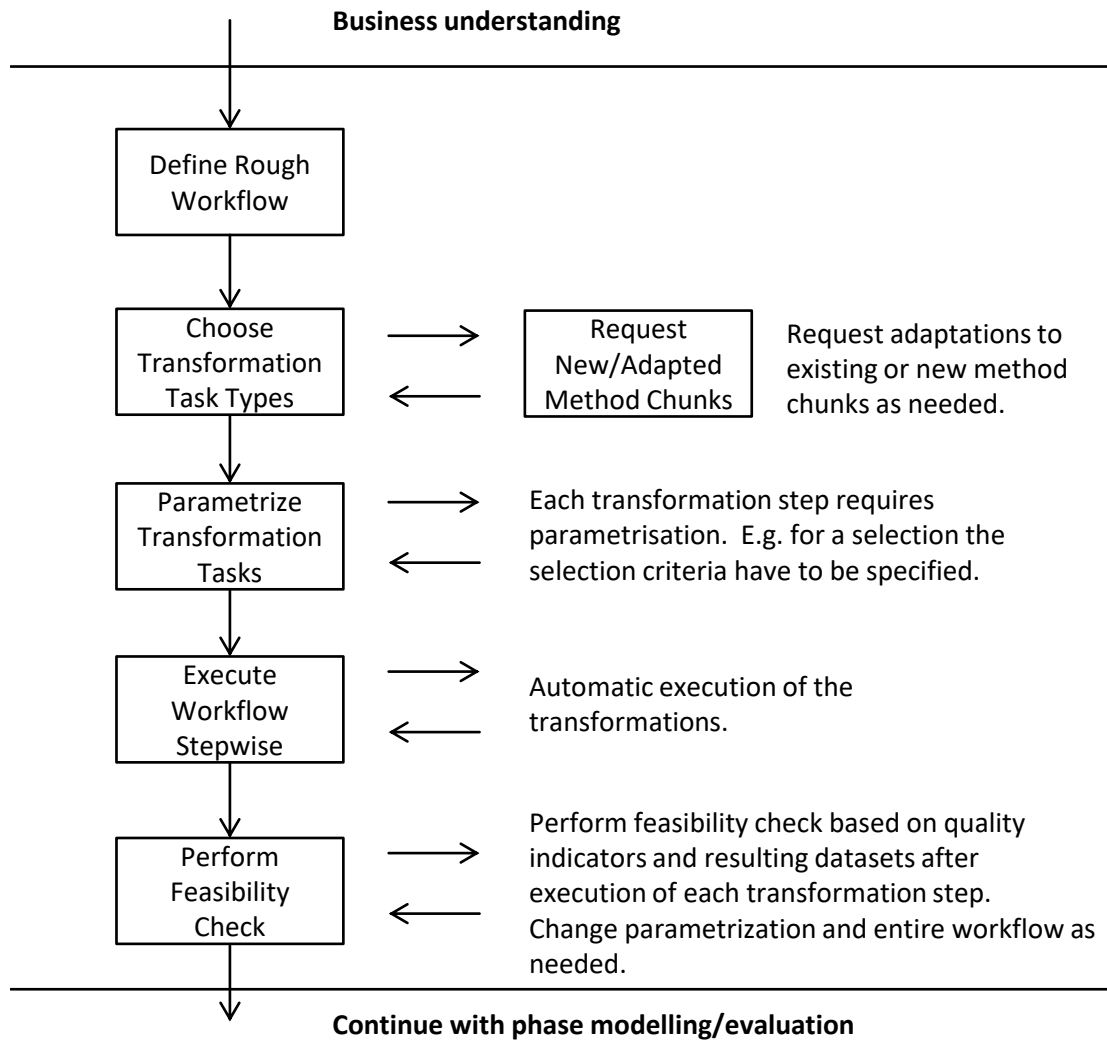


Fig. 30 Modelling procedure – Phase “Data preparation”, adapted from (Chapman et al. 2000)

Adding any transformation task to a DICE workflow involves searching for an adequate transformation task type (respectively method chunk) in the method base and instantiating it into the model. This step is supported by offering formal descriptions of the available method chunks, which facilitate the discoverability of suitable transformation task types. (Mirbel and Ralyté 2006) suggest describing the method chunks in a formalized way. A recommendation on how to describe DICE method chunks is shown in section 5.2, where the DICE metamodel is discussed in detail.

In a next step, each instantiated transformation task has to be parametrised and connected with the already instantiated transformation tasks. An example for parametrising is the assignment of input datasets to an addition task or to specify selection arguments for a

DICE - Method Conceptualization

selection task. The DICE transformation tasks are assembled via the “flow” relationship, which determines in what order the transformations have to be executed.

If a required transformation task type (method chunk) is missing in the method base, respectively in the method, or adaptations of given method chunks are required, business analysts will ask the method engineer to enhance the method base.

The prototypical DICE implementation (see section 8.1) also makes possible the creation of transformations from scratch. In this case the required transformations can be coded using a generic transformation task type as the starting point. Ideally, such cases are re-evaluated. Where suitable, these transformation tasks can be considered to be generalised/specialised and added to the method base.

Executing a DICE workflow generates an output data object. Based on the generated metadata (foremost on the calculated quality KPIs) and by analysing the actually generated dataset, the data analyst scrutinises the intermediate result of his modelling efforts and changes the workflow design as needed by aiming for an optimum workflow design. Usually, repeated backtracking within the designed workflow will be required for the purpose of correcting and optimising previously designed workflow parts or for adding/exchanging input data sources (Chapman et al. 2000).

This procedure continues until a satisfactory solution is achieved. The same holds true for the entire modelling procedure. At any point in time, the business analyst might decide to step back to a previous task and redo some of the already conducted tasks. Finally, the cleansed and integrated data objects are handed over to the *modelling, analysis and deployment phases*.

An important step when constructing and utilising a situational method such as DICE is of course, the task of “**Method Administration**”. It emphasises maintaining the method base. It has to be clearly defined under what conditions method chunks and fragments are added to the method base. In (Ter Hofstede and Verhoef 1997), the authors adduce “coherency” and “granularity” of method chunks as important categories for decision making. Too coarse grained method chunks will lead to higher efforts to refine the chunks for the given application scenario. On the other hand, if too fine grained, they will only be of value in rare cases. The hierarchical structured *DICE method base* alleviates that problem to a large extent. For example, a simple integration transformation might be supported by a method chunk “addition transformation task type”. In a standard case the merge task might cover merging

DICE - Method Conceptualization

datasets based on exact matching keys. A more sophisticated approach enhancing the “integration transformation task type” might support fuzzy merge (e.g. based on distance metrics). In this case, the method base comprises two integration tasks: the standard integration and the fuzzy merge as a specialisation of the former transformation task type. See Fig. 25 which elaborates on this example from the viewpoint of the DICE metamodel.

The following section discusses the DICE modelling language with strong emphasis on the DICE metamodel.

5.2 The DICE Modelling Language

The DICE modelling language can be divided into two parts: (1) the structural part and (2) the behavioural part. Initially inspired by the Unified Modelling Language (Rumbaugh, Jacobson and Booch 2004), this classification also reflects the two constituent parts of a method chunk namely the process fragments and the product fragments.

The concepts of the DICE modelling language, i.e. its disposable modelling classes and the relation classes that connect them, are grouped into these two categories. The structural part contains the concepts required to describe the data analysis object O^A and its corresponding meta data object O^M . The behavioural part consists of the concepts for defining the dynamic behaviour, namely for defining the workflows executing the transformations on the data objects.

UML, a modelling language widely accepted for illustrating knowledge representation languages, see e.g. (Brockmans, Haase and Studer 2006), (Buckl et al. 2007) and (Fischer and Winter 2007), is used to represent the DICE metamodel, see Fig. 31.

DICE - Method Conceptualization

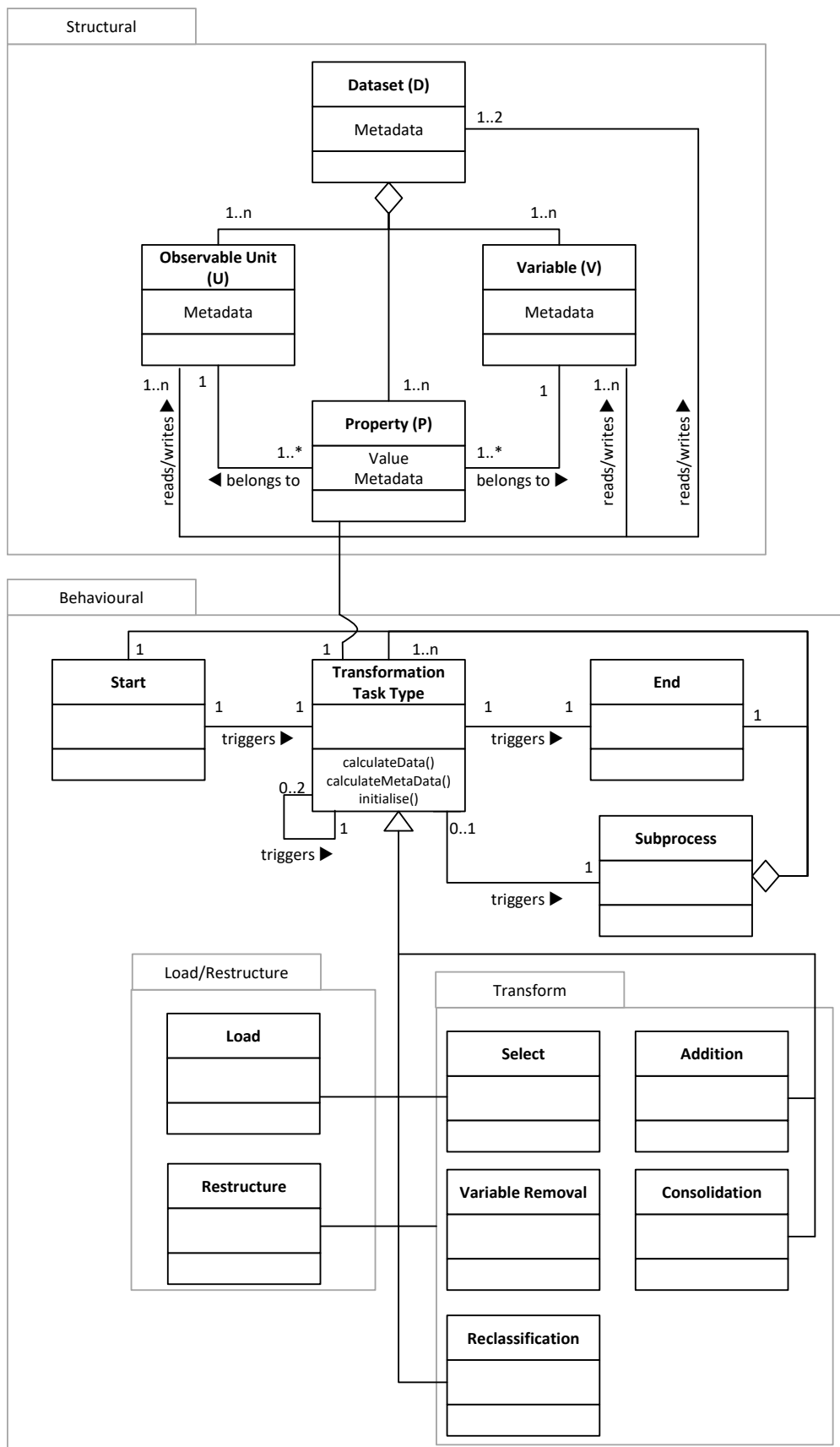


Fig. 31 DICE Metamodel Overview

DICE - Method Conceptualization

In the structural part of the metamodel one can find the aforementioned classes. Notably, only the class “property” is made up of both a data-level (the concrete values) and a meta-level. All other classes are represented solely by the meta-level, although inherent in the raw datasets (i.e. on data level).

The centrepiece of the behavioural part of the DICE metamodel is the transformation task type, which is specialised into more concrete transformation task types, such as selection, addition, consolidation and reclassification.

5.2.1 Behaviour concepts of the DICE metamodel

The **behavioural part** holds the constructs required for defining the processing of the data objects. A graphical workflow model depicting the conducted transformation tasks including their parametrisation contributes to the fulfilment of the data provenance requirements. Traceability is guaranteed by the modelled workflow itself. It represents the processing history of the data objects. By defining the transformation tasks, their parametrisation and the sequence in which the transformations have to be enacted, the modification of the data becomes comprehensible. The resulting data objects become reproducible.

A DICE workflow is defined by the following concepts: Start, Transformation Task Type, End and by the relation class “triggers”, which connects the aforementioned modelling classes. Start and End classes are required to define a structurally sound process (van der Aalst 1996). The class transformation task type specialises two classes: an atomic transformation called “Transformation Task Type” and a “Subprocess Task”, which links to an externally defined DICE process that in turn comprises the aforementioned concepts. The class “Subprocess” makes possible the assembly of transformation tasks into reusable method fragments. In this case, instantiated subprocesses serve as reusable patterns, which are stored in the DICE method base.

The abstract transformation task type class is in turn specialised into concrete transformation task types forming a specialisation hierarchy. The specialised transformation tasks are classified into the group of: “*get transformation tasks*” and “*integrate and cleansing transformation tasks*”. The latter specialises the transformation task types: *selection, addition, consolidation, reclassification and variable removal*. Inspired by the work of (Papageorgiou, Vardaki and Pentaris 2000), these transformation task types have been identified as the most

DICE - Method Conceptualization

important atomic transformation tasks for DICE. DICE provides means to assemble these atomic transformation tasks into more complex transformations (utilising the concept of subprocesses). Reflecting on the rich literature on data transformation algorithms, it becomes obvious that these archetypes often have to be further specialised.

Take the example of a consolidation transformation. In their survey on binary similarity and distance measures (Choi, Cha and Tappert 2010) cite more than 76 similarity and distance measures, all of them supporting “fuzzy merge” transformations. With the hierarchical concept of DICE it becomes possible to find a balance between a high number of specialised transformation task types versus a smaller number but with higher intrinsic complexity (accompanied by the increasing parametrisation efforts) of transformation task types. In this case, the method engineer has to decide whether to offer multiple distance metric functions within one transformation task type (by offering parametrisation options) or provide several transformation task types, each based on individual distance algorithms.

The second group of transformation task types is the group of *get transformation tasks types*. This group contains transformation task types for *accessing*, *restructuring* and *formatting* the input sources. *Access transformation task types* deal with the loading of the data from a given data source. Data might be retrieved from locally stored files but also from web resources or any other place. An example is loading data streams from social media platforms such as twitter⁴. *Formatting transformation task types* are required to interpret the data, which might reside in arbitrary technical formats such as csv (comma separated value), pdf (portable document format) or in proprietary vendor formats such as Microsoft’s xlsx (Excel file) and docx (Word file).

Besides the technical format, the datasets structural format has to be considered. In the context of EAM, data can come in various formats. The range extends from highly structured datasets instantiated from EA metamodels to less formally structured data such as architecture change requests, architecture vision papers and statements of architecture work, which typically reside in semi-structured text-heavy formats. All of these relevant datasets have to be reformatted into tabular form (horizontal format), the most typical input format for DM. As

⁴ www.twitter.com, see also (Grossmann and Moser 2016) where a specialised transformation task type for loading data from twitter is presented.

DICE - Method Conceptualization

with the group of cleansing and transformation task types, the *get transformation tasks types* are structured into a hierarchy.

Table 3 provides an overview of the introduced archetypical transformations and provides examples from the fields of EAM.

Table 3 Archetypical transformation task types – behaviour concepts in DICE

Archetypical Transformation Task Type	Definition/Example
Selection	<p>Conducting selection transformations limits the population and results in a reduced dataset. Observable units not meeting the selection criteria are removed.</p> <p>A simple example is the selection of application components using a certain operating system from an application portfolio catalogue such as the one of Fig. 19.</p>
Variable Removal	<p>Variable Removal transformations remove/discard one or more variables from a dataset.</p> <p>An example is to delete one variable (e.g. “Operational Costs”) and its properties including their values (and metadata) from the application portfolio catalogue dataset depicted in Fig. 19.</p>
Reclassification	<p>The reclassification transformation is used to convert the values of the properties of a variable from one grouping level to another.</p> <p>An example in context of the application portfolio catalogue is to change the “Operational Costs” from the measure unit TEUR into EUR.</p>
Addition	<p>The addition transformation combines two datasets by appending one dataset to another.</p> <p>An example is to merge the dataset of the application portfolio catalogue with a second dataset. Typically but not mandatory, both input datasets carry the same observable units and share at least a subset of variables.</p>
Consolidate observable units	<p>Consolidation of observable units serves a twofold purpose: (1) it is used for record linkage where duplicate observable units are removed and (2) to support groupby-transformations where a group of observable units is aggregated into a new observable unit.</p>

DICE - Method Conceptualization

	For each variable, aggregation functions have to be specified. Typical aggregation functions are: choosing values with the better quality and summative aggregations such as sum, max, average, mean etc.
--	---

5.2.2 Structural concepts of the DICE metamodel

The structural part defines the basic information logic items, namely the composite data analysis objects (O^A) and the composite metadata objects (O^M) introduced in section 4.1. Table 4 discusses the behavioural concepts of the data analysis objects and draws relations to the fields of EAM.

Table 4 Archetypical types of data analysis objects in DICE

Meta structure concept	Definition/Example
Observable unit	<p>Observable units are the entities for which information is collected and analysed.</p> <p>In the context of EA, many of them can be derived from the EA metamodels in use and are represented in the form of the metamodels' modelling classes and relations. Entities of interest are the typical EA building blocks stipulated in the EA metamodels. According to TOGAF, a building block is defined as a (potentially) reusable component of the enterprise architecture. TOGAF cites actors, business services, application components and data entities as examples of building blocks. In their work, (Aier, Riege and Winter 2008b) provide a comprehensive overview of typical building blocks. They list and score the building blocks according to their importance in EA endeavours. In general, each of these building blocks is an observable unit candidate for EA-related BA endeavours.</p>
Dataset	<p>The set of data collected for describing the observable units is called dataset. Datasets comprise observable units and their properties.</p> <p>In EA typical datasets are architecture descriptions, such as catalogues (e.g. the application portfolio catalogue in Fig. 19), matrices and models. However, many times the descriptions might not be that structured, and relevant observable units, variables, properties etc. have to be extracted first.</p>
Variables	<p>The properties of observable units are classified into variables.</p> <p>In the context of the given example (Fig. 19), the variables are the columns of the dataset.</p>
Properties	<p>Properties carry the values collected/available for observable units.</p> <p>In the context of EAM they differentiate the building blocks (observable units). In Fig. 19</p>

DICE - Method Conceptualization

	the properties are represented (as is the rule) in the cells of the application portfolio catalogue dataset.
Population	<p>A population represents the set of observable units.</p> <p>In DICE, the population is represented as metadata of the dataset. The difference between the entire population (scope) and the observed part of the population (sample population) represented in a given dataset must be distinguished.</p> <p>In the context of an enterprise architecture, typically the entire population of building blocks is of relevance. Examples are: the population of business processes, of application components (possibly documented in an application portfolio catalogue as in Fig. 19) or all technologies in use.</p>

In general it can be denoted that it is difficult to determine the appropriate metadata (Sundgren 1996). For deciding which meta information is worth keeping, the application domain has to be analysed. For the EAM domain this is done in section 7.1. However, on the DICE meta structure level predefinitions are made. Utilising the presented refinement mechanisms, adequate metadata objects can be derived as needed.

On meta structure level the DICE metamodel structures the concepts into observable unit, dataset, variable and property. Each of these concepts captures metadata, which is categorised into the following aspects, inspired by the work of (Kent and Schuerhoff 1997):

- **Semantic metadata** carry contextually relevant data that provide a level of understanding about the data, i.e. the meaning of the data.
- **Logistic metadata** is used to provide technical information on the given datasets. Examples are: locations, technical data formats, access criteria etc.
- **Process metadata** is used for processing/transforming the data. All metadata required for conducting the transformations and for documenting the performed transformations are stored in this aspect. Additionally, the process metadata represent the requirements that the property values have to comply with.
- **Quality metadata** are of utmost importance. Data transformations heavily impact the data quality. Quality measures are applied to all concepts of the data level.

Fig. 32 shows the big picture. For a better overview, only the DICE-specific meta data structures are presented in detail. That is, common metadata such as the ones defined in the *Dublin Core Metadata Element Set*: creator, description, publisher, contributor, date, type,

DICE - Method Conceptualization

language, identifier, source, relation, coverage and rights (Dublin Core Metadata Initiative 2012) can be added as needed, leveraging the DICE meta modelling capabilities. There are a great many different metadata standards available across disciplines, such as the book industry, library science, geography, archiving, e-commerce, ecology, arts and education. Dublin Core is a general purpose standard, which can be used cross-discipline. See (Duval 2001) and (Smith and Schirling 2006) for an overview.

On the following page the structural part of the DICE metamodel is depicted. **Please note that you can find the graphic split and zoomed in the annex for better readability**, see section 10.1.

DICE - Method Conceptualization

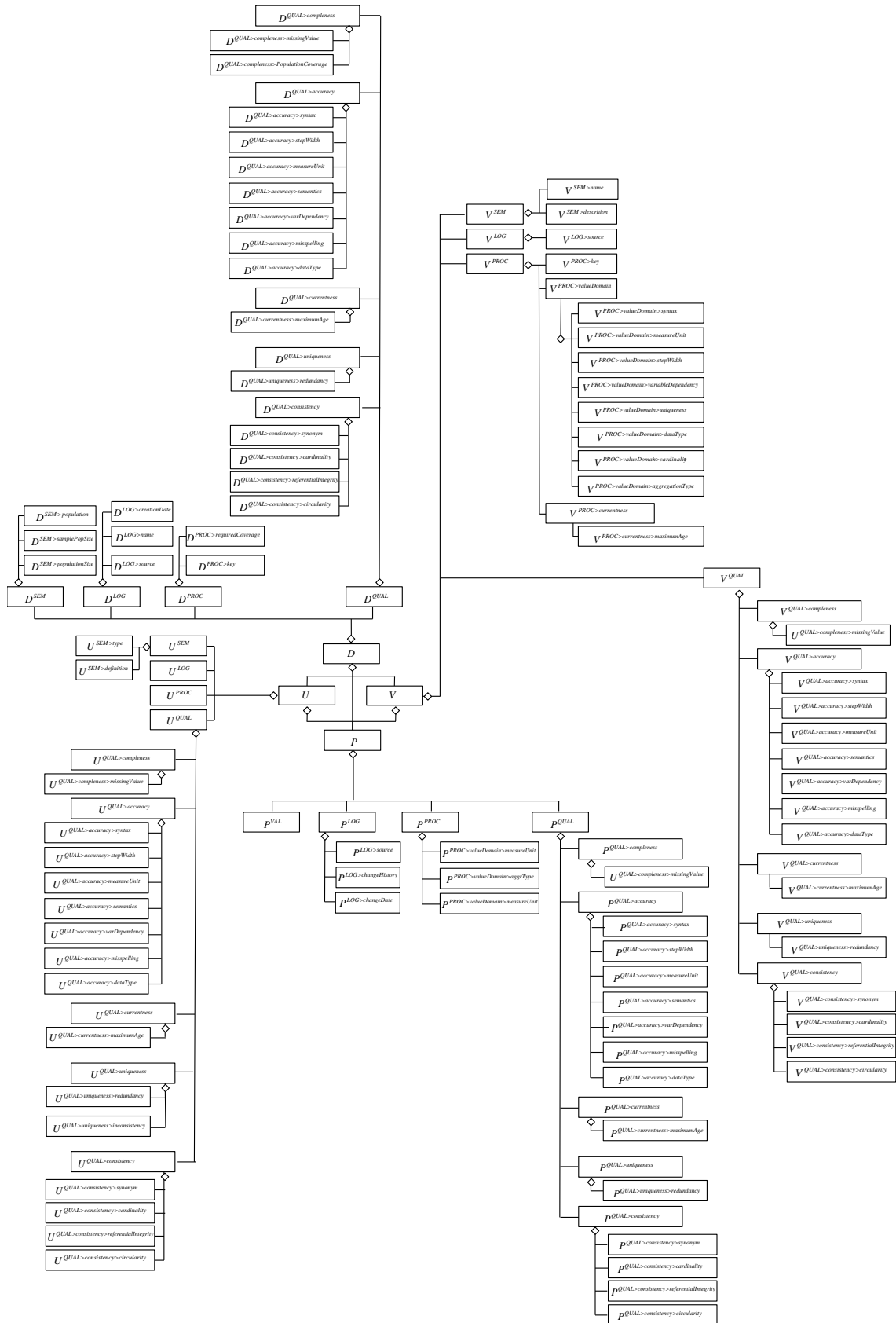


Fig. 32 Complete MM of the DICE structural part

DICE - Method Conceptualization

In a narrow sense, the dataset D^A is solely represented by its constituent properties. On data level, structural information such as the observable units and the variables are inherent but not explicitly represented. Even if a given dataset consists of observable units with unique IDs in the dataset and column headings, this information has to be extracted and sufficiently defined on metadata level. As has been discussed in the introduction to section 4, the input datasets have to be initialised; taking the data level as an input, metadata have to be added to create the required composite data object. The initialisation transformation task is presented in section 5.3.2.1 in detail.

It creates the composite data object comprising data and metadata objects and is defined as follows:

$$D^A = \{P_{uv}\} \xrightarrow{\text{initialise}} \begin{cases} P_{uv} \rightarrow \{VAL, LOG, PROC, QUAL\}, \\ V \rightarrow \{SEM, LOG, PROC, QUAL\}, \\ U \rightarrow \{SEM, LOG, PROC, QUAL\}, \\ D \rightarrow \{SEM, LOG, PROC, QUAL\} \end{cases}$$

where VAL represents the set of atomic property values on data level, and SEM , LOG , $PROC$ and $QUAL$ represent the sets of meta data objects, which are hierarchically structured into sub meta objects.

Elements within this hierarchically structured metamodel are denoted via superscript that presents the path from one of the four base categories to the concrete metadata values. The “greater than sign” is used to depict the navigation paths. Take the following example: $V_v^{PROC>ValueDomain>measureUnits}$ provides information about the specification of the measure unit of variable v which is categorised as processing information (category: PROC). It is required to specify the value domain of the variable. Likewise, $P_{uv}^{QUAL>syntax}$ provides information about whether the syntax of the value held in property P_{uv} complies with the specifications in $V_v^{PROC>valueDomain>syntax}$.

An unambiguous definition of the observable units requires specifying the metadata collected in the metadata category SEM, where the observable units are defined in natural language. As an example: $U^{SEM>type}$ provides information about the semantic definition of the

DICE - Method Conceptualization

observable unit. To be able to precisely identify an observable unit within a dataset, unique identifiers for the observable units have to be defined. Thus, the variables constituting the data key within the dataset must be indicated. The data key of a dataset can be defined by one or more variables whose property values contained within are used to uniquely identify each of the observable units.

$$D^{PROC>key} = \{V_v \mid V_v^{PROC>key} = true\}.$$

Due to the immanent importance of data quality aspects in the context of data preparation, a detailed discussion of these aspects is provided in section 6.

The set of the DICE metadata cannot be considered as a final set. Refining transformation task types, as discussed in section 5.2.1, will typically pose requirements on the structural concepts, i.e. on the product fragments. Take the example of standard and fuzzy consolidation introduced in section 5.2.2. In addition to the standard metadata, similarity measures and distance measures have to be documented in order to keep the conducted consolidation steps comprehensible. The same extensibility requirements hold true for the DICE algorithms, as will be discussed in the upcoming section.

5.3 DICE Algorithms and Mechanisms

Algorithms and mechanisms are a third important building block of a modelling method. DICE comes with two basic types of algorithms: (1) algorithms operating on the entire workflow and required for its execution and (2) algorithms (as part of the process fragments), for performing the transformations that hold the processing logic of the actual data transformations.

5.3.1 Macro Level: Execution of DICE Workflows

The DICE workflows are structured in the form of sound workflow nets using a BPMN-like notation. See (White 2004) for an introduction to BPMN. The macro level algorithms ensure the processing of the transformation tasks in the defined sequence. The algorithms verify whether all preconditions are fulfilled before executing a transformation. Typical preconditions are the checking of whether all required input parameters are available in the model.

DICE - Method Conceptualization

For the execution of a DICE workflow the execution logic of Petri nets is applied. For an introduction to the application of Petri nets in the context of workflow management see (Van der Aalst 1998). More specifically, mechanisms of place/transition nets (P/T nets) are applied where each DICE transformation task represents a transition in the Petri net. The places in the Petri net represent the state of the data objects after performing a transformation. The tokens within the Petri net represent input and output datasets.

Only if a transformation was successful (that is to say, all required data and metadata have been generated, or rather, inputted) are the upcoming transitions ready to be invoked (ready to ‘fire’). Depending on the complexity of the DICE workflow, there can be more than one data object (tokens) within the system. When a transition fires (i.e. a transformation is performed), it consumes the required input data objects (tokens) and typically creates one output data object (token) in its output places (the state after performing the transformation). Thus, only if preconditions are fulfilled, does the algorithm move on to the next transformation task. Fig. 33 presents a basic DICE workflow comprising three transformation tasks: two for loading the datasets and converting them into data objects (initialize) and one for integrating the data objects via *addition* transformation. The corresponding Petri net is also shown.

DICE - Method Conceptualization

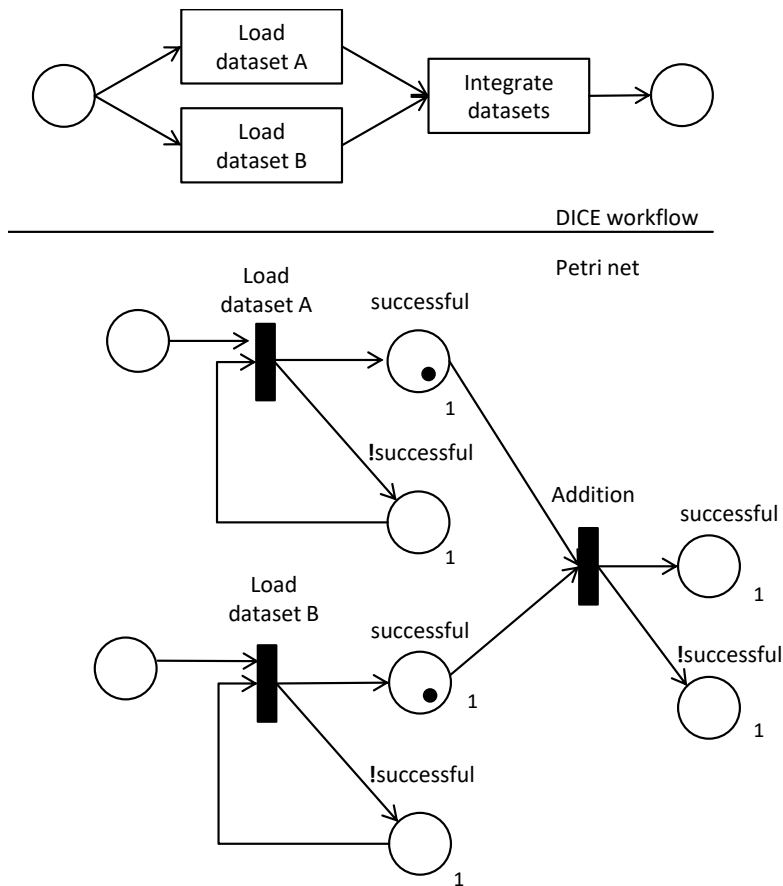


Fig. 33 Petri net representation of an exemplary DICE workflow

The marking in the Petri net shows that both datasets have been successfully loaded and initialised. As soon both datasets (represented by the tokens) are available, the transition “addition” can fire and produce the integrated dataset. For a more detailed discussion on Petri net execution algorithms see (Murata 1989).

5.3.2 Micro Level: Algorithms for Performing Data Transformations

Performing a transformation task requires initialised data objects and a parametrised transformation task. The transformation algorithms are part of process fragments of the method and typically compose: (1) dedicated transformation algorithms, transforming data and metadata and (2) a (re-)initialisation algorithm applied after performing the actual transformation task. The first type addresses the algorithms dedicated to performing the transformation on data and metadata level. The latter has to be applied after performing the transformations to re-calculate the affected metadata.

DICE - Method Conceptualization

For the data transformations, dedicated algorithms are applied. One can easily find dozens of merge algorithms, see e.g. (Mishra and Eich 1992) and (DeWitt, Naughton and Schneider 1991) for an overview. For incorporation into DICE, these algorithms typically have to be extended and restructured to fit into the DICE meta structure. Join algorithms such as the *nested loop join*, the *sort-merge join* and the *hash join* are perfect examples. Choosing an appropriate join algorithm depends on the data structure and on the size of the given input datasets.

Fig. 34 depicts the popular *nested loop join* as an example. In its most basic form the algorithm uses one dataset as the outer input table and the second dataset as the inner input table. In the outer loop the first dataset is consumed row by row (i.e. observable unit by observable unit). The inner loop, executed for each outer row, searches for matching rows in the inner input table.

```
for each observableUnit  $U_r$  in dataset  $D_r$  do  
  for each observableUnit  $U_s$  in  $D_s$  do  
    if  $U_r$  and  $U_s$  satisfy the join condition  
      then output the observableUnit  $\langle U_r, U_s \rangle$ 
```

Fig. 34 Nested loop join on data level

To perform the required calculation on metadata level, DICE extends these generic transformation algorithms by either appending additional algorithms, e.g. the initialisation algorithm for recalculating the metadata or by directly extending the given algorithms. A simple example based on the nested loop join is shown in Fig. 35. The join algorithm is extended with outputting similarity measures as part of the quality metadata. The initialisation algorithm recalculates the metadata. Details on the (re-)initialisation can be found in section 5.3.2.1.

DICE - Method Conceptualization

```
for each observableUnit  $U_r$  in dataset  $D_r$  do
  for each observableUnit  $U_s$  in  $D_s$  do
    if  $U_r$  and  $U_s$  satisfy the join condition
      then output the observableUnit  $\langle U_r, U_s \rangle$ 

    % extend algorithm to output similarity degrees
    % per observable unit as part of the metadata %
    output the similarity degree  $SD(U_r, U_s)$ 

% initialization algorithm %
initialize (output dataset)
output final data object
```

Fig. 35 Merge algorithm extended with fragments for calculation of meta objects

Summing up, algorithms are intimately intertwined with the metamodel. Specialising and refining method chunks requires the adaptation of its algorithms and likewise of its modelling classes. Parametrisation options for a transformation task type, the structure of the meta data objects and the algorithms must be aligned to form an instantiable method chunk.

In (Denk, Froeschl and Grossmann 2002), the authors present a framework for documenting statistical data processing on data and metadata level. Algorithms for altering data taking metadata into account have been discussed in (Vardaki, Papageorgiou and Pentaris 2009), (Papageorgiou, Vardaki and Pentaris 2000) and (Papageorgiou et al. 2001). DICE builds on these approaches. It defines atomic transformation task types and provides means to instantiate and assemble these tasks into data preparation workflows. Whereas the quoted research on metadata management defines strict preconditions to avoid biased data generated by inappropriate transformation tasks, DICE is designed to be more failure-tolerant by allowing processing of inadequate transformations. Instead of strictly defined preconditions, DICE provides quality indicators to the data engineer after performing the transformations. Based on these quality data (calculated via *initialisation transformation* after performing the actual data transformation task), the data engineer can reconsider and adapt previously taken transformation tasks.

Depending on the type of transformation performed, different kinds of metadata require subsequent alteration. For example, whereas a *selection* transformation applied to a dataset will create a subset of the population and thus the semantic metadata of the population requires an update, the *reclassification* of property values of a variable affects the metadata of the variable but not of the population.

DICE - Method Conceptualization

In the following sections the archetypical transformation tasks of DICE are introduced.

5.3.2.1 Initialisation

The *initialisation* transformation creates the composite data object comprising data and metadata. Input for the initialization transformation is either a raw dataset $D^{A,(i)}$ for which the metadata have to be created or a DICE data object $O^{(i)}$ for which the metadata need to be recalculated. Output of the initialization are the data objects which are made up of the dataset, the observable units, the variables and the properties, each including its metadata. Metadata are recorded manually or automatically calculated. Formally $T_{initialise}$ can be represented as follows:

$$T_{initialise}(D^{A,(i)} \vee O^{(i)}) = \{D^{(o)}, U^{(o)}, V^{(o)}\} = O^{(o)}$$

For the output dataset: semantic metadata, logistic metadata, process metadata and some of the quality metadata have to be manually added by the data analyst.

Depending on the type of quality indicators, the assessment must be carried out manually or can run fully automated. For example, whereas semantical correctness of a property value requires human appraisal, many other quality indicators such as the completeness of a variable, can be calculated automatically. In section 6 the list of DICE quality indicators is discussed in detail.

In the case of large amounts of data, a manual appraisal of all observable units will not be sufficient. Hence, a set of observable units (a sample) needs to be selected from the given dataset, which is less in number (size) but adequately represents the population so that true inferences about the data quality of the entire dataset can be obtained. A two-stage approach is suggested where in a first step for a selection of observable units, an estimation of the data quality is conducted. This information is then used to construct the missing quality indicator values of the non-evaluated properties. This approach is commonly referred to as *data imputation*. There are manifold approaches and algorithms for data imputation that can be classified into context dependent imputation (based on association rule techniques) and context-independent imputation techniques (based on clustering techniques), see (Allison 2002) for an overview. The context-independent approach hypothesizes a probabilistic relationship between the assessed properties and the missing values. Based on this, the

DICE - Method Conceptualization

occurrence probabilities of the properties are estimated. The subsequent data imputation is done randomly based on the estimated probabilities. The minimum sample size required, avoiding biased results, can be determined by applying sample size determination methods as have been discussed in section 2.4.1.4. The initialization algorithm in pseudo code can be found in the annex in section 10.2.1.

For a better understanding, Table 5 exemplarily illustrates the metadata of the application portfolio catalogue after initialisation based on the DICE metamodel.

Table 5 Example data object after initialisation

Metadata	Example	Remark
$D^{SEM>population}$	All applications used within the organisation	
$D^{SEM>sample}$	All applications used within the organisation	Note that the finite population of all applications is ideally covered: $D^{SEM>sample} = D^{SEM>population}$.
$D^{SEM>PopulationSize}$	800	Based on estimation The sample size of the population can be calculated from the observable units. Assume $ U =755$
$D^{LOG>source}$	EA repository	The organisation's EA repository is the source of the given dataset.
$D^{LOG>name}$	Application portfolio catalogue report	In most cases the file name or the heading of the table
$D^{LOG>creationDate}$	03/12/2016	MM/TT/YYYY
$D^{PROC>key}$	ID	As a unique identifier, the variable ID is defined.
$D^{QUAL>total}$	0,83	83 %, calculated from the quality indicators of the subordinated concepts (U, V, P)
...
$D^{QUAL>uniqueness}$	1	100 %, assuming no duplicate IDs in the dataset
...

DICE - Method Conceptualization

$U^{SEM>description}$	<i>“An application component is defined as a modular, deployable and replaceable part of a software system that encapsulates its behaviour and data and exposes these through a set of interfaces.”</i>	Note: In this example the definition of application component is taken from Archimate (The Open Group 2016).
$V^{SEM>description}_{OperatingCost}$	The yearly expenses which are related to the operation of the application component including license cost, infrastructure cost and HR related costs.	A precise definition of the semantic meaning of the variable must be defined. In this example of the variable “Operating Cost”
$V^{PROC>valueDomain>measureUnit}_{OperatingCost}$	€/year	
...
$U^{QUAL>total}_{1020}$	0,93	Quality indicators are evaluated per observable unit. The example shows the estimated total quality of the application CosMos (with ID 1020). Details on how to calculate the total quality indicator are presented in section 6.
...
$P^{VAL}_{1020,operatingCost}$	23.000	Operating costs of the observable unit “CosMos” with ID 1020 23.000 is the value assigned to the property.

5.3.2.2 Selection

Conducting a selection transformation limits the population of a given input data object by removing those observable units not meeting the selection criteria (Papageorgiou, Vardaki and Pentaris 2000). Formally the selection transformation can be denoted as follows:

$$U_{select}^{(o)} = T_{select}(O^{(i)}, selectionCriteria) = \{U_i \in O^{(i)} \mid selectionCriteria\}$$

where $selectionCriteria$ consist of

$$\{P_{iv}^{VAL}, operator, constant\} \vee \{P_{iv}^{QUAL}, operator, constant\} \vee \{U_i^{VAL}, operator, constant\}.$$

DICE - Method Conceptualization

Thus, selection criteria may refer to values of the dataset (P_{iv}^{VAL}) on data level but also to quality metadata of observable units (U_i^{QUAL}) and/or of quality metadata of single properties (P_{iv}^{QUAL}). An *operator* in this context is a reserved word or a character to perform operations. Operators as part of the selection criteria are typically of the following types: arithmetic, comparison and logical. The selection algorithm in pseudo code can be found in the annex in section 10.2.2.

5.3.2.3 Addition

With this archetypical transformation, two datasets typically containing observable units of the same type or with equivalent variables are appended. If the set of contained observable units is not disjointed, duplicate observable units might arise in the output dataset.

DICE considers the classical merge (join) transformation as a composite transformation made up of *addition* and subsequent *selection* transformations. Operating on data and metadata concurrently, after performing an addition transformation, the possible duplicates can be distinguished by their metadata and subsequent selection of unique observable units (see section 5.3.2.2) remains possible. The addition transformation can formally be denoted as follows:

$$O_{addition}^{(o)} = T_{initialise}(T_{addition}(O^{(i_k)}, O^{(i_l)})) = T_{initialise}(O^{(i_k)} \cup O^{(i_l)})$$

where $O^{(i_k)}$ and $O^{(i_l)}$ denote the input data objects. $T_{initialise}$ has to be performed to recalculate the metadata.

Matching variables and concatenating the variables' properties is expedient only for those variables that carry the same semantic meaning. A simple but in many cases sufficient equivalence function may consider the variables' names. Two variables of the input data objects i_k and i_l can be concatenated only if they have equivalent names:

$$V^{(o)} = \{V^{(i_k)} \times V^{(i_l)} \mid V_m^{(i_k), SEM > name} \approx V_n^{(i_l), SEM > name}\}$$

where each of the index m indicates a variable of the data object $O^{(i_k)}$ and the index n represents a variable in $O^{(i_l)}$.

DICE - Method Conceptualization

More elaborative equivalence functions will take additional metadata such as the variable definition and the value domain into consideration. In an ideal case the datasets do not contain duplicate observable units such that subsequent record linkage transformations are not required. The intersecting set of observable units of the two input objects is empty in this case; respectively the output data object (after performing the addition transformation) does not contain any observable units labelled with a high degree of similarity to another observable unit of the dataset. Formally this can be denoted as follows:

$$U^{(i_k)} \cap U^{(i_l)} \neq \{\} \Leftrightarrow T_{select}(O^{(o)}, uniqueU) = 0$$

where the selection criteria *uniqueU* retrieves all observable units marked by a high similarity value $U^{(o), QUAL > accuracy > similarity}$ which has to be calculated as part of the initialisation transformation. See section 7.2.2 for calculating similarity degrees.

In the annex in section 10.2.3 the addition algorithm is presented in pseudo code.

5.3.2.4 Variable Removal

Variable removal is involved with the discarding of one or more variables from a dataset. DICE refrains from the concept of “projection” quoted in (Papageorgiou et al. 2001) and (Papageorgiou, Vardaki and Pentaris 2000). A projection transformation in DICE is considered as a composite transformation task made up of a *variable removal transformation* and possibly followed by an *observable unit consolidation* transformation (see section 5.3.2.6). The latter is required only if a variable belonging to the data key is removed and the remaining dataset consists of duplicate entries according to the residual key attributes.

The variable removal transformation can formally be denoted as follows:

$$O_{variableRemove}^{(o)} = T_{initialise}(T_{variableRemove}(O^{(i)}, V_r^{(i)})) = T_{initialise}(\{V_i \in V^{(i)} \mid V_i \neq V_r^{(i)}\})$$

and

$$if V_r^{(i)PROC > key} = true : T_{consolidation}(O_{variableRemove}^{(o)})$$

The variable removal algorithm in pseudo code can be found in the annex in section 10.2.4.

5.3.2.5 Reclassification

Reclassification transformations change the classification schema of a variable, i.e. they convert the properties of a variable from one grouping level to another. (Papageorgiou, Vardaki and Pentaris 2000) call this type of operation “grouping transformations”. They differentiate between two types of grouping transformations: (1) transformations applied to key variables of the dataset and (2) transformations applied to any other variables. DICE considers these grouping tasks as composite transformation task types: first, the actual *reclassification transformation* and in a second step (if data key variables are affected) a *consolidation* transformation (see section 5.3.2.6).

Reclassification transformations allow transformation of the value domains of a variable into another. In this context, DICE acknowledges the three main types of statistical data: numerical (discrete and continues), categorical and ordinal. Equivalent variables can be transformed by applying transformation rules. Property values that do not meet the conditions of these rules are assigned the value ‘not available’ (NA). Transformation rules are specified via simple lookup tables or via conversion functions.

The reclassification transformation can formally be denoted as follows:

$$O_{reclassification}^{(o)} = T_{reclassification}(O^{(i)}, V_r^{(i)}) = \begin{cases} RF(P_{ir}^{(i), VAL}), & \text{if } P_{ir}^{(i), QUAL > valueDomain} = fail \\ P_{ir}^{(i), VAL}, & \text{otherwise} \end{cases}$$

where $RF()$ denotes the reclassification function that delivers a reclassified value in case of success or the value ‘NA’ for ‘not available’ if the reclassification failed. $RF()$ is only applied if the property value $P_{ir}^{(i), VAL}$ does not comply the defined value domain, i.e. $P_{ir}^{(i), QUAL > valueDomain} = false$. See section 6.2 where this quality KPI is defined in detail.

The reclassification algorithm in pseudo code can be found in the annex in section 10.2.5.

5.3.2.6 Consolidation

Consolidation is the transformation step which merges observable units. As a precondition, observable units have to be equivalent. In DM, equivalence of data is typically measured via similarity/distance functions where similarity functions calculate the similarity and distance functions calculate dissimilarity. Pairs of observable units with high similarity have a low distance and vice versa.

DICE - Method Conceptualization

Two observable units are considered as equivalent if they compose similar property values in a defined set of variables. Not necessarily all values have to be similar. Thus, duplicate observable units are often identified taking only their data key into account. Duplicate observable units might arise through incorrect data entry, integration transformations, etc. but also through *reclassification transformations* (as discussed in section 5.3.2.5) or *variable removal transformations* which affected the data key (see section 0). In most cases data engineers try to avoid duplicate observable units by either assigning a unique key to the similar (all) observable units to keep them distinguishable or by consolidating the observable units in the case of real duplicates. Consolidation of duplicates is usually referred to as *record linkage* in the DM domain. A third option is to consolidate observable units via aggregation mechanisms onto an observable unit of a superior grouping level. DICE distinguishes between these types $T_{consolidationRecordLinkage}$ and $T_{consolidationAggregation}$.

$$T_{consolidation}(O^{(i)}) = T_{initialise} \left(\begin{array}{l} T_{consolidationRecordLinkage} \left(U^{(i)} \times U^{(i)} \mid SF(U_a^{(i)}, U_b^{(i)}) > threshold \right) \vee \\ T_{consolidationAggregation} \left(U^{(i)} \times U^{(i)} \mid SF(U_a^{(i)}, U_b^{(i)}) > threshold \right) \end{array} \right)$$

For record linkage, naive strategy is to remove one of the observable units:

$$T_{consolidationRecordLinkage}(U_a^{(i)}, U_b^{(i)}) = U_a^{(i)} \vee U_b^{(i)}.$$

A more elaborate strategy is to choose the observable unit of higher quality or to create a new observable unit choosing the properties with the higher quality indicator from each of the input observable units. In the latter case, for all of the variables of the pair of observable units, the *total property quality* has to be compared. The property with the lower quality indicator is withdrawn. Formally this can be denoted as follows:

$$T_{consolidationRecordLinkage}(U_a^{(i)}, U_b^{(i)}) = U^{(o)} = \forall V_c \in V^{(i)} : P_{i,c}^{(i)} \mid P_{i,c}^{(i), QUAL > total} > P_{j,c}^{(i), QUAL > total}, (i, j \dots a, b)$$

The consolidation function for performing *aggregation* transformations is similar. In contrast to the deduplication approach, the pairs of property values (per variable) are aggregated instead of choosing one of the properties. Formally the *aggregation consolidation transformation* can be denoted as follows:

$$T_{consolidationAggregation}(U_a^{(i)}, U_b^{(i)}) = U^{(o)} = \forall V_c \in V^{(i)} : aggregation(P_{a,c}^{(i)}, P_{b,c}^{(i)})$$

where *aggregation()* is a function that aggregates the property values per variable.

DICE - Method Conceptualization

The consolidation algorithm in pseudo code can be found in the annex in section 10.2.6.

5.3.2.7 Restructure

Consequently, techniques are needed to restructure data sources into this horizontal layout. To restructure relational data, the discussed transformation task types can be applied. However, there are a great many other types of data structures ranging from structured data to semi-structured data such as plain text. The difference between the types of data is not sharply defined and varies depending on particular disciplines and data representations. Roughly speaking, semi-structured data does not conform to strict standards. It is not strictly typed and not strictly constrained by a schema as compared to structured data. In contrast to structured data, which is typically considered to be relational or object-oriented data where each property has a designated variable, semi-structured data spans a continuum from plain text documents to fully-structured data.

Abiteboul's (Abiteboul 1997) definitions provide an overview of indications for semi-structuredness, see Table 6.

Table 6 Characteristics of semi-structured data based on (Abiteboul 1997)

Characteristic	Description
Irregular structure	The data consists of heterogeneous entities. In some cases the entities are incomplete, meaning that information is missing. In other cases additional information is available. Furthermore, different types for the same information type might be used.
Implicit structure	<p>In the case of implicit structure, a well-formed structure exists. An example is an XML document which has been successfully validated against the corresponding doctype declaration (DTD). By parsing the document the contained entities, their variables and properties can be extracted.</p> <p>However, the data is considered to be semi-structured, since extraction and reformatting steps have to be applied and the interplay between the parse tree and the required relational representation is not always obvious. (Mourya and Saxena 2015) provide an overview of schema conversion methods between XML and relational models by comparing seventeen different approaches.</p>

DICE - Method Conceptualization

Partial structure	Partial structure is given if parts of the data are not structured. Examples are documents with embedded graphics (e.g. in png format).
Indicated structure	<p>In contrast to a constrained structure, which presupposes a strict typing policy, an indicated structure permits deviations from the given typing policy.</p> <p>An example is the data guide approach propagated by the Lore Project of Stanford University, see e.g. the work of (McHugh et al. 1997). It proposes a database management system for XML-based data which does not require an explicitly defined schema.</p>
Non-A-priory schema	Unlike traditional relational or object-oriented data structures, in this case, the schema is not fixed prior to capturing the data. The data is often already given and has to be structured to support querying and analysis of the data.
Large schema	In contrast to structured data where the schema is precisely defined, semi-structured data often comprises a wide schema. Transforming these data sources into relational schema creates wide sparse tables.
Ignored schema	For some of the typical scenarios the schema is ignored. Examples are, the simple browsing and searching for information without taking the schema into account.
Rapidly evolving schema	As opposed to relational and object-oriented data, the schema is rapidly changing. Thus, velocity not only refers to the speed at which new data is generated but also the ever-changing and non-predictability of its structure.
Data is eclectic	<p>This characteristic applies to the versatility of data, taking into account that the structure of a data entity depends on the point of view and point in time the data entity is used.</p> <p>Whereas at the beginning only a few attributes might be considered, the entity might be enriched and restructured for further use in later phases of a given BA initiative.</p> <p>An example in EA is that for a business process one might just require the name and the description in a first phase, whereas in later phases a detailed list of process tasks has to be extracted from the process description.</p>

DICE - Method Conceptualization

Data is blurred	<p>In semi-structured data there is often no clear distinction between schema and the data content. Data content and schema are intertwined in a single format.</p> <p>Typical examples are XML-based documents where no DTD (doctype declaration) is given.</p>
-----------------	--

Considering this wide range of data structure types, a multiplicity of approaches to extract relevant data has been developed in recent years. *Information extraction* is the process of selecting and restructuring text fragments identified via text pattern matching in one or more text documents. The results of this extraction process differ from case to case. However, generally speaking, *information extraction* results in populating some type of database (Cowie and Lehnert 1996). Usually the focus lies on the processing of human language text by employing *natural language processing* techniques (NLP). As of late, content extraction from images, audio and video must also be considered under the roof of information extraction. In general, information extraction is part of the greater puzzle of data mining and KDD (see section 2.3).

In the context of EAA specifically: the sub fields of *entity recognition* and *relationship extraction* play an important role. Entity recognition is concerned with the identification of named entities based on predefined categories, such as locations, expressions of time or specifically to EAM: of technologies, application components and any other of the typical EA building blocks. Relation extraction is concerned with the identification of relations between the entities (e.g. application component >> runs on >> technology). In the context of EAM, relation extraction focuses on extraction of relations between the building blocks. In an EA, the building blocks and the relations together form a tight net. The schema which makes possible the storage of the identified entities and relations, i.e. the building blocks and the relationships between them is the EA metamodel introduced in section 2.4.1.3.

In section 7.2.1 an example of a restructure transformation task is shown. In the example the XML-based structure of the Archimate Model Exchange Format is transformed into the tabular-structured DICE schema.

5.4 Summary

DICE is designed following the principles of situational method engineering. Data may reside in many different formats and structures. A multitude of data mining techniques has been developed to prepare and cleanse the data for actual data mining needs. DICE is intended to support any data preparation endeavour providing a meta structure for integrating and assembling existing algorithms and techniques into a situational BA method. In this way, the individual algorithms and techniques serve as method fragments which are restructured to fit into DICE and serve as reusable method chunks. In this vein, DICE does not strive for a universal solution for every kind of data preparation problem. DICE is designed as a situational method allowing method engineers to assemble individual methods from its method base.

In this context, metamodeling plays an important role. Applying agile method engineering techniques as introduced by (Karagiannis 2015), DICE is made up of three main building blocks:

- a modelling language to design data integration and cleansing processes,
- a procedure providing guidance on how to accomplish a data preparation endeavor and
- the algorithms for performing the data and metadata transformations.

At its core, DICE provides the data structures for holding required data and metadata in the form of a composite data object. It consists of the main transformation tasks required for data preparation which operate on these data objects, (such as selection, addition etc.). Both the DICE metamodel as well as the transformation tasks (algorithms) are designed to be specialised and extended as needed.

6 DICE Quality Framework

6.1 Quality Models and Quality Issues

It has been stressed that quality measures are of immanent importance in the data preparation phase and for DM endeavours as a whole. For that reason DICE comes with a framework that makes possible the definition of data quality requirements and the evaluation of these requirements.

According to (Fröschl and Grossmann 2001), quality assurance in statistical data processing has to consider:

- the design of the production process (i.e. of the DICE workflow) as a whole,
- the quality of data transformations within the production process and most importantly,
- the quality of the generated data objects.

There are many data quality models, all of them comprising similar quality criteria. One prominent example is the “Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model” (ISO/IEC/IEEE 25012 2008), which defines fifteen data quality characteristics that must be reflected when assessing a data product. The DICE output datasets can be understood as such a data product. The standard supports definition and evaluation of data quality requirements in “*data production, acquisition and integration processes*” (ISO/IEC/IEEE 25012 2008). It comprises data quality characteristics, such as accuracy, completeness, consistency, credibility, currentness and accessibility. A decision regarding which of these data quality criteria have to be applied and what efforts are to be invested related to apprising significant quality assessments, is of course, dependent on the application domain and situational circumstances.

When conducting data production and preparation processes, one has to consider two perspectives on data quality: (1) the actual data quality inherent in the given data source and (2) the expected data quality that is required for the subsequent analysis phases (Berti-Équille 2007). Due to this relative aspect, DICE has to cope with data quality in a twofold manner: it must support data quality appraisals and the matching of data quality against defined data quality requirements. In this vein, DICE focusses on the *inherent data quality aspects* as

DICE Quality Framework

opposed to *system-dependent data quality aspects*. According to (ISO/IEC/IEEE 25012 2008), inherent data quality refers to “*the degree to which quality characteristics have the intrinsic potential to satisfy stated and implied needs [...]*”. System-dependent data quality focuses on software and user interfaces, and thus, on aspects, such as availability, portability and recoverability of the data. DICE presupposes that the data is available and thus does not consider quality aspects of the source systems which provide the data. Focusing on data-inherent quality aspects, DICE considers the following quality dimensions taken from (ISO/IEC/IEEE 25012 2008):

- Accuracy: data have to represent the true values correctly.
- Completeness: datasets provide all required variables and values for these variables.
- Consistency: data are coherent and data dependencies are clearly specified.
- Creditability: data are considered to be reliable by the users.
- Currency: data are of a suitable age for the given purpose.
- Precision: data reside in the required measure units.
- Traceability: data provenance is comprehensible.
- Understandability: observable units, variables and population are unambiguously defined.

In order to measure these quality aspects, a more detailed analysis of possible data quality issues is required. Taxonomies of data quality problems have been presented by (Rahm and Do 2000) who divide data quality problems into single-source problems and multi-source problems. On the next level they categorise quality problems into schema and instance level problems. Single-source problems arise mainly from data entry (e.g. misspellings, contradicting values) whereas multi-source problems (duplicates, overlappings etc.) are rooted in performed data integration activities. (Kim et al. 2003) present a more detailed taxonomy of dirty data. Their taxonomy is hierarchically structured and comprises more than twenty typical quality issues accompanied by techniques for preventing, checking or cleaning the quality issues. (Oliveira et al. 2005) also provide a comprehensive taxonomy. Additionally, they present methods for detecting data quality issues based on checklists structured into binary trees according to their hierarchical taxonomy of dirty data. With their rule-based taxonomy on dirty data, (Li, Peng and Kennedy 2014) provide a taxonomy that makes

DICE Quality Framework

possible the classification of the data quality issues into classical data quality dimensions, such as accuracy, completeness, currentness, consistency and uniqueness.

In the following, typical data quality issues extracted from the above research are introduced and defined. The problem fields are taken from (Oliveira et al. 2005). Additional problem fields extracted from the dirty data taxonomies of (Rahm and Do 2000), (Kim et al. 2003) and (Li, Peng and Kennedy 2014) are added:

- Missing value: Properties without values.
- Syntax violation: Values not conforming to the defined patterns.
- Outdated values: Property values not conforming to the defined data freshness criteria.
- Interval violation: Numeric data values that are not within specified boundaries.
- Set violation: Categorical data values not in conformity with the predefined values.
- Wrong datatype: Property values violating the defined datatype, e.g. strings in a numeric variable.
- Misspelling: Incorrectly spelled values.
- Meaningless values: Values not conforming to the designated meaning of the variable.
- Erroneous value: Values that do not violate any constraints but which are simply wrong.
- Lack of value: An example is, when a part of the required entry is missing/cropped.
- Value with imprecise and doubtful meaning, e.g. values consisting of abbreviations.
- Uniqueness violation: Values that violate an identity rule.
- Synonyms existence: Occurrence of syntactically different values with the same semantic meaning.
- Semi-emptiness: Observable units with many missing values defined in light of a certain percentage of missing values.
- Inconsistency among values: Violation of defined dependencies between property values of two variables.
- Redundancy about an entity: Duplicate observable units within one dataset.
- Inconsistency about an entity: Special case of duplicate observable unit with inconsistent property values.
- Missing tuple: Missing observable units within a specified population.
- Referential integrity violation: Wrong reference to an observable unit.

DICE Quality Framework

- Outdated reference: Outdated reference to an observable unit.
- Circularity among tuples in a self-relationship: Recursive relationship problem.
- Cardinality violation: when defined minimum/maximum references between observable units of two datasets are not fulfilled.
- Non-compliant measure units: Property values not measured in the defined measure unit, e.g. cost values defined in EUR and not in Dollar.
- Representation inconsistency: Different value denomination of the same type, e.g. Boolean values defined in true/false and as specified in 1/0.
- Heterogeneous aggregation levels: Different level of abstraction, e.g. costs per product and costs per product group.
- Synonyms existence: Syntactically different values with the same semantic meaning, e.g. “customer” versus “client”.
- Homonyms existence: Syntactically equal values with different semantic meaning. The term “glass” can be interpreted as “magnifying glass” or “tumbler”.

6.2 DICE Quality Profile

In Table 7 the data quality issues are categorised along the DICE meta structure. Additionally, inspired by (Li, Peng and Kennedy 2014), they are assigned to a quality dimension. According to Rahm and Do (Rahm and Do 2000), schema-related issues comprise issues that originate from bad schema design, schema translation and schema integration. In contrast, instance-related data quality issues arise in the actual datasets and cannot be avoided on the schema level. The classification into schema and instance level can also be found in the table. From the definition of schema vs. instance level, requirements for DICE are drawn. Schema-level violations can typically be covered/healed via concise definition and examination of metadata. Thus, at minimum for the schema-level issues, DICE has to provide clear specifications of the admissible property values by precisely defining their value domain. Data property values have to be assessable against these definitions. The column “Metadata” of Table 7 defines the metadata carrying these definitions.

DICE Quality Framework

Table 7 Common quality indicators organised along the DICE metamodel

Problem field	DICE element	Single/multi source	Schema vs. instance	Defined in DICE Metadata	Quality perspective & DICE target quality indicator
Missing value	P	Single	instance	$V_v^{PROC>valueDomain>syntax}$	Completeness measured in $P_{uv}^{QUAL>completeness}$
Syntax violation	P	single	schema	$V_v^{PROC>valueDomain>syntax}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>syntax}$
Outdated value	P	single	instance	$V_v^{PROC>currentness>max.Age}$	Currentness measured in $P_{uv}^{QUAL>currentness>maximumAge}$
Interval violation	P	single	schema	$V_v^{PROC>valueDomain>syntax}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>syntax}$
Set violation	P	single	schema	$V_v^{PROC>valueDomain>syntax}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>syntax}$
Wrong datatype	P	single	schema	$V_v^{PROC>valueDomain>dataType}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>dataType}$
Imprecise value	P	single	schema	$V_v^{PROC>valueDomain>stepWidth}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>stepWidth}$
Misspelling	P	single	instance/ schema	(not required)	Accuracy measured in $P_{uv}^{QUAL>accuracy>misspelling}$
Meaningless value	P	single	instance	$V_v^{SEM>description}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>semantics}$
Erroneous value	P	single	instance	$V_v^{SEM>description}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>semantics}$

DICE Quality Framework

Problem field	DICE element	Single/ multi source	Schema vs. instance	Defined in DICE Metadata	Quality perspective & DICE target quality indicator
Lack of value	P	single	instance	<i>(not required)</i>	Accuracy measured in $P_{uv}^{QUAL>accuracy>semantics}$ or $P_{uv}^{QUAL>accuracy>syntax}$
Value with imprecise and doubtful meaning	P	Single	instance	$V_v^{SEM>description}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>semantics}$
Uniqueness value violation	P	Single	schema	$V_v^{PROC>key}$	Consistency measured in $P_{uv}^{QUAL>uniqueness>redundancy}$
Synonyms existence	P	Single	schema	<i>(not required)</i>	Currentness measured in $P_{uv}^{QUAL>consistency>synonym}$
Semi-emptiness	U	Single	instance	<i>(not required)</i>	Completeness measured in $U^{QUAL>completeness}$
Inconsistency among attribute values	P	Single	schema	$V_v^{PROC>valueDomain>varDep}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>variableDep}$
Redundancy about an entity	D	single	schema	<i>(not required)</i>	Uniqueness measured in $U_u^{QUAL>uniqueness>redundancy}$
Inconsistency about an entity	D	single	instance	<i>(not required)</i>	Uniqueness measured in $U_u^{QUAL>uniqueness>inconsistency}$
Missing tuple (observable unit)	D	single	instance	$D^{PROC>requiredCoverage}$	Completeness measured in $D^{QUAL>completeness>popCoverage}$

DICE Quality Framework

Problem field	DICE element	Single/ multi source	Schema vs. instance	Defined in DICE Metadata	Quality perspective & DICE target quality indicator
Referential integrity violation	P	Single	schema	<i>(not required)</i>	Consistency measured in $P_{uv}^{QUAL>consistency>refIntegrity}$
Outdated reference	P	Single	schema	$V_v^{PROC>currentness>maxAge}$	Currentness measured in $P_{uv}^{QUAL>currentness>maximumAge}$
Circularity among tuples in a self-relationship	P	Single	schema	<i>(not required)</i>	Consistency measured in $P_{uv}^{QUAL>consistency>circularity}$
Cardinality violation	P	Multi	schema	$V_v^{PROC>valueDomain>cardinality}$	Consistency measured in $P_{uv}^{QUAL>consistency>cardinality}$
Syntax inconsistency	V	Multi	schema	$V_v^{PROC>valueDomain>syntax}$	Accuracy average from $P_{uv}^{QUAL>accuracy>syntax}$
Non-compliant measure units	P	Multi	schema	$V_v^{PROC>valueDom>measureUnit}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>measureUnit}$
Representation incons.	P	Multi	schema	$V_v^{PROC>valueDomain>syntax}$	Accuracy measured in $P_{uv}^{QUAL>accuracy>syntax}$
Synonyms existence	D	Multi	n.a.	<i>(not required)</i>	Consistency measured in $D^{QUAL>uniqueness>synonymy}$
Homonyms existence	D	Multi	n.a.	<i>(not required)</i>	Consistency measured in $D^{QUAL>consistency>homonymy}$
Redundancy about an entity	D	Multi	n.a.	<i>(not required)</i>	Uniqueness measured in $D^{QUAL>uniqueness>redundancy}$

DICE Quality Framework

Problem field	DICE element	Single/multi source	Schema vs. instance	Defined in DICE Metadata	Quality perspective & DICE target quality indicator
Inconsistency about an entity	D	Multi	n.a.	(not required)	Consistency measured in $D^{QUAL>uniqueness>inconsistency}$

In a detailed analysis of the above atomic quality indicators, it becomes clear that some of these cannot be evaluated automatically. Whereas quality aspects such as *missing values* and compliance with defined *syntax* requirements can be calculated by cross-checking with value domain requirements (defined in V^{PROC}), others such as *meaningless value* have to be assessed manually. In order to obtain the atomic quality indicators (pass vs. fail), the property values have to be compared with the specifications defined in the set of quality requirements V^{PROC} that are made up of the following parameters:

- $V^{SEM>description}$: This parameter carries the documentation of the semantic meaning of the properties assigned to the variable.
- $V^{PROC>key}$: This parameter states whether the properties of the variable serve as data key or are part of the data key of the given dataset.
- $V^{PROC>currentness>maximumAge}$: The values of the properties have to be of the right age. This parameter carries the allowed maximum age measured in days.
- $V^{PROC>valueDomain>uniqueness}$: This parameter defines whether values of properties have to be unique throughout the variable.
- $V^{PROC>valueDomain>measureUnit}$: The measure unit has to be unambiguously defined. Examples are: currencies in the case of monetary values or measure units for temperature such as Celsius and Fahrenheit.
- $V^{PROC>valueDomain>stepWidth}$: This parameter defines the required precision of the data. For example, monetary values could be measured in EUR or TEUR.
- $V^{PROC>valueDomain>dataType}$ defines the required datatype for the properties of variables.
- $V^{PROC>valueDomain>syntax}$: This quality characteristic defines the abstract set of possible values for the properties of the variable. Intervals and value sets are typical examples.
- $V^{PROC>valueDomain>aggregationType}$: In some cases it is important to understand whether the property values are atomic or represent summative values (e.g. such as max, min, mean or average). Condensing averaged values in many cases will lead to biased results.

DICE Quality Framework

- $V^{PROC>valueDomain>variableDependency}$: Property values of variables often pose dependencies on one another. Metadata have to provide information about these dependencies.
- $V^{PROC>valueDomain>cardinality}$: This variable definition carries the minimum/maximum number of outgoing/incoming relations of an observable unit.

DICE ascribes the same importance to each of the problem fields of a certain quality category (quality aspect). The problem fields are measured via dedicated atomic quality indicators. These indicators are categorised into quality categories. The atomic quality indicators of properties are rated in a Boolean variable: *pass* for proper and *fail* for erroneous. On property level this can be formally denoted as follows:

$$P^{QUAL} = \{qualityCategory_1, qualityCategory_2, qualityCategory_3, \dots\},$$

$$P^{QUAL>qualityCategory} = \{indicator_1, indicator_2, indicator_3, \dots\}$$

and

$$P_{uv}^{QUAL>qualityCategory>indicator} = pass \vee fail$$

For each property, a set of quality indicators P^{QUAL} is defined. The set of quality indicators is categorised into quality categories which in turn comprise the atomic quality indicators of a variable. Atomic qualities (per property) can be either fulfilled (*pass*) or non-fulfilled (*fail*). In the following the rules for evaluating the atomic quality indicators are defined.

For the evaluation of values regarding **missing values** the following rule is applied:

$$P_{uv}^{QUAL>completeness>missingValue} = \begin{cases} pass, & \text{if } P_{uv}^{VAL} \neq (\{ \} \vee NA) \\ fail, & \text{otherwise} \end{cases}$$

where *NA* stands for ‘not available’ or any other symbol indicating non-existence of the value. Examples for *NA values* are: default values such as 01.01.1970 (Unix time) for date values, NULL values such as the string ‘no entry’ etc.

Syntax violations may come in many flavours and are dependent on the data type of the variable. Simple data types, such as numeric, strings, date and complex data such as arrays (sets) are considered. The following rule checks whether a property value is compliant with the defined datatype.

$$P_{uv}^{QUAL>accuracy>dataType} = \begin{cases} pass, & \text{if } DT(P_{uv}^{VAL}) = V_v^{PROC>valueDomain>dataType} \\ fail, & \text{otherwise} \end{cases}$$

where $DT()$ is a function based on regular expression making possible the retrieval of the datatype of a given value.

DICE Quality Framework

The rule for checking whether a numeric value is within a defined interval can be denoted as follows:

$$P_{uv}^{QUAL>accuracy>syntax} = \begin{cases} pass, & \text{if } x \leq P_{uv}^{VAL} \leq y \\ fail, & \text{otherwise} \end{cases}$$

where x and y are the interval delimiters defined in $V_v^{PROC>valueDomain>syntax}$.

To evaluate whether a string value complies with a specified pattern (e.g. expressed in the form of regular expressions) the following rule applies:

$$P_{uv}^{QUAL>accuracy>syntax} = \begin{cases} pass, & \text{if } RE(P_{uv}^{VAL}, V_v^{PROC>valueDomain>syntax}) = true \\ fail, & \text{otherwise} \end{cases}$$

where $V_v^{PROC>valueDomain>syntax}$ defines a text pattern, and $RE()$ is a function which matches the property value against the text pattern.

Using this rule, **lack of value issues** can also be determined in case the properties have to fulfil a certain pattern (e.g. 4-digit postal codes, email addresses).

In some cases the **step width** of numeric values, i.e. of double values is of relevance. DICE applies the definition of (Wolfram Alpha 2017a) where step width (precision) is defined as the total number of significant decimal digits in a numeric value:

$$P_{uv}^{QUAL>accuracy>stepWidth} = \begin{cases} pass, & \text{if } -\log_{10} \frac{1}{|P_{uv}^{VAL}|} < V_v^{PROC>valueDomain>stepWidth} \\ fail, & \text{otherwise} \end{cases}$$

where $V_v^{PROC>accuracy>stepWidth}$ defines the required number of decimal digits.

Misspellings can be identified by comparing property values to entries in a dictionary.

$$P_{uv}^{QUAL>accuracy>misspelling} = \begin{cases} pass, & \text{if } DICT(P_{uv}^{VAL}) = 'yes' \\ fail, & \text{otherwise} \end{cases}$$

where $DICT()$ is a function which returns “yes” where P_{uv}^{VAL} matches an entry in the dictionary.

Data freshness is dependent on the actual age of a property (i.e. date when the data was collected or changed). This value has to be compared with the freshness requirements defined in $V_v^{PROC>currentnes>maximumAge}$. To determine whether a property value is outdated, the following expression has to be evaluated:

$$P_{uv}^{QUAL>currentnes>maximumAge} = \begin{cases} pass, & \text{if } AGE(P_{uv}^{LOG>changeDate}) < V_v^{PROC>currentnes>maximumAge} \\ fail, & \text{otherwise} \end{cases}$$

where $AGE()$ is the function which delivers the age in days.

DICE Quality Framework

Functional dependencies of variables may exist between properties of variables of the same dataset (variable dependencies) and between properties of variables that reference observable units (references). To evaluate whether stated dependency requirements between variables are fulfilled the following rule is applied:

$$P_{uv}^{QUAL>accuracy>variableDependency} = \begin{cases} pass, & \text{if } P_{uv}^{VAL} = FDF(V_v^{PROC>valueDomain>variableDependency}) \\ fail, & \text{otherwise} \end{cases}$$

where $FDF()$ is a function that implements the functional dependency rules defined in $V_v^{PROC>variableDependency}$.

Uniqueness value violations of properties arise in variables marked as “uniqueness required” where a property value is contained multiple times:

$$P_{uv}^{QUAL>uniqueness>redundancy} = \begin{cases} pass, & \text{if } V_v^{PROC>valueDomain>uniqueness} = false \vee |P_{uv}^{VAL} \in \{P_{iv}^{VAL} \mid i = 1, 2, \dots, U\}| = 1 \\ fail, & \text{otherwise} \end{cases}$$

Synonyms existence arises in variables carrying syntactically different property values with the same meaning, i.e. if there exists another property value within the variable which has the same meaning.

$$P_{uv}^{QUAL>consistency>synonym} = \begin{cases} pass, & \text{if } \exists x \in \{P_v^{VAL} \setminus P_{uv}^{VAL}\} : SF(x, P_{uv}^{VAL}) > \tau \\ fail, & \text{otherwise} \end{cases}$$

where $SF()$ is an ontology-based function to compute semantic similarity of a pair of property values, and τ is a defined threshold. For an overview of such similarity functions see (Lee et al. 2008).

Redundancy of an entity is given for an observable unit when the dataset contains another observable unit with the exact same property values.

$$U_u^{QUAL>uniqueness>redundancy} = \begin{cases} pass, & \text{if } |\{P_{uv}^{VAL}, v = 1..|V|\} \cap \{D\}| = 1 \\ fail, & \text{otherwise} \end{cases}$$

Inconsistency of an entity is given when the same observable unit exists, i.e. an observable unit with the same data key but with different property values:

$$U_u^{QUAL>uniqueness>inconsistency} = \begin{cases} pass, & \text{if } \exists U_z \in \{D \setminus U_u\} : \forall P_{uv} \in \{V_v \mid V_v^{PROC>key} = true\} : P_{uv}^{VAL} = P_{zv}^{VAL} \\ & \wedge U_u^{QUAL>uniqueness>redundancy} = fail \\ fail, & \text{otherwise} \end{cases}$$

DICE Quality Framework

Whether a property value is compliant with the defined **measure unit** or not is determined by comparison of the property's "measure unit" ($P^{PROC>valueDomain>measureUnit}$) with the required measure unit stated within the process metadata of the variable ($V^{PROC>valueDomain>measureUnit}$):

$$P_{uv}^{QUAL>accuracy>measureUnit} = \begin{cases} pass, & \text{if } P_{uv}^{PROC>valueDomain>measureUnit} = V_v^{PROC>valueDomain>measureUnit} \\ fail, & \text{otherwise} \end{cases}$$

Referential integrity violations arise where a variable of a dataset contains wrong references to observable units (usually, but not necessarily held in another dataset). Reference variables permit only those values that appear in the variable to which it refers. Note that in DICE the data key of a given dataset is defined by labeling one or more variables as key variables.

The same holds true for references. If the "foreign key" does not match a "primary key" in the target dataset (D'), a referential integrity violation arises.

$$P_{uv}^{QUAL>consistency>referentialIntegrity} = \begin{cases} pass, & \text{if } \neg \exists U_{u'} \in D': \{(v, v') \mid v \stackrel{\wedge}{=} v' \wedge P_{uv}^{VAL} = P_{u'v'}^{VAL}\} \\ fail, & \text{otherwise} \end{cases}$$

where D' is the referenced dataset, and v' is the referenced variable, and (v, v') is a set of variables, as the references are not necessarily defined by a pair of single variables.

Cardinalities define relationships between two tables. Basically, two types of relationships exist: one-to-many and many-to-many. Relationships between two tables are defined via foreign key reference. The variable(s) that compose the primary key value for one table are linked to another column in another table. Via cardinality constraints the number of links between observable units of the two tables can be restricted. In DICE, $V^{PROC>cardinality}$ defines such restrictions by defining intervals of the number of allowed relationships between observable units between two tables. A **cardinality violation** arises where the number of allowed relations is not within this defined interval. The quality indicator rule can be denoted as follows:

$$P_{uv}^{QUAL>consistency>cardinality} = \begin{cases} pass, & \text{if } x \leq \left| \{P_{uv}^{VAL} = P_{jv}^{VAL}, j = 1..|U|\} \right| \leq y \\ fail, & \text{otherwise} \end{cases}$$

where x and y are the cardinality delimiters defined in $V_v^{PROC>valueDomain>cardinality}$ and $x=0, y = \infty$ if undefined. In case of a 1:1 cardinality $x=1$ and $y=1$.

DICE Quality Framework

Outdated references are defined in the same way as outdated values, as references are represented the same way as properties with the only difference being that the same property has to be defined as a primary key in a variable of any other dataset.⁵

Circularity among tuples arises where the relationships k (pairs of foreign key and primary key) can be arranged in a cyclic sequence $(k_1, k_2, k_3, \dots, k_l)$ such that:

$$V_v^{QUAL>consistency>circularity} = \begin{cases} \text{pass, if } \neg \exists n : \{k_i, k_{i+1} | i = 1, \dots, n-1\} : k_n = k_i \\ \text{fail, otherwise} \end{cases}$$

The remaining quality indicators not discussed in the previous sections have to be assessed manually. Examples are **meaningless values** where non-conformance to the designated meaning of their variable description (semantics) cannot be assessed automatically.

The stated quality indicators clearly pose dependencies on each other. For example, a *missing value* cannot be *meaningless*, and a property with “data type violation” cannot be assessed in the context of “set violation”. DICE recognizes this fact by introducing the measurement category “not applicable (n.a.)” for quality indicators that do not apply. In addition to these dependencies not all quality indicators apply to all variables. Applicability of the stated quality indicators is dependent on the requirements defined in other V^{PROC} . For example, the indicator “set violation” is not suitable for evaluating string properties. From this fact, it follows that assessing the quality of a certain property is ideally done in a certain sequence. Similar to the approach of (Oliveira et al. 2005), DICE introduces an assessment algorithm based on a binary tree metaphor defined in the algorithm of Fig. 36.

```
input:
 $O^{(i)}$ 

output:
 $O^{(o)}$ 

begin
for  $v=1$  to  $|V|$  in  $V_v$  // for each variable in the dataset
```

⁵ Note that in section 7.1.1.2 the concept of edge tables is introduced, the typical relational concept to depict the references (pairs of primary and foreign key). In this context outdated references have to be evaluated per table row.

DICE Quality Framework

```

for  $u=1$  to  $|U|$ 
  check completeness of  $P_{uv}$  // check for missing value
  set  $P_{uv}^{QUAL>completeness>missingValue}$ 
  if missingValue = fail
    % in case of missing value none of the subsequent checks has
    % to be performed %
    set 'n.a.' for all indicators in  $P_{uv}^{QUAL}$ 
  else
    if  $P_{uv}^{QUAL}$  represents a value //not a relation
      check datatype, set  $P_{uv}^{QUAL>accuracy>dataType}$ 
      check semantics, set  $P_{uv}^{QUAL>accuracy>semantics}$ 
      check syntax, set  $P_{uv}^{QUAL>accuracy>syntax}$ 
      if datatype = double
        check syntax, set  $P_{uv}^{QUAL>accuracy>stepWidth}$ 
      endif
      check misspelling, set  $P_{uv}^{QUAL>accuracy>misspelling}$ 
      check data freshness, set  $P_{uv}^{QUAL>currentness>maximumAge}$ 
      check dependencies, set  $P_{uv}^{QUAL>accuracy>variableDependencies}$ 
      check redundancy, set  $P_{uv}^{QUAL>uniqueness>redundancy}$ 
      check synonyms existence, set  $P_{uv}^{QUAL>consistency>synonym}$ 
      check measureUnits compliance, set  $P_{uv}^{QUAL>accuracy>measureUnit}$ 
    elseif  $P_{uv}^{QUAL}$  represents a relation
      check referential integrity, set  $P_{uv}^{QUAL>consistency>referentialIntegrity}$ 
      check cardinality requirements, set  $P_{uv}^{QUAL>consistency>cardinality}$ 
      check outdated references, set  $P_{uv}^{QUAL>currentness>maximumAge}$ 
    endif
  endfor
  check circularity among obs. units, set  $V_v^{QUAL>consistency>circularity}$ 
for  $u=1$  to  $|U|$ 
  check uniqueness, set  $U_u^{QUAL>uniqueness>redundancy}$ 
  check uniqueness, set  $U_u^{QUAL>uniqueness>inconsistency}$ 
endfor

calculate summative indicators on property level
  
```


DICE Quality Framework

calculate summative indicators on variable level
calculate summative indicators on observable units level
calculate summative indicators on dataset level

Fig. 36 Determine quality indicators for properties

The total quality of a property is defined as the averaged quality category values of all quality categories (except the total quality indicator itself) and hence can be denoted as follows:

$$P_{uv}^{QUAL} = \frac{\sum_{qualityCategory} P_{uv}^{QUAL>qualityCategory}}{\left| \{P_{uv}^{QUAL>qualityCategory}\} \right|}$$

where the superscript *qualityCategory* indicates the quality value per category, which in turn is calculated from the averaged values of its subordinated quality indicator values:

$$P_{uv}^{QUAL>qualityCategory} = \frac{\left| \{P_{uv}^{QUAL>qualityCategory>indicator} \mid P_{uv}^{QUAL>qualityCategory>indicator} = pass\} \right|}{\left| \{P_{uv}^{QUAL>qualityCategory>indicator}\} \right|}$$

where the superscript *indicator* is the index representing an atomic quality indicator categorised into a given quality category.

Whereas all of the quality indicators of a quality category are considered of equal importance, corrective measures related to quality issues require different efforts. Take the following example: “missing values” have to be imputed (if expedient at all) considering additional sources or often complex imputation techniques. On the other hand, “representation inconsistencies” can often be resolved using simple reclassification transformations (see section 5.3.2.5). Whereas many of the identified quality issues relate to single property values, others refer to observable units, variables or entire datasets.

DICE defines summative quality measures based on averaged values of the quality indicators on property level, i.e. quality issues arising on property level are averaged onto variable and/or observable unit level and from there on dataset level.

In line with this, DICE offers a “data quality profile” for datasets. Based on this data quality profile, the data engineer can easily investigate the data quality and take corrective actions as needed. Moreover, as the quality measures are part of the composite data object, they can be used for data cleansing purposes, e.g. by selecting observable units of high quality only. Fig. 37 shows the dependencies between quality indicators on property level and summative quality indicators on level of observable units and variables.

DICE Quality Framework

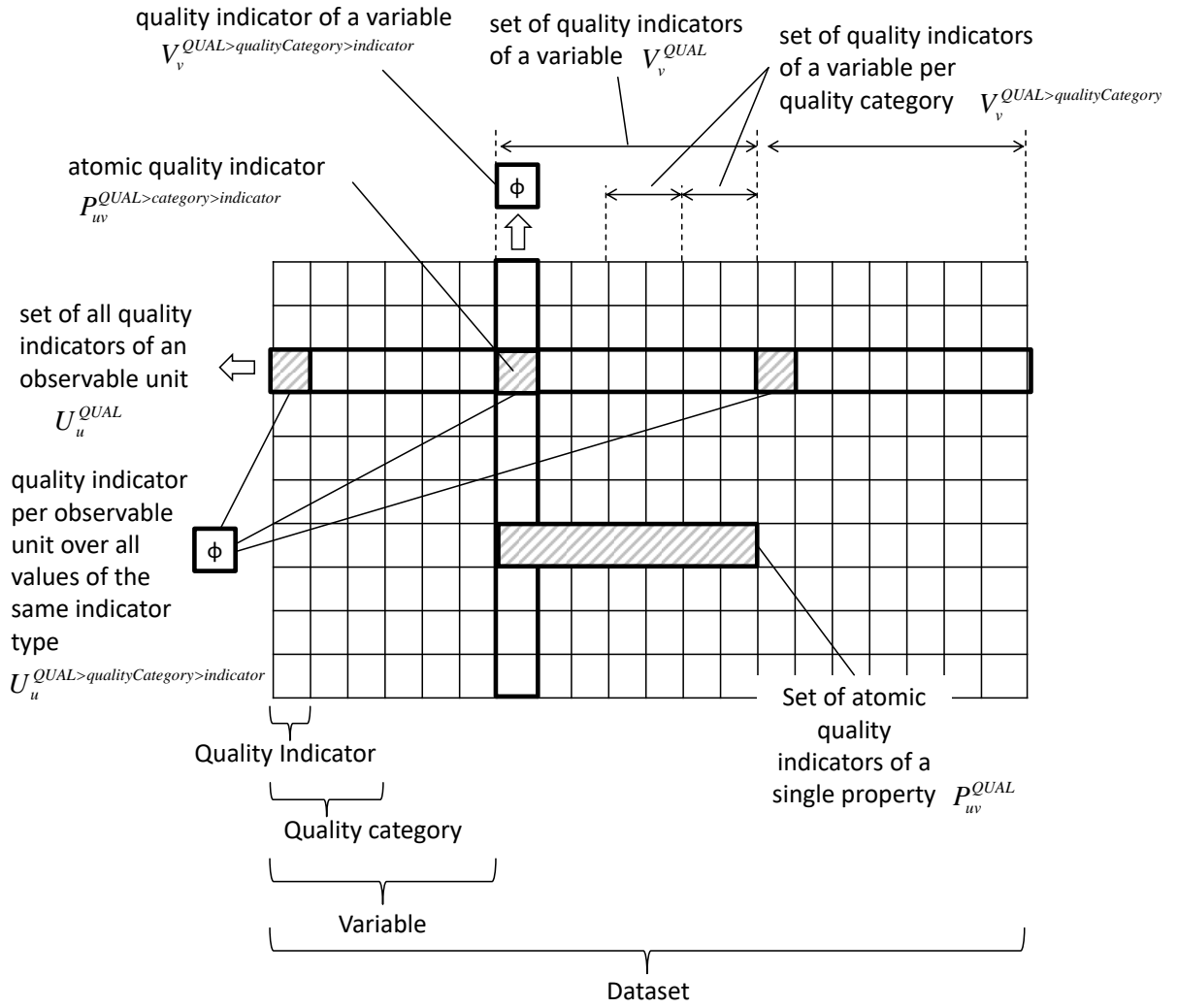


Fig. 37 Levels and structure of DICE quality indicators

For each variable the quality measures are calculated for the entire variable for its constituent quality categories and on lowest level per quality indicator.

The quality measure of a variable per atomic quality indicator is defined as follows:

$$V_v^{QUAL > qualityCategory > indicator} = \frac{\left| \left\{ P_{uv} \mid P_{uv}^{QUAL > qualityCategory > indicator} = pass \right\} \right|}{|U|}.$$

An example is $V_v^{QUAL > accuracy > syntax}$ which for example, shows the average fulfilment of the syntax requirement stated for the variable v over all observable units within a dataset.

Note that for reasons of readability, properties evaluated with ‘not applicable’ are not considered in this and the subsequent equations.

DICE Quality Framework

The total quality per quality category (e.g. completeness, accuracy and currentness) of a variable is calculated as the average of the indicator-based quality measures and can be formalised as follows:

$$V_v^{QUAL>qualityCategory} = \frac{\sum_v V_v^{QUAL>qualityCategory>indicator}}{\left| \left\{ V_v^{QUAL>qualityCategory>indicator} \right\} \right|}.$$

An example is $V_v^{QUAL>accuracy}$ which presents the averaged quality measures in the context of *syntax* compliance, *valueStep* compliance, etc.

The total quality of a given variable is calculated as the average of its category-based quality measures and can be formalised as follows:

$$V_v^{QUAL} = \frac{\sum_v V_v^{QUAL>qualityCategory}}{\left| \left\{ V_v^{QUAL>qualityCategory} \right\} \right|}.$$

Similar to variables, the averaged values for the observable units are calculated. The averaged quality measures per quality category of an observable unit are calculated as follows:

$$U_u^{QUAL>qualityCategory} = \frac{\sum_{uv} P_{uv}^{QUAL>qualityCategory}}{\left| \left\{ P_{uv}^{QUAL>qualityCategory} \right\} \right|}.$$

The averaged quality measures per variable of an observable unit are calculated as follows:

$$U_{uv}^{QUAL} = \frac{\sum_{uv} P_{uv}^{QUAL>qualityCategory}}{\left| \left\{ P_{uv}^{QUAL>qualityCategory} \right\} \right|}$$

where *qualityCategory* is the index for all possible quality categories and $\{P_{uv}^{QUAL>qualityCategory}\}$ is the set of all quality categories (comprising the atomic quality indicators) of one property within a given dataset.

The total quality of an observable unit is calculated as the average of all property quality indicators:

$$U_u^{QUAL} = \frac{\sum_{uv} P_{uv}^{QUAL}}{|V|}.$$

For the entire dataset the quality per atomic indicator of a variable is calculated as follows:

DICE Quality Framework

$$D^{QUAL>qualityCategory>indicator} = \frac{\sum_v V_v^{QUAL>qualityCategory>indicator}}{|V|}$$

where $V_v^{QUAL>qualityCategory>indicator}$ represents the average quality of a given variable per indicator.

Likewise, the quality of the dataset in the context of its quality categories is defined as follows:

$$D^{QUAL>qualityCategory} = \frac{\sum_v V_v^{QUAL>qualityCategory}}{|V|}$$

and the total quality of the entire dataset is calculated as the average of the total quality indicator of its variables:

$$D^{QUAL} = \frac{\sum_v V_v^{QUAL}}{|V|}.$$

6.3 Interpreting the Quality Profile

The data engineer uses the quality profile after performing a transformation task to evaluate the data quality of the output dataset. By investigating the total quality per variable ($V^{QUAL>total}$), problematic variables can easily be identified. By analysing the variable quality indicators ($V^{QUAL>indicator}$), the reasons for poor quality can be identified easily and quality improvement measures can be triggered. In many cases redesigning the DICE workflow will help to raise data quality, e.g. by adding additional transformation tasks or by changing the sequence of transformation tasks.

The same holds true for the quality measures of observable units. One strategy to improve data quality could be to simply remove observable units with bad data quality from the data object via selection transformation. Of course, the data engineer has to ensure that this transformation does not violate the coverage requirements ($D^{PROC>RequiredPopulationSize}$), i.e. that the selection leads to coverage of a population that is too small. In many cases improving data quality will require optimising the values within the given input datasets. For such cases (Kim et al. 2003) identify the following strategies:

DICE Quality Framework

- Intervention by domain expert, i.e. provision of additional high quality data or performing manual quality improvements on the input datasets.
- Use of a lookup-table, e.g. for regrouping categorical data.
- Use of abbreviation dictionaries for resolving abbreviations.
- Use of conversion algorithms for resolving representation differences.
- Use of encoding tables for resolving format issues, e.g. to convert from ASCII to Unicode.
- Recalculation of properties in the case of functional dependencies.
- Running a spell-checker to correct misspellings.
- Applying data imputation techniques in the case of missing values.

However, one must point out that in the vast majority of quality issues according to (Kim et al. 2003), in more than 75% of the stated cases intervention by a domain expert is the only alternative for raising data quality.

6.4 Summary

The introduced quality framework serves a two-fold purpose: (1) it supports unambiguous definition of the output data object ready to be used for upcoming DM modelling/evaluation phases, (2) it makes possible the determination of the data quality after each performed transformation task and thereby supports the definition of an adequate data transformation process.

The DICE data quality indicators have been deliberately derived from the ISO standard for “Software product Quality Requirements and Evaluation, Data Quality Model” (ISO/IEC/IEEE 25012 2008) and shaped by the extraction of typical quality issues from “dirty data taxonomies”. Quality indicators are assigned to the meta structure elements of DICE, making possible the evaluation of the data quality for entire datasets, for observable units and variables and on lowest level, for each of the properties within the dataset. The DICE quality framework follows the principles of situational method engineering, so that the framework can be enhanced with additional quality indicators and algorithms for their calculation.

7 Application of DICE in the Fields of EAA

As for any KDD endeavour, Enterprise Architecture Analytics requires a profound understanding of the available and required data. KDD frameworks such as CRISP-DM (Chapman et al. 2000) savour this requirement and propose data understanding/exploration as one of the first phases of KDD endeavours (see section 2.3). This of course also applies for EAA and is reflected in the DICE modelling procedure (see section 5.1). In this section an analysis of the typical structure of EA data is performed. Peculiarities of EA data are analysed and discussed. From the findings, requirements on method chunks to support EAA are derived. Section 7.2 draws on these requirements by discussing how these requirements can be satisfied by assembling and deriving new method chunks from the DICE meta structure to build a situational method for EAA data preparation.

For illustration purposes, the EA modelling language Archimate and the TOGAF content metamodel are used. Archimate is often quoted in the examined research papers and TOGAF can be considered as one of the most prominent EA frameworks (Schekkerman 2004b), (Moser, Fürstenau and Junginger 2010), (Urbaczewski and Mrdalj 2006). Where required, additional metamodels from EA frameworks such as from DODAF (DoD 2010) are used for illustration purposes.

7.1 Data Understanding - EAM-specific Requirements

EA relevant data comes in manifold flavours. To apply DICE on datasets residing in EA models, adequate transformation task types have to be derived from the DICE meta structure introduced in section 5. For this purpose, common EA data structures are analysed in the subsequent sections. Requirements of the DICE method base are collected. For the purpose of illustration business IT alignment is used, an often quoted problem in context of EAM, (see e.g. Winter and Fischer 2006), (Pereira and Sousa 2005), (Aier and Winter 2009) and (Wegmann 2002)). For a better understanding, the following guiding example is used to examine the different problem cases.

Guiding case: The case focuses on the interplay of applications and technologies (i.e. system software) both intended to serve business processes. The assumption is that due to the use of inappropriate technology, the application architecture of

Application of DICE in the Fields of EAA

an organisation is inefficient, leading to suboptimal business support in terms of functional business requirements, cost, flexibility and time to market. To address this problem, enterprise architects need to have a clear understanding of the applications and technologies in place. To detect problematic areas in the business architecture, the enterprise architects connect the applications and technologies to business processes of the organisations. In this way, enterprise architects are able to provide an overview of problematic business areas in regard to IT support and foster targeted budget allocation for IT investments. To illustrate the findings, the architecture team plans to use a heatmapped clustermap as sketched in in Fig. 38, as the model kind.

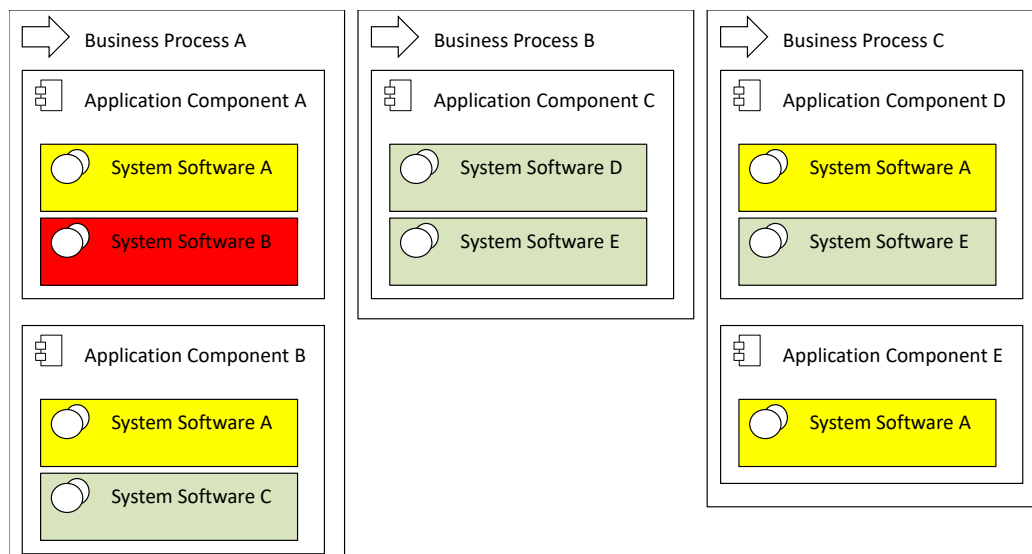


Fig. 38 Heatmapped Clustermap – Technology support of business processes

In the example, the colour code applied to the system software artefacts represents the individual technology fit. System software building blocks coloured in green represent high technological fit, yellow-coloured system software is problematic and system software in red is highly recommended to be decommissioned. Similar diagrams have been proposed by (Karagiannis, Moser and Mostashari 2012) and (Pouya Aleatrati Khosroshahi et al. 2015).

The example draws on the likely assumption that this is only one of the arbitrary scenarios addressed by the architecture team and that no encompassing EA repository containing the required data is in place. However, in the fictive organisation there are a multitude of EA descriptions available which might serve as a valuable input for the required analysis.

7.1.1 The object-oriented nature of EA data

Many of the EA frameworks use as a base the formal definitions of systems and software: ISO/IEC 42010:2007 Systems and software engineering, see TOGAF (The Open Group 2011), Archimate (The Open Group 2016), NATO (Handley and Smillie 2008), the TUM EA pattern catalogue (Pouya Aleatrati Khosroshahi et al. 2015) and many more.

Architecture descriptions based on this “schema” come in a variety of formats and contents. TOGAF for example defines the following major architecture artefacts types, i.e. model kinds (The Open Group 2011): (1) catalogues, (2) matrices and (3) diagrams. An overview has been provided in Fig. 8. In the following subsections, the most common data structures used to represent EA data are discussed. To this end, the typical representation formats used in EA analysis are examined using the research papers discussed in section 3.1.3 and architecture artefacts recommended by TOGAF and Archimate as a main input.

7.1.1.1 Catalogues

Catalogues are lists of building blocks. They have to be considered as the basis for any EA analysis, as they carry the atomic design elements of EA, namely the EA building blocks. The application portfolio catalogue introduced in Fig. 19 is a perfect example of such a catalogue. According to TOGAF, the purpose of this catalogue is to identify, categorize and maintain a list of all the application components used by the enterprise. TOGAF recommends maintaining catalogues of all relevant building blocks in a so-called architecture repository. They serve as the datasets on which to base any EA analysis. TOGAF suggests (not meant to be exhaustive) more than ten catalogues, examples are:

- Interface catalogue, providing information on interfaces between applications. Applications typically create, read, update and delete data via these interfaces from other applications.
- Technology portfolio catalogue, capturing the list of all technologies used to implement and run the applications and interfaces throughout the entire organization.
- Requirements catalogue, holding the list of all requirements generated in the course of architecture engagements.

Application of DICE in the Fields of EAA

The building blocks (in DICE terms, the observable units) within these lists are always derived from the content metamodel and in an ideal case, the catalogues cover the entire population of these observable units. The applications in the application portfolio catalogue can correspond to the modelling classes: *information system service*, *logical application component* and *physical application component* of TOGAF's content metamodel. In equivalence, Archimate offers the concept of *application components* and DoDAF offers the concept of *system* as the core elements of the application architecture/layer.

The variables within the catalogues capture the characteristics of the building blocks, e.g. characteristics such as operating costs, production date and planned decommission date. Thus, catalogues represent multivariate datasets such as the DICE standard format defined in section 4.1. Typical graphical representations of the catalogued building blocks are: bar charts, portfolio charts and box plots, all of them ways to graphically depict the property values of numerical or categorical variables of observable units held within a catalogue.

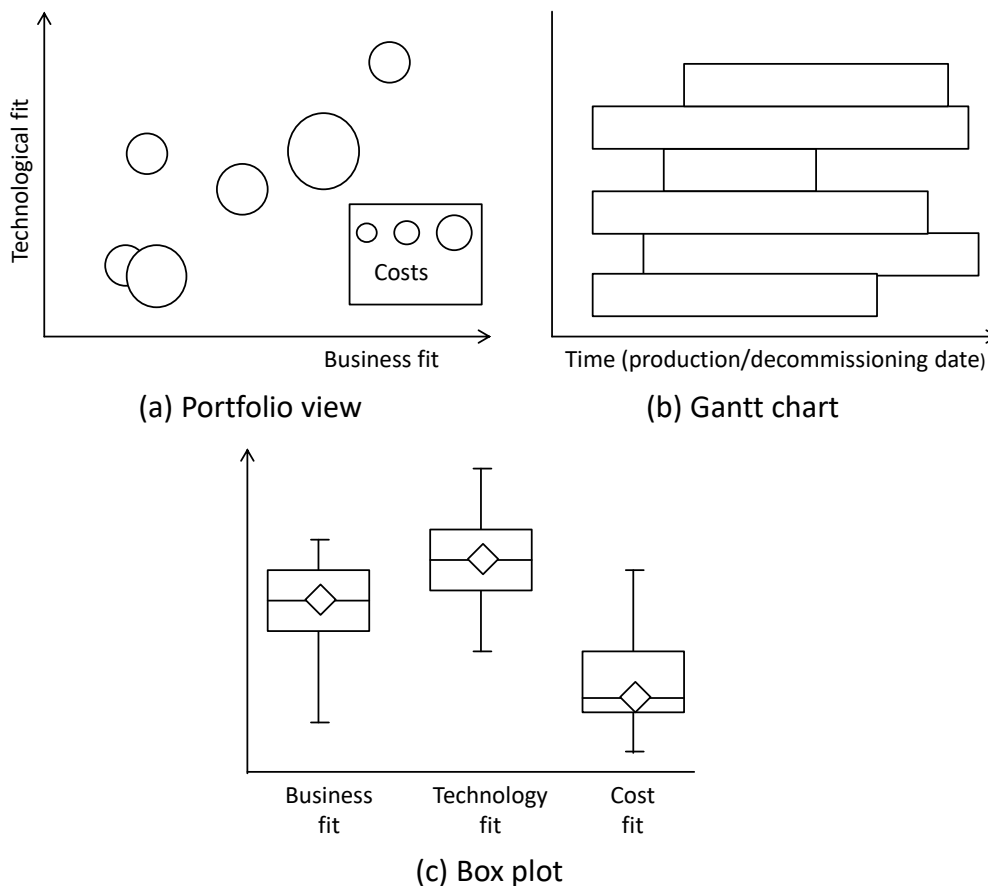


Fig. 39 Gantt, portfolio and box plot views on catalogues

Guiding case (continued): For the targeted architecture model, the enterprise architects need to gain an overview of the business processes, the application components as well as the technologies (i.e. system software) in use. For representing these datasets, catalogues such as the application portfolio catalogue need to be in place. The catalogues can be represented in the DICE standard data format, which is a horizontal layout or roughly speaking, a table. Let us assume that for all the three building block types corresponding datasets are available.

As the architects strive to assess the technology adequacy, additional information on technologies such as standard conformity, maintainability, operation costs etc. needs to be in place, serving as the atomic data for calculating the technology fitness. These atomic data elements, as far as they are available, are categorised into the set of variables of the technology dataset.

7.1.1.2 Matrices

Matrices represent the relationships between building blocks (the observable units) instantiated from the given metamodel. In the conducted literature review, matrices are explicitly mentioned in (The Open Group 2011), (Pouya Aleatrati Khosroshahi et al. 2015) and (Karagiannis, Moser and Mostashari 2012). Examples from TOGAF are:

- Application/Data Matrix, to depict the relationships between data elements and the application components that manipulate these data elements.
- Application/Organization Matrix, holding information about which organisational units use which application components.
- System/Technology Matrix, capturing the mapping of applications to technologies required for operating the applications.

The most basic form of representation is an adjacency matrix. The underlying data structure is a bipartite graph whose vertices are divided into two disjoint sets of building blocks. Formally this graph can be defined as $G=(\Omega_A, \Omega_B, R_{AB})$ where Ω_A, Ω_B define the sets of observable units (i.e. the population) instantiated from a given metamodel, and R_{AB} is the set of relations of a certain type between the building blocks. The values represented in the matrix cells indicate whether there is a relation between the building blocks or not.

Application of DICE in the Fields of EAA

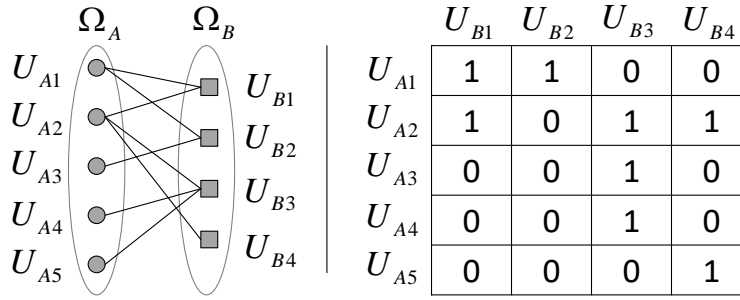


Fig. 40 The data structure of EA matrices

Analysing typical EA artefacts, one finds evidence of more complex matrix structures. Fig. 41 shows an example from (Moser et al. 2017).

















Application \ Standard	⌘ Authorisation Services	⌘ DBMS	⌘ ...
 Cash System			
 ControllIT			
 CRMconnect			
 ...			

Fig. 41 Heatmapped matrix (Moser et al. 2017)

The example shows a matrix based on populations (of the observable units: applications and standards) represented by the observable units on the axis and dependencies between them within the cells. The matrix is colour-coded, thus the relationships between the observable units are labelled, i.e. relations have properties and property values. The required data structure is in the form of Blaha's node-edge directed graph template (Blaha 2010) comprising two tables for presenting the catalogues of observable units and a third table, more specifically an edge table, specifying the relations between the observable units. Fig. 42 shows the structure in the form of a UML model and Fig. 43 provides an example.

Application of DICE in the Fields of EAA

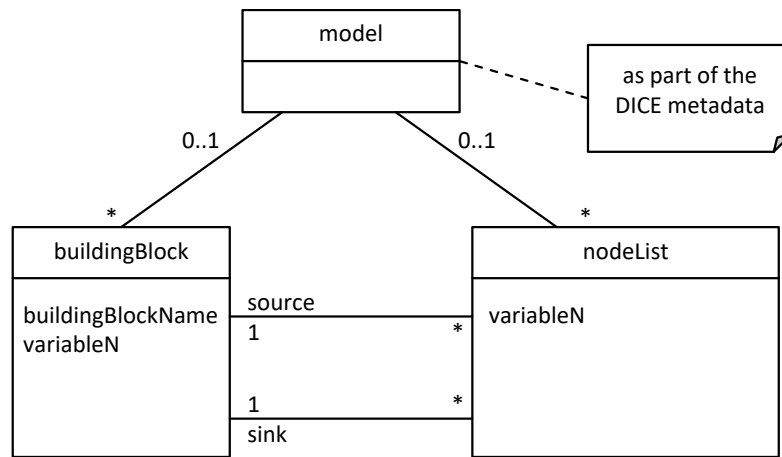
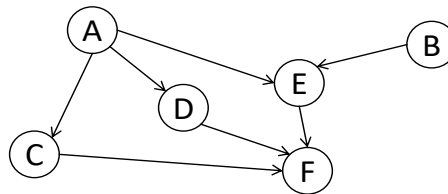


Fig. 42 Structure of node/edge directed graph template in UML, adapted from (Blaha 2010)



building BlockID	model ID	building Block Name	Relation ID	modelID	source Building BlockID	Sink Building BlockID
1	1	A	51	1	1	3
2	1	B	52	1	1	4
3	1	C	53	1	1	5
4	1	D	54	1	2	5
5	1	E	55	1	3	6
6	1	F	56	1	4	6
			57	1	5	6

Fig. 43 Node edge template, adapted from (Blaha 2010)

Guiding case (continued): It is obvious that for the envisioned clustermap model, information is required concerning the interplay between applications and technologies as well as use of applications in the context of business processes.

Let us assume that the IT operations team supplies the enterprise architects with a matrix presenting the technology usage of the applications. Fig. 44 presents this system/technology matrix where applications are represented on the x-axis and technologies are represented on the y-axis.

Application of DICE in the Fields of EAA

	Oracle 11g	Oracle 11g.R12	SQLserver 2016	Windows Server 2014	...
Document Management System	1	0	0	0	...
CRM System	1	0	0	0	...
Home and Away Financial Application	0	1	0	0	...
Home and Away Policy Administration	0	0	1	0	...
...

Fig. 44 Example system/technology matrix

As the application portfolio catalogue and the system/technology matrix stem from different sources, it is unlikely that the applications contained in both datasets will fully match. However, the system/technology matrix can easily be converted into a node-table representing a dataset in conformity with DICE.

Let us furthermore assume that a similar table matching business processes with application components is not available. Thus, the enterprise architects have to look for additional sources to extract the missing mapping between applications and business processes...

7.1.1.3 Diagrams

The third category of EA artifacts of TOGAF is denoted as diagrams. TOGAF vaguely defines diagrams as a means to graphically “*present building blocks plus their relationships and interconnections in a graphical way that supports effective stakeholder communication*” (The Open Group 2011). This definition corresponds to the concept of model defined in (ISO/IEC/IEEE 42010 2011), see section 2.2.

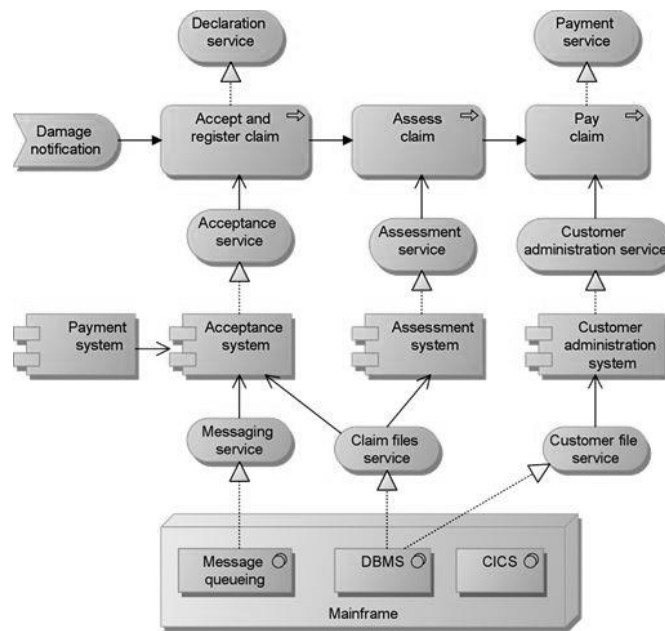
Examining TOGAF, one mainly finds evidence of two types of artifacts:

- nested box diagrams and
- node/edge diagrams.

Models of this type are common in almost all of the studied EA research papers. Archimate uses node-edge diagram as its main model kind. The majority of investigated EA research papers, e.g. (Ekstedt et al. 2009), (Johnson, Nordström and Lagerström 2007), (Niemann

Application of DICE in the Fields of EAA

2006), (Florez, Sánchez and Villalobos 2016) and (Manzur et al. 2015)) use node-edge diagrams for visualising the EA. Nested-box diagrams are not that common but are also used in some of the analysed EA research papers, see e.g. (Pouya Aleatrati Khosroshahi et al. 2015), (Karagiannis, Moser and Mostashari 2012) and (The Open Group 2016). Fig. 45 exemplarily depicts examples of the two artefact types.



(a) Node-edge diagram



(b) Nested-box diagram

Fig. 45 Nested box (The Open Group 2016) and node/edge diagrams (The Open Group 2011)

Application of DICE in the Fields of EAA

Enterprise Architecture (EA) models can be thought of as structured, object-oriented descriptions that usually correspond to an organisation-specific, i.e. customized metamodel that evolves over time due to ever changing organisational requirements and problem fields (Roth and Matthes 2014). The metamodels described in EA standards and frameworks, comprise varying levels of formality. See for example: TOGAFs Content Metamodel, the DODAF metamodel, the I-patterns of TUM's pattern catalog and many of the revisited EA publications in section 3. Whereas Archimate and DoDAF provide concise definitions of their metamodels, in TOGAF the metamodel is only roughly sketched. Archimate goes one step further and defines a modelling language on top of its metamodel (The Open Group 2016) including concepts such as viewpoints, model types, graphical notations for their modelling classes, etc.

Building blocks instantiated from EA metamodels are typically complex, viz. they are units of components that themselves may be put together from subordinated building blocks. Thus, building blocks have to be considered either as containers or atomic objects. The typical course of action in design modelling reinforces the need for such a structure: where typically the design elements (building blocks in EA) tend to be less explicit and more abstract in the early design phases, additional building blocks and descriptive data are added to the model as the design evolves. Analogous data structures can be found in construction planning where the design objects are seen as “*a convenient aggregation of information describing real world concepts*” (Ahmed and Navathe 1991). Building blocks such as business processes, application components and technology components describe these real world concepts, i.e. the main structure elements of an organisation.

Ehrig et al. state in (Ehrig et al. 2005) that any metamodel can be represented in the form of a labelled, directed and finite graph. From this, it follows that EA data can be represented in the form of labelled, directed graphs. Based on (van Buuren et al. 2004), a metamodel is described by a signature $M = (C, T, R, A)$ where:

- C is a finite set of modelling classes.
- T is a finite set of relation types.
- $R \in C \times C \times T$ is a finite set of *relations* between the modelling classes. Thus, an element $(c_1, c_2, t) \in R$ represents a relation type and expresses the fact that a relation of type t exists from concept c_1 to c_2 .

Application of DICE in the Fields of EAA

- A is a finite set of attributes assigned to the modelling classes and to the relation classes.

On data level (respectively on model level), the EA data can be represented as a set of models following the above metamodel definitions. (van Buuren et al. 2004) defined a model (X) by a 4-tuple $X = (O, T', F, A')$ where:

- O is a set of objects, respectively the building blocks.
- T' is a set of relations (of the relation types that are defined in the metamodel), in DICE represented in the form of 3-degree edge lists comprising the node pairs and an additional variable for denoting the relation type.
- $F : O \mapsto C \wedge T' \mapsto T$ is a function that maps objects to metamodel concepts and relations to relation types. In DICE terms, this mapping corresponds to the structure of the composite data object where observable units and variables are defined by their associated metadata.
- A' is the set of attributes available for an object (defined in the metamodel). In DICE the attributes are named variables (the equivalent term used in the domain of statistical metadata management).

In this vein, a model X must be understood as a multi-attributed directed graph. In the following sections, the identified EA analysis mechanisms are revisited and their underlying data structures are examined. At the outset, typical EA visualization patterns are discussed.

For representing a model graph in DICE, Blaha's node edge directed graph template (Blaha 2010), as exemplarily depicted in Fig. 43 appears to be sufficient.

Guiding case (continued): From previous architecture work a lot of architecture models in Archimate language exist. Many of the models are instantiated from Archimate's *application usage viewpoint* comprising information on business processes and application components. Fig. 46 shows an example.

Application of DICE in the Fields of EAA

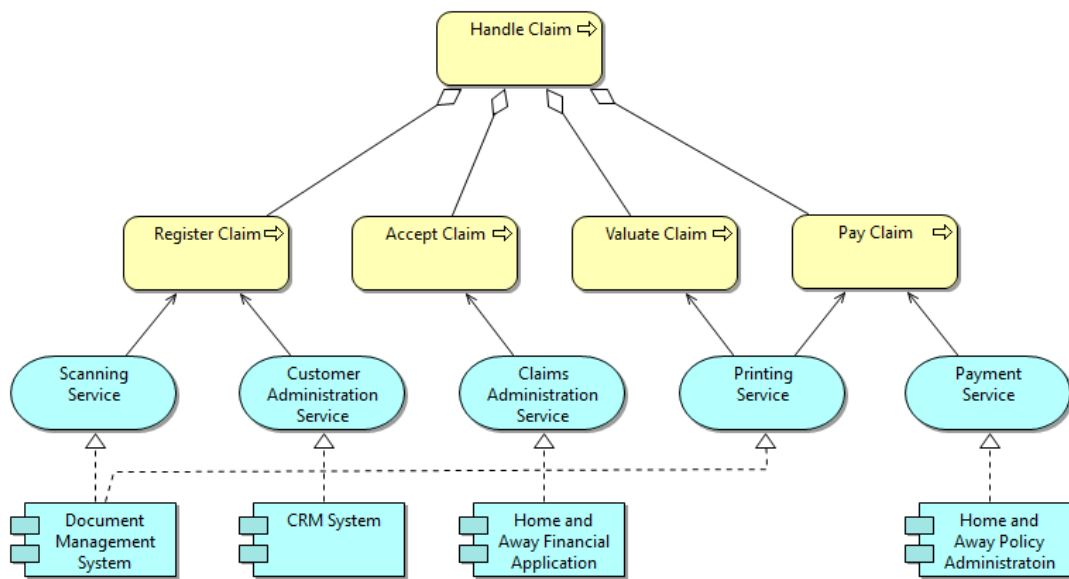


Fig. 46 Application Usage Viewpoint (The Open Group 2016)

By restructuring the input models the enterprise architects create the following data structure.

building BlockID	modelID	building Block Name	modelling class
1	1	Handle Claim	business process
2	1	Register Claim	business process
3	1	Accept Claim	business process
4	1	Valuate Claim	business process
5	1	Pay Claim	business process
6	1	Scanning Service	application service
7	1	Customer Administration Service	application service
...

relationID	modelID	source Building BlockID	sink Building BlockID	relation class
1	1	1	2	aggregation
2	1	1	3	aggregation
3	1	1	4	aggregation
4	1	1	5	aggregation
5	1	6	2	aggregation
6	1	7	2	serves
7	1	8	3	serves
...

Fig. 47 Node edge representation of application usage models⁶

It is obvious that the intermediate business service layer (comprising the services “Scanning Service”, “Customer Administration Service” etc.) is not required for the given problem statement and can be removed. This issue is addressed in section 7.1.4.2 by introducing a transformation task type for bypassing EA layers.

7.1.1.4 Data Structures for EA Analysis based on Probabilistic Relation Models

Numerous application scenarios for EA analysis based on probabilistic relation models (PRM) have been studied and presented by the research group “Industrial Information and Control Systems” of the “KTH Royal Institute of Technology”. Their PRM-based approach requires extensions to the given definition of metamodel in section 3.1.3: it requires a capability to express dependencies between attributes (Buckl et al. 2011). Fig. 48 provides an example. Note that not only the EA concepts but also single attributes (in the example the attribute “availability”) are interlinked.

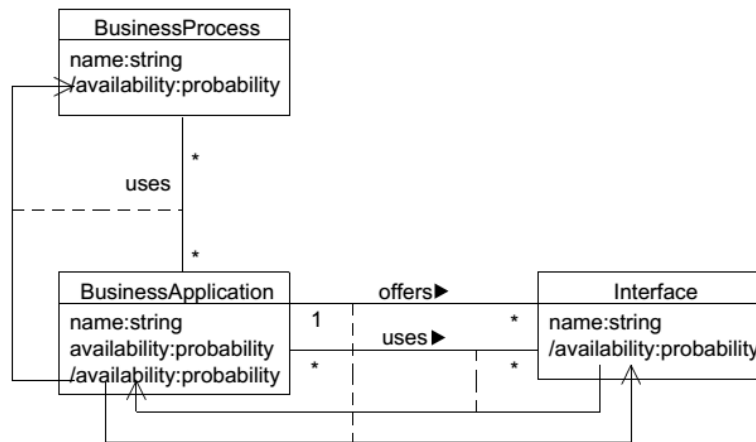


Fig. 48 Exemplary meta model snippet supporting PRM analysis (Buckl et al. 2011)

Therefore, the given metamodel definition and the underlying data structure defined in section 7.1.1.3 has to be extended to support PRM based-analysis. The following definition is added:

⁶ Note that in the given example data and metadata might be considered mashed, as the information on modelling class and relation types can likewise be seen as data and metadata.

Application of DICE in the Fields of EAA

$$R \in C \times C \times A$$

where the finite set of *relations* is extended by the set of relations connecting properties of modelling classes.

(Friedman et al. 1999) who specify a relational schema for PRM analysis in their work on “Learning Probability Models”, consider node edge templates as sufficient for storing the data. Importantly it must be noted that in this case, the rows of the edge table do not represent the relations between the building blocks but rather relations between attributes of the same type of the building blocks.

7.1.1.5 Multi-criteria Decision Making Methods

In essence, this type of EA analysis requires a structure to depict models in the form of a weighed tree, to whose nodes and edges, labels are assigned. Each of the nodes represents a quality criterion (as opposed to building blocks) that can be categorized into sub criteria. The topmost node represents an architecture scenario that is being evaluated. The leafs of such a tree are assessed by an expert, and based on the edge weights, for each node within the tree, its total quality is calculated until reaching the topmost node. Typically multiple scenarios (each represented by such a tree structure) are assessed by different experts and finally have to be consolidated into one resulting tree. Again, the *node edge directed graph template* 7.1.1.3 appears to be most efficient for storing the required data.

The modelling of concrete building blocks and relations is not required for this approach. However, use of EA models is suggested to depict the different scenarios. With its plateau concept, Archimate makes it possible to represent various EA scenarios within one model. From that, one can conclude that the structure for EA data also has to cope with optional EA scenarios. In section 7.1.7 this problem area is discussed in more detail.

7.1.1.6 The Data Structure for Indicator-based EA Analysis

Indicator-based EA analysis requires a well-structured dataset. Approaches such as presented in (Brückmann et al. 2009), (Addicks and Appelrath 2010), (Vasconcelos, Sousa and Tribolet 2015) pose requirements on their underlying datasets with a strong focus on the metamodels. All of the examined research papers define a concise metamodel for calculating the indicators. Although not providing how to ensure data quality, it appears obvious that for

Application of DICE in the Fields of EAA

calculating the EA indicators completeness on observable units, property values and relationships are required. An example of such a metamodel obtained from (Vasconcelos, Sousa and Tribolet 2007) is shown in Fig. 49.

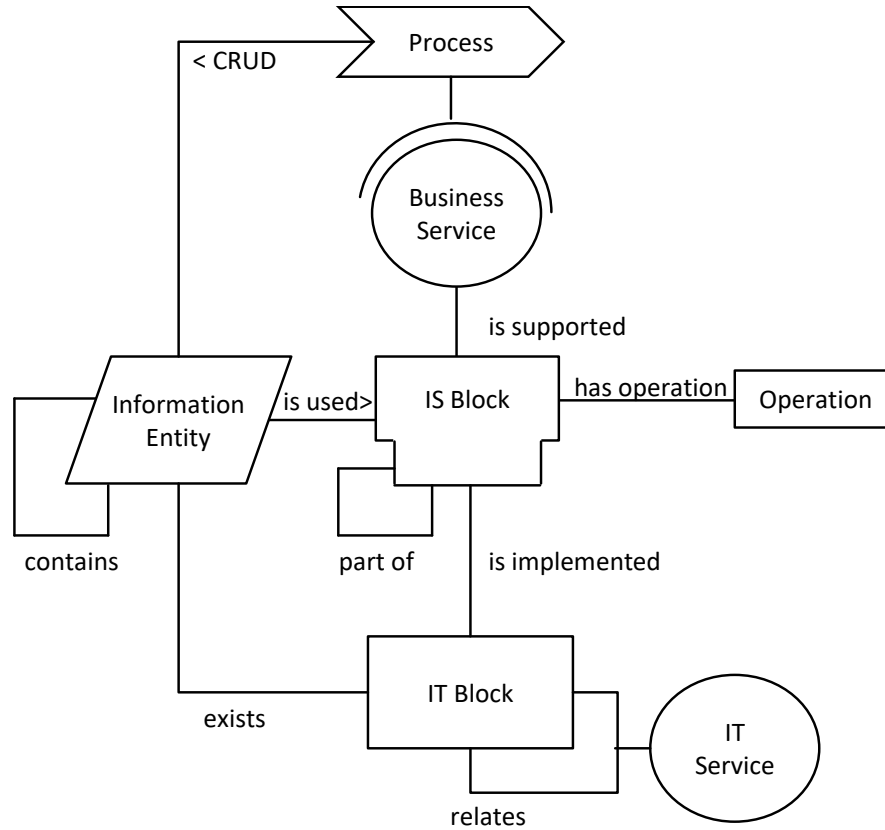


Fig. 49 Exemplary metamodel for an indicator-based approach (Vasconcelos, Sousa and Tribolet 2015)

Based on data structured in conformity with the given metamodel, the authors calculate indicators, such as *average number of possible operating systems*, *average number of different implementations of an information entity* and *average number of used security components*.

The required data structure corresponds to the definition of an EA metamodel provided in section 7.1.1.3.

Guiding case (continued): In conformity with (Vasconcelos, Sousa and Tribolet 2015), the enterprise architects might be interested in the “average response of a business process”, which can be computed by averaging the number of applications supporting a business process. Assumed that the hitherto compelled datasets are complete, this indicator can easily be calculated.

Suppose that additionally, the system indicator “average number of used security components” is one of the indicators used to calculate the technology fitness of the applications and thus will be required to calculate the colour coding of the envisioned heatmap. In accordance with (Vasconcelos, Sousa and Tribolet 2015), the indicator requires all technologies in use to run an application as an input. Security relevant technology components have to be flagged accordingly.

From this fact, the requirement is drawn that the technology component catalogue has to be extended with a variable, “Security relevant”. As this information is not available so far, the enterprise architects have to find ways to deal with this “missing value”...

7.1.1.7 EA Analysis Based on Network Measurements

This type of EA analysis considers the EA as a complex network. In their literature review (Santana, Fischbach and Moura 2016) focus strongly on EA analysis based on structural aspects of EA models. Typical measurements from graph theory are: central measurements (determining the prominence of nodes in a network), such as degree centrality, closeness centrality, betweenness centrality and eigenvector centrality, see the work of (Schoonjans 2016) who cites (Faust 1997) and (Rusinowska et al. 2011) in this context as a theoretical foundation.

The most efficient data structure for this graph-based analysis is organized in the node and edge undirected graph template (Blaha 2010). In graph theory, node edge tables are denoted as incident lists representing a list of edges incident to nodes. Fig. 50 exemplarily shows the required structure. Note that no attributes for edges and nodes are required and that the edges are not typed. However, without any informational loss, the node edge directed graph template introduced in section 7.1.1.3 serves the purpose as well.

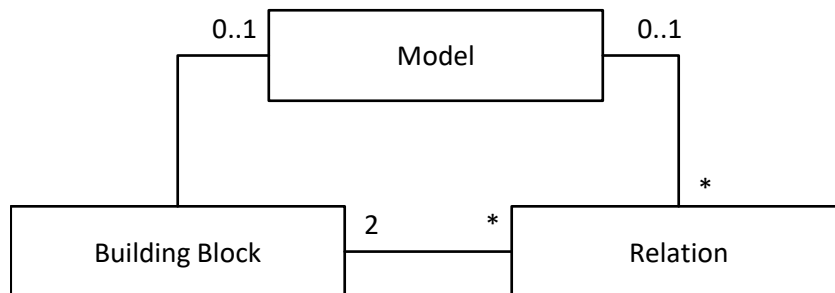


Fig. 50 Node edge undirected graph template, adapted from (Blaha 2010)

Application of DICE in the Fields of EAA

From the explanations in the above sections, it becomes clear that the node-edge directed graph template is ideal for structuring EA data for subsequent analysis.

To summarize, the node-edge directed graph template is suitable for most of the EA analysis approaches. DICE requires means to restructure given input datasets into this format.

Requirement 1: Provide method chunks to support the restructuring of EA data structures according to the node-edge directed graph template.

7.1.2 Amount of Data

Typically in EAA scenarios, there will not be millions of records as in the often-quoted BA and DM applications scenarios operating on census data and public health data files. Even in major enterprises the set of relevant EA models will not contain millions of building blocks. This fact is based on an evaluation of five EA repositories of organizations from the public administration, the financial sector and from the logistics sector. All of these organisations run EA endeavours at minimum for five years. The largest repository contained approximately 517.000 architecture artefacts, where it must be concluded that this repository to a great extent held detailed system parameters that are not considered as the typical EA building blocks.

If one takes the SAP reference model (Daneva 2004) with the rather “small” amount of 20.000 different process tasks dispersed throughout various industrial sectors (Mendling, Reijers and Recker 2010) as an example, it becomes evident that the population of EA building blocks of an organisation will not go into the millions.

This is of course in contrast to the operational business analytics data gathered to extend the EA models (see section 7.1.5.1).

7.1.3 Issues with Structural and Technical Formats

Any relevant input ideally is converted into the horizontal (tabular) layout for further processing. EA input data will typically come in various technical formats. Typical exchange formats are: spreadsheet formats (such as csv, xlsx), formatted text-files and xml-based formats. Some EA frameworks such as Archimate and DoDAF offer tool-agnostic exchange formats. Both Archimates' Model Exchange File Format (The Open Group 2015) and

Application of DICE in the Fields of EAA

DoDAF's PES (DoD 2010) provide well-documented XML schemas which make possible the exchange of EA models.

Clearly, EA management does not exclusively rely on modelled information sources. Information will also come from external sources, which have to be pre-processed and integrated with the EA model base for further use. The external sources might be used to enrich and validate the existing EA data. Take the following common examples:

- Service Level Agreements (SLAs) and underpinning contracts will often be available in the form of text-heavy documents only. In some cases they might be based on an agreed template. In other cases no template may be used at all. However, typically such documents will carry EA-relevant information about IT services and applications.
- Plain text business process documentations can be considered as an important part of the business architecture descriptions.
- Technology lifecycle data and technology ratings might be obtained from vendor websites or from sources such as www.technopedia.com, a platform providing a taxonomy of technologies and technology descriptions. Another valuable external source are reference models published in HTML and other formats (see e.g. the BIAN service catalogue, www.bian.org).
- Customer sentiment data and market trends extracted from social media platforms are among the classical sources for BA.

Guiding case (continued): So far the enterprise architects identified a couple of datasets relevant for the endeavour. Table 8 offers an overview of the data sources and their structural and technical formats.

Table 8 Exemplary sources for EA data

Source	Structural format	Technical format
Application portfolio catalogue	Horizontal layout, One row per application, One column per variable	Microsoft Excel
Application x Technology Matrix	Structured as adjacency matrix	CSV (comma separated value format)
Archimate models	Models in conformity with the Archimate metamodel (more precisely: in the form of	XML structure based on the Archimate model exchange file format

Application of DICE in the Fields of EAA

	"Application Usage Viewpoint" models)	
Set of security relevant software technologies	Not available	Not available

From this, one can conclude that DICE has to cope with manifold structural and technical formats. Means to load and interpret technical formats will be highly dependent on the capabilities of the data mining tool in use.

Requirement 2: Provide method chunks to load EA data from a diverse set of technical formats and sources.

7.1.4 Granularity, Generality and Abstractness of EA Data

Architecture design processes are typically of iterative and exploratory nature. Take TOGAF's ADM as one typical design approach for enterprise architectures. In the first phase (architecture vision), architectures are roughly designed. In the successive phases, (business architecture, information system architecture and technology architecture) the architectures are refined. Even later, in the phase "opportunities and solutions", the architecture is detailed into a solution architecture. While running through the different phases, arbitrary models might be created. The models in the later phases are typically more detailed as compared to those in the previous phases.

To foster this progressive approach, e.g. Archimate and DoDAF provide relationship types within their metamodels that make possible the documentation and analysis of architectures on different levels of detail. Aggregation (part-whole relation), composition (whole-part relation), generalisation (IS-A) and specialisation (whole-part) relationships are offered to facilitate the description the EA at different abstractions and level of detail. For EA analysis, this is clearly impeding. Transformation tasks have to be in place to "normalise" the EA data as needed. In the following sections archetypical EA structures in this context are discussed.

7.1.4.1 Reflexive Relations

In the case of reflexive relations, the building blocks of the same modelling class are related to each other. Often the building blocks are (hierarchically) decomposed into building blocks of their own type. In other words, the building blocks are participants in a parent-child relationship. To reduce complexity and gain an overview, lower level building blocks can be condensed into higher levels. Archimate's metamodel defines aggregation, composition and specialisation relationships being permitted between building blocks of the same modelling class. Each building block in an Archimate model can be refined into building blocks of its own modelling class using one of these relations, and conversely, complexity can be reduced by condensing the building blocks to higher layer building blocks. Reflexive relations of type association do not indicate different layers. Usually (but not mandatorily) building blocks of the same level of detail are connected via relationships of type "association".

Fig. 51 shows an example of reflexive relations modelled in Archimate.

Guiding example (continued): Looking closer at the given data structure extracted from the application usage viewpoint in Fig. 46, the enterprise architects detect that the business processes "Register Claim", "Accept Claim", "Valuate Claim" and "Pay Claim" are grouped into a super-ordinated business process, "Handle Claim". Fig. 51 depicts the situation in an alternative Archimate compliant representation.

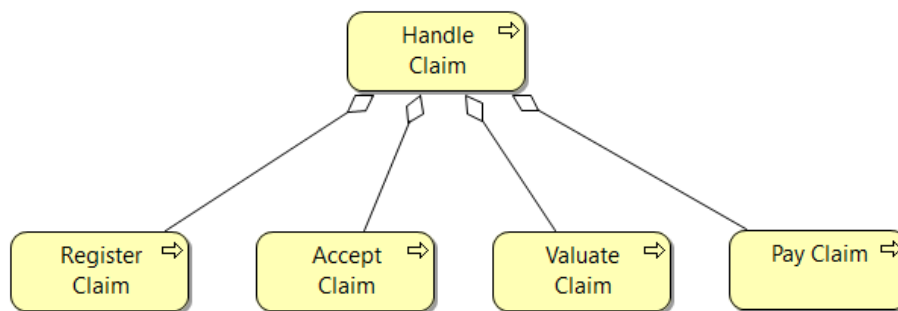


Fig. 51 Reflexive relations – example in Archimate notation

The enterprise architects have to define the required level of detail for the envisioned clustermap. They decide: either using level 1 or level 2 of the process architecture or alternatively, to extend the initial clustermap layout with another level. The alternative represents the business processes in a two-level hierarchy visualised in a box-in-a-box layout.

Application of DICE in the Fields of EAA

To avoid biased results in subsequent analysis phases it is essential to clearly identify these hierarchies and to “normalise” them where required.

Requirement 3: Provide method chunks to filter/consolidate building blocks organised in reflexive relations.

7.1.4.2 Hierarchical Decomposition

According to Fischer et al. (Fisher et al. 2014), the majority of concepts defined within an EA metamodel can be interpreted as aggregation hierarchies. Consequently, architecture building blocks are typically decomposed into more specific building blocks. The multi-level systems theory differentiates between strata, layers and echelons (Fischer and Winter 2007). These concepts have been addressed in the context of EA in various publications, see e.g. (Abraham, Tribolet and Winter 2013), (Korhonen, Yildiz and Mykkanen 2009) and (Gale and Eldred 1996).

Examples of stratified hierarchies are: the refinement of strategies into objectives/goals and the decomposition of applications into their software components etc. These hierarchies are typically established via relation classes providing inheritance and containment mechanisms (Sprinkle et al. 2010) such as the aforementioned relation types (aggregation, composition, etc.). Perfect examples of such relationships can be found in almost all EA metamodels. Examples from the TOGAF content metamodel are: relationships such as “decomposes” for the composition of application and technology components or the “is realised through” relationship which decomposes goals into objectives. The case of *reflexive relations* discussed in section 7.1.4.1 must be understood as a special case of stratified composition where the hierarchically ordered concepts are of the same type, i.e. of the same modelling class.

The concept of *layers* refers to the typical architecture layers, such as TOGAF’s business architecture, information systems architecture and technology architecture or Archimate’s business, application and technology layer. In keeping with the hierarchical multi-level systems theory, conditions and definitions of a superordinate EA hierarchy layer reduce the degrees of freedom of the downstream layers (Torokhti and Howlett 2000), i.e. “business follows IT”. Echelons refer to responsibilities and decision-making structures in the context of such stratified structures.

Application of DICE in the Fields of EAA

EA is typically interested in the top level strata and in the dependencies between its elements.

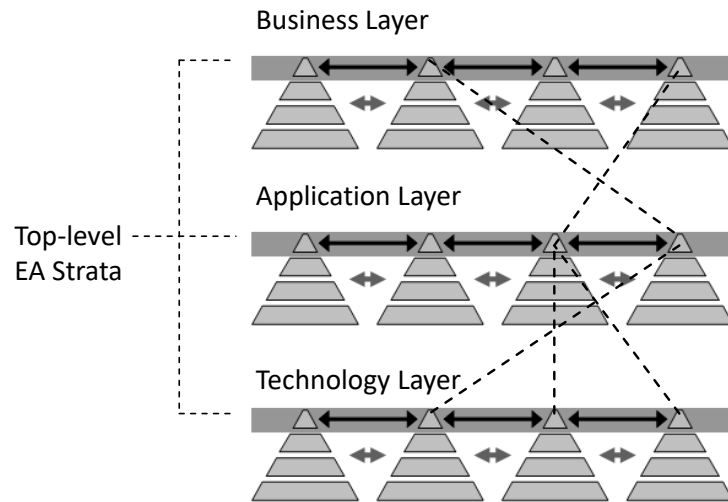


Fig. 52 EA strata, adapted from (Fischer and Winter 2007)

One significant impact on DICE results from the different modelling classes involved in hierarchical decomposition, as the different modelling classes typically will carry different attributes, and merging objects on different levels will not be that straight forward as in the case of reflexive relations.

Guiding example (continued): The provided architecture usage diagrams contain application services which implement (in Archimate terms “realise”) application services. Although the relation class “realise” is not classified as superclass-class relationship, the enterprise architects decide to apply a contraction transformation to these objects, thus, the objects are deleted from the dataset and the former relations are restored by directly connecting business processes to applications. Fig. 52 shows the resulting model.

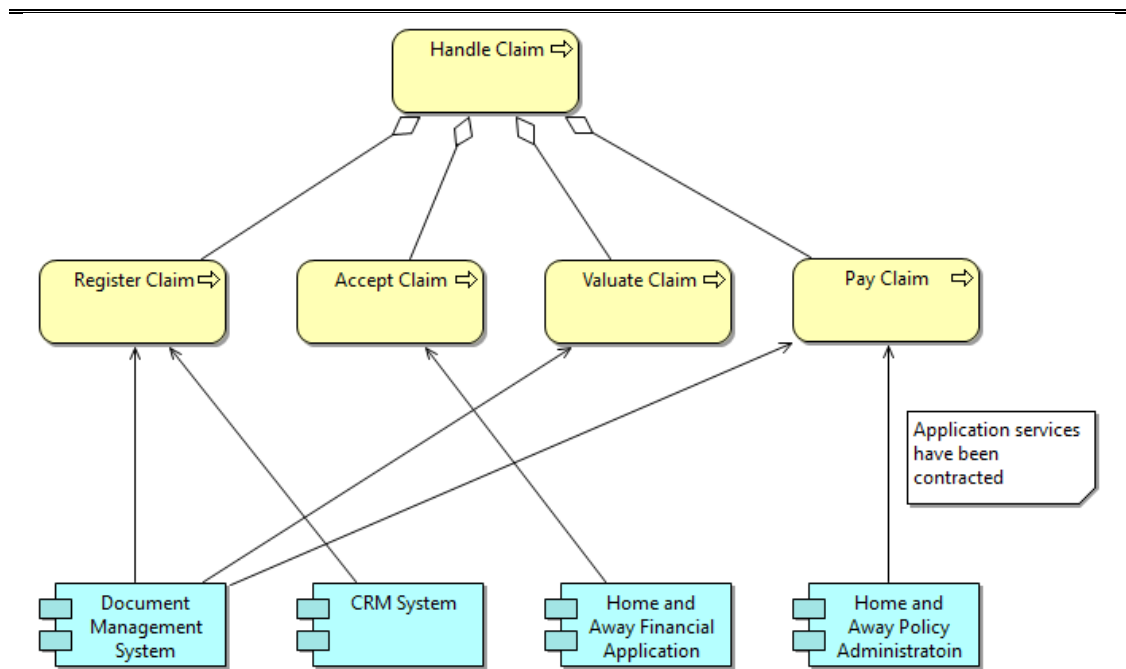


Fig. 53 Application usage model with contracted application services

From the above explanations, the requirement to “bypass” intermediary building blocks on the path between a pair of building blocks is drawn. In this connection possible problems of *cyclic references* (Beyer, Noack and Lewerentz 2005) have to be tackled where changes to one building block might influence an entire cycle of building blocks and *degenerate inheritance* (Beyer, Noack and Lewerentz 2005) or where a building block might inherit (e.g. via groupby transformation) values from another building block.

Requirement 4: Provide method chunks to filter/consolidate structures organised into different strata.

7.1.5 Logical versus Physical Layers

EA descriptions often comprise logical and physical building blocks. Whereas logical views are intended to offer an implementation-independent view of the architecture, the physical view is intended to represent the implementation details of the logical view. In this vein, the physical view describes the real world entities.

Application of DICE in the Fields of EAA

Consider the following example: a technology component might be characterised independently of any specific vendor or technology solution that represents the logical view; it focuses on functional aspects of the EA. In TOGAF, these logical building blocks are referred to as “architecture building blocks” (ABBs).

In contrast, the physical view describes so-called solution building blocks. For a better understanding Fig. 54 shows a few examples.

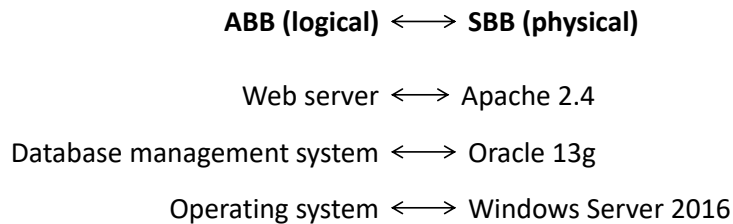


Fig. 54 ABBs and SBBs

Whereas the logical building blocks are used to describe the envisioned “ideal” architecture, the physical building blocks describe the actual implementation, i.e. the solution architecture (The Open Group 2011).

Guiding example (continued): By comparing the application components within the application portfolio catalogue (Fig. 19) and the application components documented within the architecture usage model (Fig. 46), the difference becomes obvious. Whereas the application portfolio catalogue contains physical entities, e.g. such as a concrete version of SharePoint, the architecture usage models comprise solely logical application components, such as the “Home and Away Financial Application”. This situation will raise complexity for the given endeavour, as simple merge transformations will not be sufficient. Additionally, the situation stresses the importance of unambiguous specification of the observable units, i.e. the need for semantic metadata.

Some enterprise architecture frameworks differentiate between logical and physical building blocks in their metamodels. Evidence can be found in TOGAF’s content metamodel, which offers dedicated modelling classes for ABBs and SBBs. The differentiation between “logical application component” and “physical application component” is one example. Notably, this differentiation in TOGAF is not made for building blocks of the business architecture. Archimate’s approach for dealing with the progression from abstract to concrete building

Application of DICE in the Fields of EAA

blocks mainly focuses on the refinement of building blocks via the relationship type “realisation”: “*The realization relationship indicates that more abstract entities (‘what’ or ‘logical’) are realized by means of more tangible entities (‘how’ or ‘physical’)*“, (The Open Group 2016).

Additionally, for describing the physical representations of application components and data objects Archimate stipulates the modelling class “Artefact”. The concepts node and device are defined as physical resources. However, their logical pendants are not defined in the Archimate specification.

For example, the information flow between two application components might be modelled using two or more physical interfaces connecting the two application components. In the logical view, these physical interfaces might be depicted with only one logical interface or in some cases, just by a relation class "Information flow". For clarity, Fig. 55 illustrates these examples.

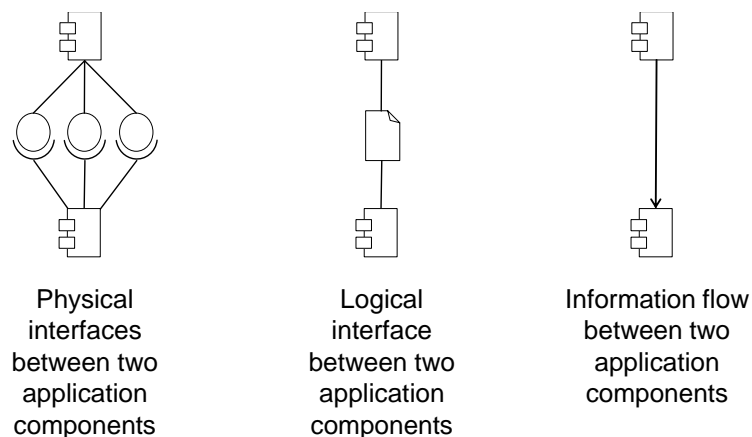


Fig. 55 Different representations of logical and physical interfaces in Archimate

Providing an overview of all interfaces within the enterprise, in this case, is a non-trivial task. Applicable harmonisation strategies in such a case are contradiction of the physical interface into one logical interface, deletion of the interfaces and reconstruction of the relations etc. In terms of DICE, populations and observable units (in this case the interfaces) have to be clearly specified to avoid biased results. Rules for differentiating between logical and physical have to be worked out by the enterprise architects in such cases. DICE can then support selecting and restructuring the data as needed.

Requirement 5: Provide method chunks to filter/consolidate logical/physical building blocks.

7.1.5.1 Types and Instances

Another characteristic of EA concepts, not to be confused with the previously introduced concepts of *logical and physical* building blocks, is the concept of *types and instances* (Moser, Winklhofer and Kuplich 2008). Types and instances can be considered as specialisation of the concept of physical building blocks discussed in the previous section.

In this context, the meaning of the term “type” is rather similar to the meaning of the word “type” in everyday language. Types represent collections of instances, and instances are specific realizations of a type. In accordance with DoDAF (DoD 2010), instances can be defined as things that exist in 3D space and time. Enterprise architects usually model types of things rather than instances, but some architecture frameworks such as DoDAF consider instances explicitly. In DoDAF, instances of types are named using the preface “individual” and an explicit “is-type-of” relation is offered (see e.g. the relation “typeInstance”). Archimate and TOGAF do not differentiate between types and instances in their metamodels.

For example, a solution building block (physical) such as the technology component Oracle 11g can be interpreted as type or as instance. If we interpret Oracle 11g as instance, it represents a solution building block, which is installed on a concrete node (a concrete server), holds a concrete database schema and stores concrete data. As opposed to types, instances correspond to things found in the real world. Typically tools such as configuration management systems (CMS) manage instances (Klosterboer 2007). They hold physical instances of databases, application components etc. of the “real world” IT landscape. Enterprise architecture is typically not interested in modelling these concrete instances. However, datasets comprising concrete instances (such as CMS/CMDB) are often quoted as valuable sources for populating EA repositories, see e.g. the survey of (Farwick et al. 2013) where the majority of EA architects are convinced that CMS/CMDBs hold EA-relevant data. A concrete example would be that enterprise architects are interested in the absolute number of instances of a certain technology. (E.g. how many oracle database instances do we have?)

Guiding example (continued): To make sure that the list of used technologies is fully covered, the enterprise architects plan to run a completeness check, i.e. in

Application of DICE in the Fields of EAA

DICE terms: the architects determine the completeness of the sample population of technologies.

Table 9 shows an excerpt of the list of technologies retrieved from the organisations Configuration Management Database System (CMDB) in csv format.

Table 9 Exemplary excerpt of CMDB report

[...]
Oracle 11gR12,CosMosDB
Oracle 11gR12,OrinocoDB
Oracle 11gR12,CRMDB
SQLserver 2016 (13.0),FinAppDB
[...]

Obviously the table contains physical configuration Items (CIs) as is customary for CMDB data. The list has to be cleaned to truly represent the list of technologies on type level. Removal of the second variable followed by consolidation of the observable units in the resulting dataset would be a sufficient approach.

With a focus on the building blocks of the business architecture level, the importance of the concept of types and instances is even more explicit. Instance data on products (sales transactions and prices), customers (customer data) and processes (workflow instances) are the classical sources for BA analysis.

By aggregating this data, valuable information for EA initiatives can be generated. The BA data will produce insights that enable decision making on competitive actions that change organisational capabilities. Examples of such competitive actions are: the launch of new products or product extension, changes in the channels through which customers are reached and expansion of target markets and customer segments (Sharma et al. 2010). Each of these changes affects the organisations business capabilities where the resources of the capabilities are based on: business processes, stakeholders, information entities, applications and technologies.

Application of DICE in the Fields of EAA

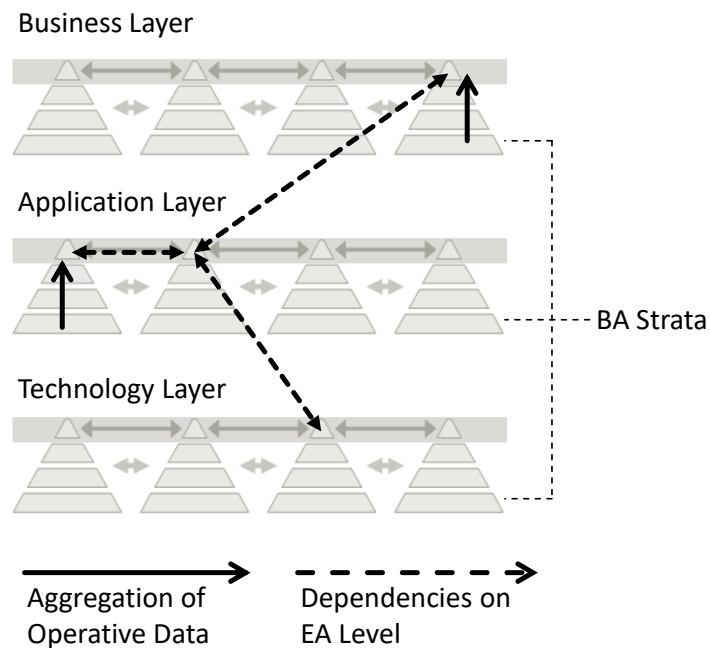


Fig. 56 Types and Instances - Usage of BA data in EAA

In regards to DICE, it must be possible to identify and filter building blocks based on their kinds (types versus instances) and preparing/aggregating instance data to be used in type data.

Requirement 6: Provide method chunks to filter/consolidate types and instance building blocks.

7.1.6 Time Dimensions

From the definitions of EA and EAM (shown in section 2.1), it is evident that time aspects play an important role. Take for example Gartner’s definition which emphasises the need for “*models that describe the enterprise’s future state and enable the organisations evolution*” (Lapkin et al. 2008). From this, one can conclude right away that time aspects are critical in the context of EA and that time aspects are inseparably related to EA descriptions. This also becomes obvious from Zachman’s Architecture Framework (Zachman 1987) where the interrogative “When?” is prominently positioned.

The reasons for permanently adapting the EA and the EA descriptions are manifold. Examples given in TOGAF are: the adaption of the organisational structure (e.g. the introduction of business units, introduction of mid-office etc.), the harmonisation of business

processes, the launch of new products or the enhancement of legacy systems to support the service oriented architectures (The Open Group 2011). In any case, building blocks and their interplay will change over time. From an EA description point of view, this holds true for the building blocks as a whole, for the relations between the building blocks as well as for their attribute values.

7.1.6.1 Architecture Increments

Enterprise architecture frameworks such as TOGAF, GERAM, FEAF, see e.g. (Saha 2004), (Buckl et al. 2009) and (Saha 2007) postulate mechanisms for time-related views on the enterprise architecture. Typically, as-is (i.e. baseline) and target architectures as well as transition architectures are differentiated. Transition architectures represent interim architectures that point the way towards target architecture. Architecture increments represent architectures at a certain point in time. All of the above mentioned EA frameworks specify metamodels, however, in none of the metamodels are time-related aspects referred to (Buckl et al. 2009).

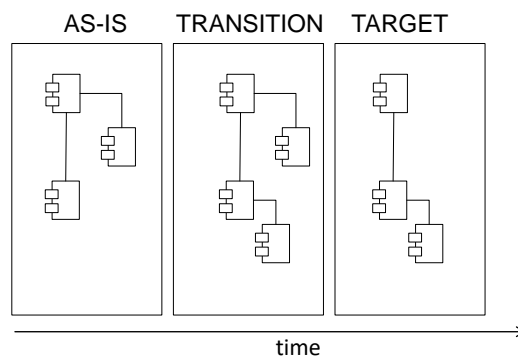


Fig. 57 As-is, transition and target architectures

Time-based architecture descriptions are “user-level versioned” models, i.e. they represent versions of EA models and building blocks created for specific purposes (Sciore 1994). Model types (model kinds) such as the “Business Support Migration Roadmap”, see (Buckl et al. 2009) typically point the way from as-is architectures towards the desired target architectures by depicting building blocks, such as applications, functions and implementation projects in Gantt-like diagrams (see Fig. 39).

Another important fact to be considered in this context is that typically the level of details between as-is, transformation and target architectures differ considerably. Whereas the as-is

Application of DICE in the Fields of EAA

architecture typically gives a clear picture of the overall architecture, target architectures usually anticipating the needs of the next 3-5 years, are not defined at the same level of detail (strata), see e.g. (The Open Group 2011).

As an example, as-is architectures often comprise concrete versions of technology products (i.e. physical view building blocks), whereas target architectures typically refer to technology capabilities and required qualities only (logical building blocks). The concrete design of implementation details is not important years in advance and might even restrict the solution space for architectural solutions in the future. Thus, as-is architectures usually comprise deeper levels of the strata as compared to target architectures (see sections 7.1.4.2 and 7.1.4.1).

Archimate recognises the need for time-based views and introduces the concept “plateau” as a solution. Plateaus are part of its language definition and are represented via the modelling class "plateau". Plateaus are used to specify points in time and can be assigned to any building block (Jonkers et al. 2010). For a better understanding Fig. 58 provides an example. The cuboid-like elements represent the plateaus, each of them representing periods in time. The associated building blocks are decommissioned / set in production at the end / at the beginning of these time periods.

Application of DICE in the Fields of EAA

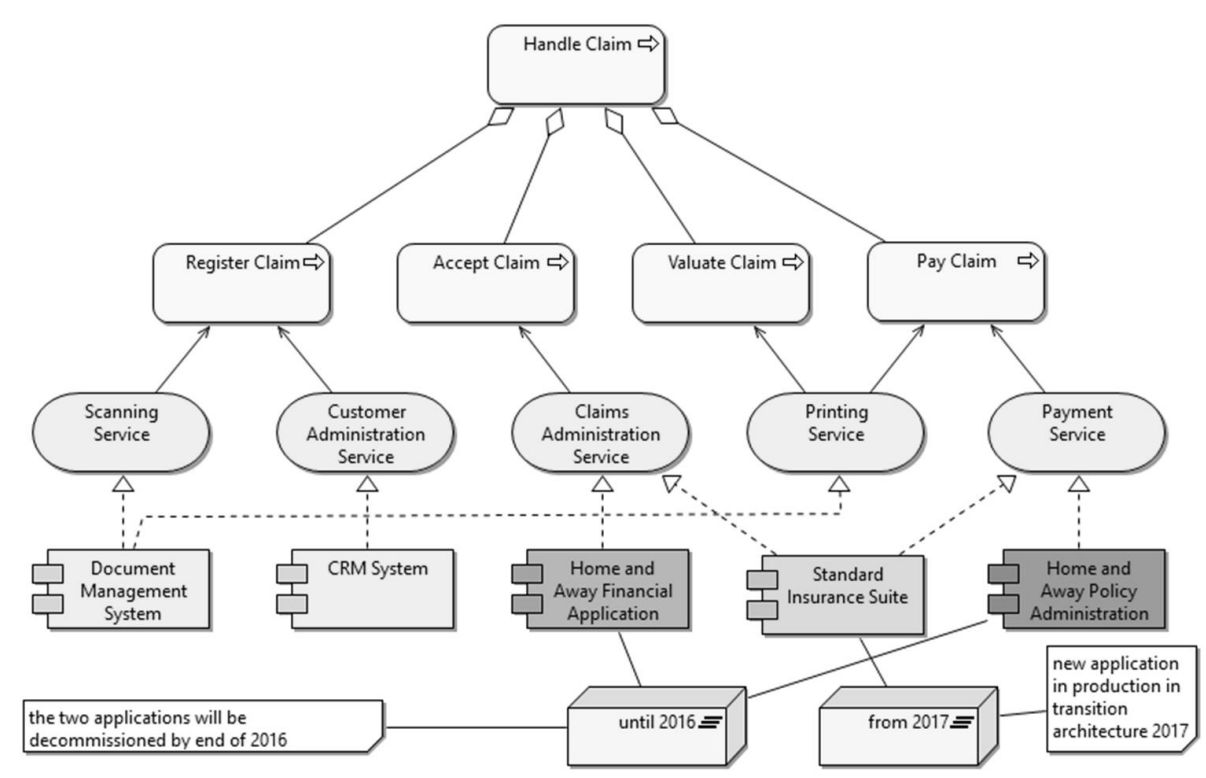


Fig. 58 Application usage model incl. plateaus depicting transition architectures

Other approaches equip the modelling classes and relation classes, respectively the building blocks and their relation classes, with time attributes directly (e.g. production date, decommission date). Via this construct, it becomes possible to select the set of active (valid) building blocks for a given architecture increment (respectively period in time). These concepts are summarised under the heading of time-based versioning and temporal modelling, see e.g. (Parent, Spaccapietra and Zimányi 1999) and (Theodoulidis and Loucopoulos 1991). Subsetting the gaps between as-is and target architectures respectively between as-is and transition architectures is also of major importance.

Another important fact is that target architectures are evolutionary in nature and are subject to continuous planning and update in consideration of evolving business requirements and technological change. By contrast, in order to avoid complications caused by moving targets, transition architectures (at least in the short-term) should not be altered during the course of the implementation projects (The Open Group 2011).

A third major aspect is the possible amalgamation of as-is and target building blocks in one EA model. In such a case, models typically contain building blocks documented on different strata. Whereas as-is building blocks are modelled on a deeper strata and possibly on a deeper

Application of DICE in the Fields of EAA

versioning level (see section 7.1.4), planned building blocks serve as “placeholders” and are described on a higher EA strata, as implementation details are not required (or not even known). Temporal aspects have to be considered in the context of building blocks, the relations between building blocks and their attributes.

Dealing with temporal aspects is a non-trivial task. EA descriptions apply different time concepts. (Gschwandtner et al. 2012) define the following types of time inherent in datasets:

- Non-rastered points in time, representing concrete points in time.
- Non-rastered intervals, representing time intervals defined by start and end date.
- Rastered intervals, representing units of time constituting a raster. An example is using Q1, Q2, Q3 and Q4 to represent quarters of a year.

Guiding example (continued): Luckily, the application portfolio catalogue carries the variables “production date” and “decommission” date for applications. Based on these variables the enterprise architects can select those applications that are currently in production. Already decommissioned application components are not of interest for the envisaged clustermap.

DICE method chunks that allow for filtering the datasets in a time-based manner need to be in place.

Requirement 7: The selection transformation task type has to cope with time-related data.

7.1.6.2 Versioning of Building Blocks

Building blocks and their relations are typically regarded as versioned items. Versioning is required for comprehensible handling of changes within the EA. The concept of versioning anticipates the possibility of the concurrent existence of arbitrary implementations of the same building block. Notably, prominent frameworks such as TOGAF, MODAF, DODAF and the modelling language Archimate do not explicitly recognise this requirement. None of their metamodels explicitly define attributes for version control of their building blocks.

As discussed in (Conradi and Westfechtel 1998) and (Sciore 1994), versioning is performed with several intentions. In the context of building blocks, a version is typically defined to supersede its predecessor. This type of version is commonly called *revision*. An example is a new version of a technology component that typically will be the result of a bug fix or feature

Application of DICE in the Fields of EAA

enhancement of a previous version. The same principle holds true for any type of building block.

The level of depth of the applied versioning strategy plays an important role. As a rule of thumb, building blocks such as applications and technology products are managed on the level of major releases in the context of EA management (Moser, Fürstenau and Junginger 2010). However, depending on the case, a deeper versioning level might have been chosen, see e.g. (Klosterboer 2007). An example is technology data held for security management issues where detailed versioning levels down to the patch level of software products might be required.

For a better understanding Fig. 59 exemplarily depicts common versioning units for applications and technology products.

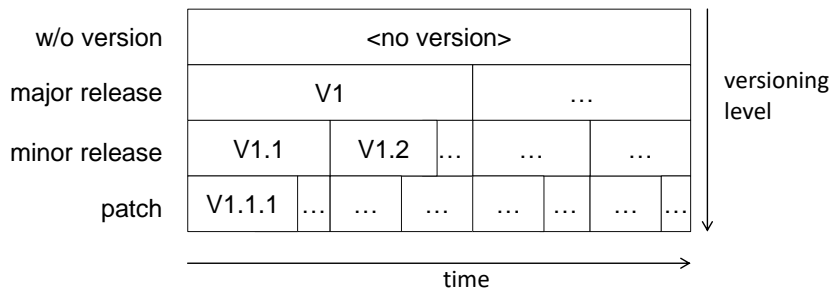


Fig. 59 Varying depths of versioning for applications and technologies

Guiding example (continued): In the used *system × technology matrix* (see Fig. 44) one can find the database system “Oracle” multiple times. It holds Oracle 11g and on a more detailed versioning level Oracle 11g.R12.

The latter is on a level too detailed for the intended purposes, thus, the enterprise architects decide to consolidate the two technologies into one building block (Oracle 11g).

In a DICE endeavour, for all observable units (EA building blocks) the applied versioning strategy has to be examined. Groupby mechanisms are required to conflate low level versioned building blocks. An example is the conflation of technologies such as Archimate’s modelling class “system software” versioned on the level of service packs into building blocks versioned on the major release level.

Application of DICE in the Fields of EAA

Requirement 8: Provide method chunks to consolidate low level versioned building blocks into higher level versioned building blocks.

7.1.6.3 Versioning of Architecture Descriptions

As the EA of an organisation undergoes continuous change, the architecture descriptions are updated on a frequent basis. Thus, EA descriptions typically evolve over time through various revisions. This versioning will lead to the existence of EA descriptions depicting the same or similar scope of the EA at different points in time.

In their work, (Moser, Winklhofer and Kuplich 2008) propose a release workflow for EA models implemented as a status automation that defines states such as “Draft”, “Quality Assurance”, “Released” and “Archived” to control the process of model creation and release. In many cases only released models will be of interest, as the quality of these models typically has been approved and will comprise validated EA data.

The given EA descriptions need to be analysed regarding their currentness and correctness. The available meta information of the models must be captured in DICE. Information about approval status, approval date, spatial information etc. is of high relevance. It is obvious that the data in formally released models is more trustworthy than data without such a characteristic. In the case of duplicates within the dataset record, linkage mechanism must be in place to resolve duplicate building blocks including their relations.

Requirement 9: Provide mechanism to filter/select building blocks in datasets based on their (description) version.

7.1.7 Planning Scenarios - Alternative Architectures

Scenarios have been widely used as a requirements engineering technique, especially with respect to software architectures, see (Dardenne 1993), (Gough et al. 1995) and (Kazman et al. 1996). Architecture scenarios are used in the design phases as a method for comparing design alternatives. The EA analysis techniques such as the “multi-criteria decision making techniques” (see section 3.1.3) build heavily on this concept.

In TOGAF, in the Phases B, C and D of the Architecture Development Method, the as-is architecture and target architectures are defined. In Phase E: Opportunities and Solutions, one

Application of DICE in the Fields of EAA

or more alternative solution architectures may be defined. Alternative design options may for example, depend on build-versus-buy-versus-reuse options (Moser, Winklhofer and Kuplich 2008). In order to choose the best solution, the alternative architecture designs are evaluated and compared. Applied evaluation criteria are: costs, time to market and architectural fit (i.e. compliance with stated architecture principles). As in TOGAF, other EA frameworks such as FEAF and GERAM recommend the design and comparison of planning scenarios.

The starting point for defining a new scenario (baseline) is the EA at any given point in time. Typically, scenarios will be designed for the target architecture. In this context, the baseline architecture represents the released and authorised state of the EA. The baseline serves as the basis for designing and implementing architectural changes.

Archimate takes advantage of its plateau concept which was introduced in section 7.1.6. As discussed, plateaus serve to depict architecture increments; they are also used to represent architecture scenarios and competing architecture designs as well. Utilizing the concept of plateaus, baseline building blocks and scenario-specific building blocks can be distinguished.

The main criteria for identifying building blocks of a baseline are that they are formally agreed and released, see (Moser, Winklhofer and Kuplich 2008). Fig. 60 depicts the baseline and its progression to the target architecture incl. examples of alternative architecture designs.

Application of DICE in the Fields of EAA

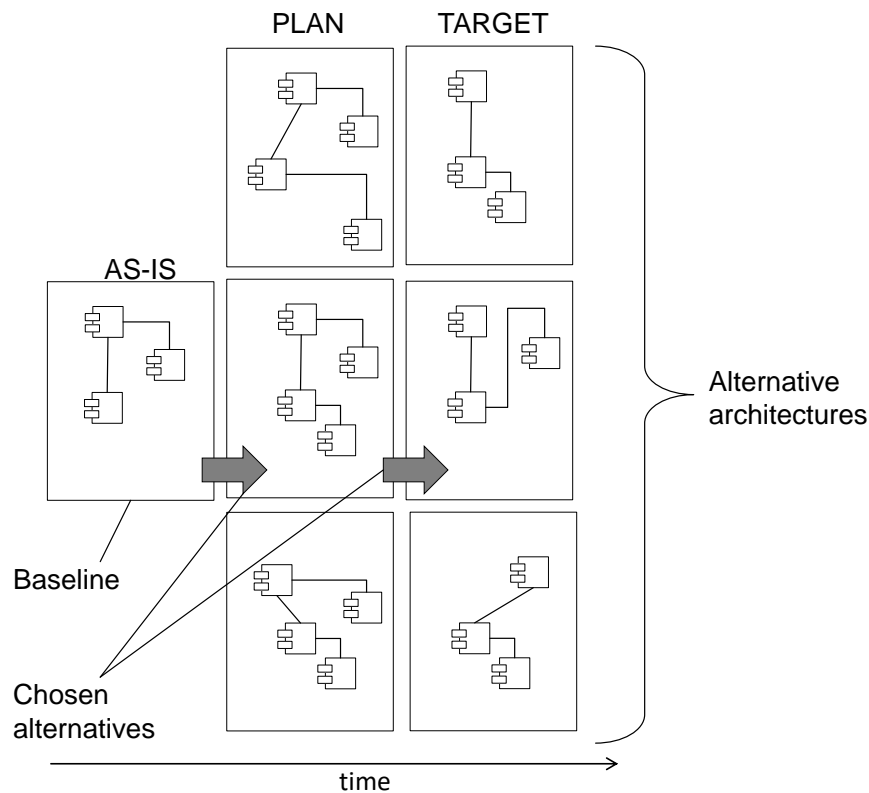


Fig. 60 Alternative Architectures

Guiding example (continued): The collected application usage models might not only be categorised as “as-is” architecture models but also planning alternatives might be contained. The enterprise architects have to analyse the models and sort out non-relevant variants to avoid biased results by merging non-relevant data into the dataset used to create the envisioned clustermap.

Summing up, it is important to differentiate between architecture scenarios and agreed plan and target architectures. Obtaining the required information from the input models and other architecture descriptions in an automated way will, in many cases, be impossible without detailed investigation of available metadata. Scenario comparison must be possible.

Requirement 10: Support scenario-based data structures.

7.1.8 Heterogeneous Metamodels

As mentioned in section 2.1, there are a great many EA metamodels. Almost all EA frameworks and approaches come with their own metamodels. In addition to that, in EAA endeavours dealing with EA content not following a “standard” metamodel at all might be required. “Standard” metamodels might have been tailored to fit a purpose in the context of specific scenarios. Thus, mechanisms for the integration of data stemming from heterogeneous metamodels are a critical requirement.

In spite of the differences in these metamodels, many of them share a common ground. All of them are intended to organise the topmost important concepts of an organisation. Take for example the metamodels of Archimate and TOGAF. Fig. 61 shows the mapping between the two metamodels published by the Open Group (Band et al. 2015).

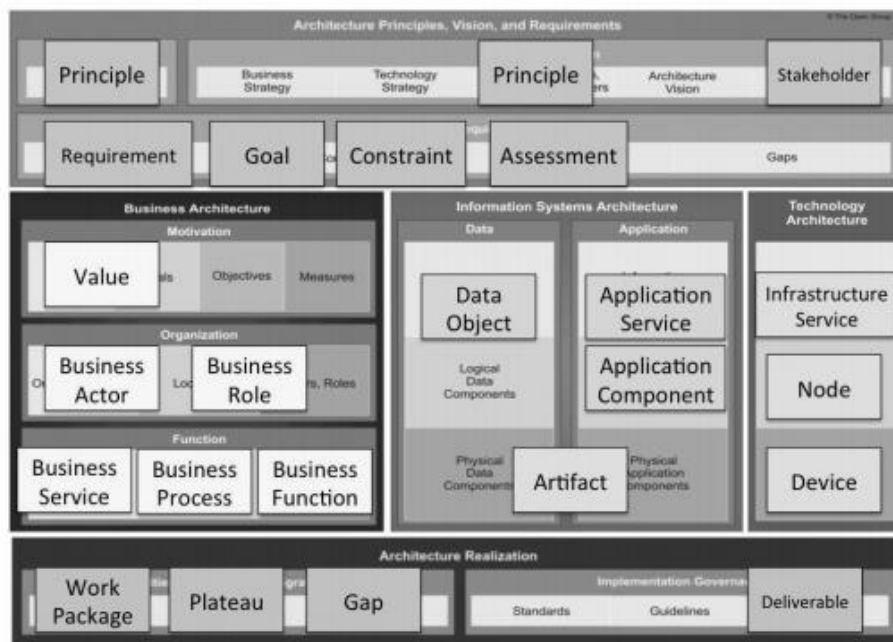


Fig. 61 Metamodel mapping: Archimate vs. TOGAF content metamodel (Band et al. 2015)

Although originally stemming from different organisations and developed independently of one another, one can recognise that the two metamodels have a high degree of overlap. Other examples of available mappings are: mappings between the military frameworks DoDAF and MODAF (Hause, Brookshier and Bleakley 2012) and the TOGAF-BIAN mapping (TOGAF BIAN Collaboration Work Group 2012) to name just some examples. Where such mappings are available, they provide valuable input for the required “normalisation” of the data.

Application of DICE in the Fields of EAA

However, often the available data may not be instantiated from EA metamodels or may be based on individually tailored metamodels.

Guiding example (continued): The case of heterogeneous metamodels exactly reflects the situation of our EAA endeavour. Datasets from different sources have been taken into consideration. Because of the small number of involved types of architecture artefacts (modelling classes), relations and attributes, the mapping is straight forward.

However, if many different types of input sources are available, support for the required mapping would be appreciated by the enterprise architects. These mechanisms ideally would generate mapping proposals based on the input datasets.

For these cases DICE needs to provide mechanisms that support the integration of the EA data.

Requirement 11: Provide means to integrate datasets stemming from heterogeneous metamodels.

7.1.9 Common Data Quality Issues

Data quality issues have already been discussed in chapter 6.1. The same criteria apply to EA data. Probably the biggest difference to common BA and DM endeavours is that in EA in many cases the population has to be fully covered, i.e. $D^{SEM>populationSize} = D^{SEM>samplePopulationSize}$. An important issue requiring special attention is the consolidation of observable units. As becomes obvious from the guiding example, consolidation of building blocks is one of the major challenges when integrating models. Thus, adequate mechanisms to identify equivalent building blocks need to be in place.

As opposed to the standard case in horizontally laid out data, in graph-oriented data additional information such as the observable units' neighbours and incoming/outgoing relations have to be taken into account.

Requirement 12: Provide mechanisms to identify equivalent building blocks.

Application of DICE in the Fields of EAA

Requirement 13: Provide mechanism for performing record linkage on EA building blocks.

7.1.10 Summary

As has been shown, EA data is object-oriented in nature. In order to examine relationships between building blocks, most of the EA analysis approaches require not only information on observable units and their characteristics but also rich structural information. From the analysis of the nature of the EA the following requirements have been derived that require an extension of the DICE core method base. Enriching the DICE core method base makes it possible to assemble a situational method for data integration and cleansing in the fields of EAA. Table 10 summarizes the derived requirements:

Table 10 Requirements for supporting EAA

Requirement	Source
Requirement 1: Provide method chunks to support the restructuring of EA data in accordance with the node-edge directed graph template.	The object-oriented nature of EA data
Requirement 2: Provide method chunks to load EA data from a diverse set of technical formats and sources.	Issues with Structural and Technical Formats
Requirement 3: Provide method chunks to filter/consolidate building blocks organised in reflexive relations.	Reflexive Relations
Requirement 4: Provide method chunks to filter/consolidate EA data organised into different strata.	Hierarchical Decomposition
Requirement 5: Provide method chunks to filter/consolidate logical/physical building blocks.	Logical versus Physical Layers
Requirement 6: Provide method chunks to filter/consolidate types and instance building blocks.	Types and Instances

Application of DICE in the Fields of EAA

Requirement 7: The selection transformation task has to cope with time-related data.	Architecture Increments
Requirement 8: Provide method chunks to consolidate low level versioned building blocks into higher level versioned building blocks.	Versioning of Building Blocks
Requirement 9: Provide a mechanism to filter/select building blocks in datasets based on their (description) version.	Versioning of Architecture Descriptions
Requirement 10: Support scenario-based data structures.	Planning Scenarios - Alternative Architectures
Requirement 11: Provide means to integrate datasets stemming from heterogeneous metamodels.	Heterogeneous Metamodels
Requirement 12: Provide mechanisms to identify equivalent building blocks.	Common Data Quality Issues
Requirement 13: Provide a mechanism for performing record linkage on EA building blocks.	Common Data Quality Issues

The following section focuses on the implementation of the above requirements by extending the DICE method base with specialised method chunks required for preparing EA data for subsequent EA analysis. The enhancement of the method base leads to a situational method for data integration and cleansing for EAA.

Requirements 1 and 2 deal with the problem of loading and restructuring data. As repeatedly stated, a countless number of possible technical and structural formats exist. Section 7.2.1 focusses on loading and restructuring data residing in the Archimate Model Exchange Format. The defined transformation task for the restructuring of models in this format is used as a representative example. **Requirements 3, 4, 5 and 8** can be condensed to the following graph of theoretical problems: contraction and filtering of vertices, edges and entire paths. This problem area is addressed in section 7.3. Solutions to the **requirements 7 and 10** (time-related and scenario-based views on EA data) follow the same solution pattern

Application of DICE in the Fields of EAA

and thus are discussed jointly in section 7.2.1. Functions supporting identification of equivalent observable units (**Requirement 13**) are presented in section 7.2.2. Record linkage issues (**Requirement 14**) are presented in section 7.2.3. Finally, 7.4 deals with heterogeneous metamodels (Requirement 11 and 12).

7.2 Extending the DICE Method Base for EAA

7.2.1 Restructuring EA Datasets

The first crucial step is always to restructure the input data into the DICE meta structure (see section 5.3.2.7). The datasets have to be *initialised* (see the instantiation transformation task type introduced section 5.3.2.1), i.e. the input datasets have to be structured into a composite data object (see Fig. 31). The initialisation transformation creates the structure and calculates the metadata where possible. Additional metadata such as the semantic metadata and the logistical metadata have to be inputted manually.

Transferring an EA model (such as the exemplary application usage diagrams, see Fig. 46) but also the matrix-oriented datasets (see the system \times technology matrix of Fig. 44) or any other input format will result in a number of datasets organised in *node and edge directed graph templates* (see section 7.1.1.3). The overall structure will be as follows:

- One dataset per building block type (modelling class) contained in the input model.
- One variable to carry the ID values for building blocks which can be arbitrary but require consistency within the set of relevant datasets.
- One edge table where each row represents one relation between a pair of building blocks indicating source and sink building blocks. The variable “Relation Type” within the edge table provides information about the type of relation (composition, aggregation etc.).

Fig. 62 represents the resulting structure (on data level) in the form of an UML diagram.

Application of DICE in the Fields of EAA

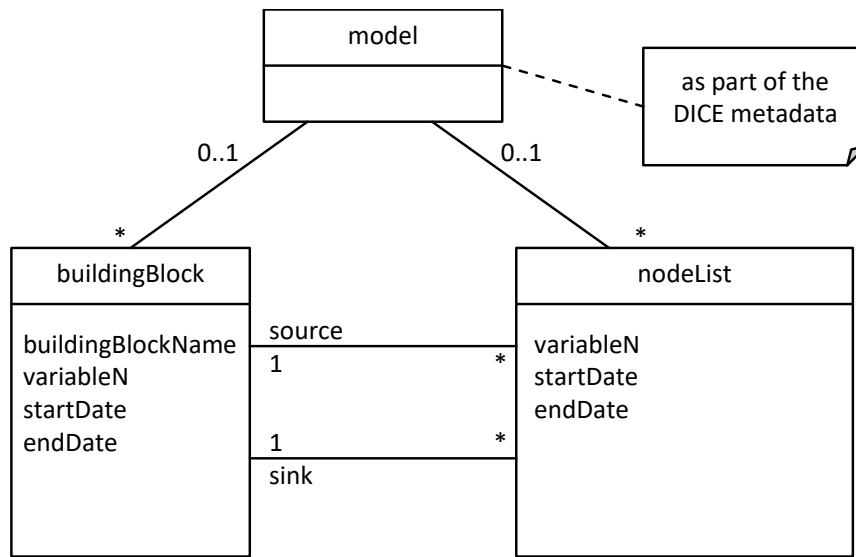


Fig. 62 Structure of datasets on data-level to represent EA data

Note (1): As none of the prominent EA frameworks (TOGAF, Archimate, DoDAF) define attributes for relation classes, this structure is seen as suitable. However, where required the aforementioned edge table can be split up into edge table per relation class, if relations carry different attributes (variables).

Note (2): In exceptional cases not all relation classes are directed. This fact is ignored due to the advantage of a simplified data structure. A more exact schema would penalize data retrieval and raise the complexity regarding the conducted data transformation steps, e.g. by requiring multi-table joins (Moody and Kortink 2000).

From the analysis of EA data structures it becomes clear that this initial data structure has to be extended to provide information about time aspects (see section 7.1.6.1). Thus, each of the above datasets (carrying the building blocks and their relations) has to be extended by the following columns:

- Start date, to provide information about the birth of a building block (e.g. production date of an application component) and
- End date, to provide information about the expiration date of a building block (e.g. decommission date of an application component).

Note (3): Rastered intervals representing units of time constituting a raster (e.g. quarters, years etc.) require only one variable indicating the rastered interval (see section 7.1.6.1).

Application of DICE in the Fields of EAA

The resulting data structure on data level has the structure as illustrated in the form of an UML class diagram in Fig. 63.

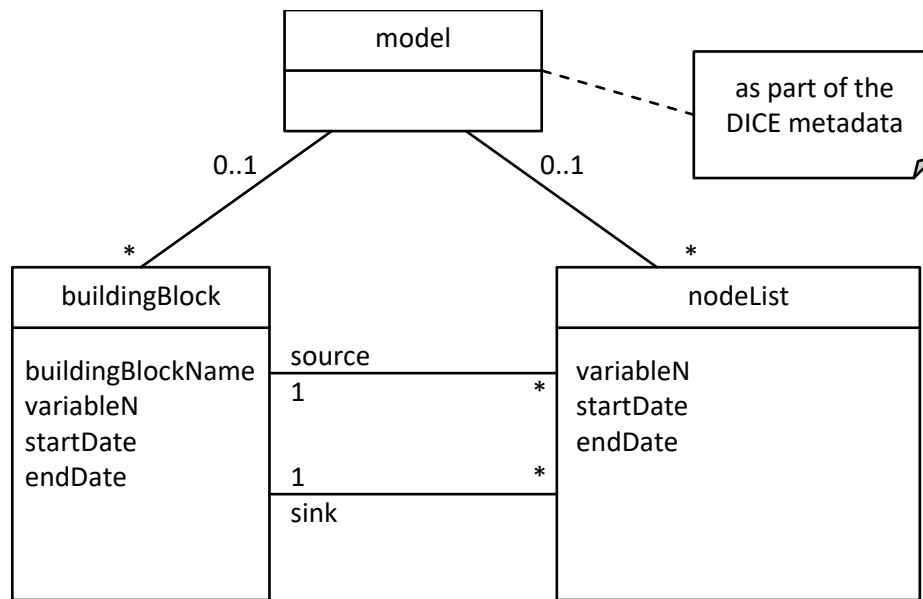


Fig. 63 Structure of datasets on data-level to represent EA data including time aspects

Typical exchange formats for EA models are: spreadsheet formats (such as csv, xlsx files), formatted text-files and xml-based formats. Some EA frameworks such as Archimate and DoDAF offer toolset-agnostic exchange formats. Both the Archimate Model Exchange File Format (The Open Group 2015) and DoDAF's PES (DoD 2010) provide well-documented XML schemas which facilitate the exchange of EA models based on their modelling languages. Of course, it cannot be guaranteed that the model-base is available based on this format. Thus, transformation of Archimate models residing in the Archimate Model Exchange File Format is solely used for illustration purposes. Based on common XML transformation techniques, the models are transformed into the required structure. Divergences to Archimate's standard format or completely different initial structures need to be tackled based on adequate restructuring of transformation tasks. In these cases, the DICE method base has to be extended with additional transformation task types. (Shu 1987) provides an overview of typical transformation techniques.

Guiding example (continued): Let us assume that the application usage models and numerous other models representing Archimate viewpoints of the organisation are available in the Archimate Model Exchange File Format. Figure Fig. 64 shows a representative snippet of such an input file. For more information see the XML schema

definition given in (The Open Group 2015).

```
<?xml version="1.0" encoding="UTF-8"?>
<model xmlns= ...>
  <metadata>
  <elements>
    <element identifier="id-855" xsi:type="ApplicationComponent">
      <label xml:lang="en">Customer Data Access</label>
    </element>
    <element identifier="id-1414" xsi:type="ApplicationService">
      <label xml:lang="en">Claim InfoServ</label>
    </element>
    <element identifier="id-596" xsi:type="BusinessProcess">
      <label xml:lang="en">Close Contract</label>
    </element>
    <element identifier="...
    ...
  </element>
</elements>
<relationships>
  <relationship identifier="id-884" source="id-861" target="id-838" xsi:type="AccessRelationship"/>
  <relationship identifier="id-880" source="id-855" target="id-837" xsi:type="AccessRelationship"/>
  <relationship identifier=".../>
</relationships>
<organization>
<views>
</model>
```

Fig. 64 Exemplary snippet of an Archimate model in the Archimate Model Exchange Format

The transformation task type “Restructure Archimate Model” reads in the file and generates the two datasets: one for the building blocks and a second one carrying the relations between the building blocks. The enterprise architects load the data and perform the restructure transformation. Finally, by executing the initialisation transformation, they create the required DICE data objects, i.e. the composite objects holding data and metadata. See section 5.3.2.1 for more information on the initialisation transformation task.

The new transformation task type “Restructure Archimate Model” is a specialisation of the transformation task type “Restructure Dataset”. Fig. 65 illustrates this extension of the DICE method base.

Application of DICE in the Fields of EAA

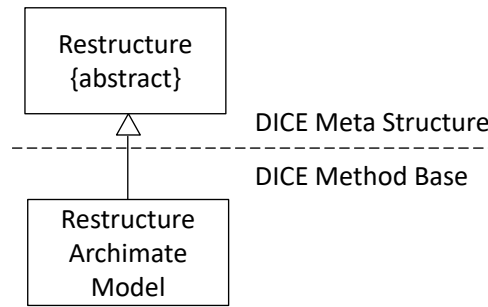


Fig. 65 The DICE “Restructure Archimate Model” transformation task type

7.2.2 Equivalence Functions

Equivalence functions serve a twofold purpose: (1) they are required for performing integration of data objects and (2) for record linkage transformations in the case of duplicate observable units.

Note: In DICE the term integration is preferred over the term join, to stress the fact that integration of datasets requires more steps than a common join operation known from relational database operations. In addition, it has to be stated that “integration” is not seen as an atomic transformation step. Integration transformation is assembled by the atomic transformation task types *addition* and *selection* (see 5.3.2.3).

Equivalence functions provide similarity/distance metrics. Discrete matches where an “all-or-nothing” principle (Dusetzina et al. 2014) is applied will lead to first suitable matches. Pairs of building blocks are classified as a match if the two building blocks share exactly the same properties in the data key. For all other building blocks probabilistic linkage methods (so-called *continues methods*) are applied.

In their work, (Jeners, Lichter and Pyatkova 2012) classify techniques to determine the similarity of entities on measurement theory into the following four categories:

- **Spatial methods** that consider entities as vectors in the n-dimensional space. Recognized representatives of these methods are the Euclidean Distance (Deza and Deza 2009) and the Cosine Similarity (Deza and Deza 2009).
- **Feature-based methods** where the set of features that the entities have in common are the basis for calculating similarity measures. Prominent methods are: the Jaccard similarity (Deza and Deza 2009) and Tversky index (Deza and Deza 2009), both representatives of similarity measures based on sets theory.

Application of DICE in the Fields of EAA

- **Transformational methods** which recognize that entities that are perceived to be similar have representations that can be transformed into one another with little effort. The number of required transformation steps is the main indicator. Prominent representatives are distance metrics such as Jaro, Jaro-Winkler and Levenshtein (Deza and Deza 2009). For a detailed overview see the work of (Choi, Cha and Tappert 2010).
- **Alignment methods** are based on Gentner's structure-mapping theory (Gentner 1983). Techniques such as the Structure Mapping Engine offer an analogy-driven approach where entities are compared based on their structural features.

In the context of EA descriptions all of these techniques appear helpful for the indication of equivalence between building blocks and thus are discussed in detail in the subsequent sections. Since DICE is conceived as a situational method, all the presented techniques can be understood as being method fragments. They are adapted to be used in the DICE initialisation transformation task ($T_{initialise}$) which (re)calculates the metadata of a data object including $D^{QUAL>uniqueness}$, the indicator that provides information on the degree of similarity of observable units within a given dataset.

In the following sections similarity techniques applicable for EA descriptions are presented. All of these techniques are understood as (process) method fragments specialising the equivalency algorithm of the initialisation transformation task type. Fig. 66 illustrates this extension introducing four EA applicable similarity algorithms: (1) syntactic similarity analysis, (2) semantic similarity analysis, (3) structural similarity analysis and (4) attribute-based similarity analysis.

Application of DICE in the Fields of EAA

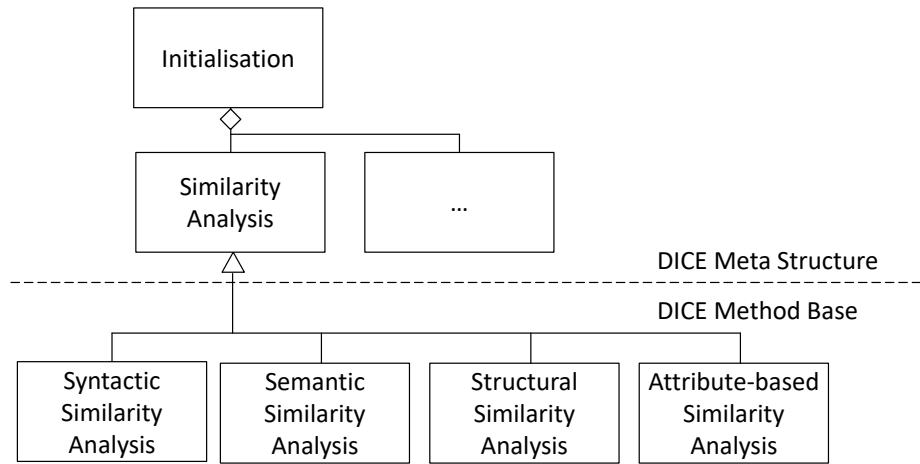


Fig. 66 Extending the initialisation transformation task type

Note that similarity analysis and its specialisations are not transformation tasks on their own; they are algorithms as part of the initialisation transformation task type.

7.2.2.1 Syntactic Similarity Analysis

This analysis technique is based on string similarity metrics. The names of the building blocks are compared. A wealth of similarity/distance measurement techniques exist for string matching. For a brief overview see, e.g. the work of (Choi, Cha and Tappert 2010) and the work of (Cohen, Ravikumar and Fienberg 2003). These techniques are commonly known as distance or similarity functions.

Note: Distance is a measurement of dissimilarity. Any distance measurement can be converted into a similarity measurement and vice versa. The distance measurements presented in the following paragraphs are normalised into intervals from 0 to 1 where 0 represents a full match and 1 stands for completely dissimilar. From this, it follows that $1 - d = s$ where d represents a distance measurement and s the corresponding similarity measurement.

Distance functions can be divided into three main categories: edit distance, token-based and hybrid (Ribeiro and Härder 2006). All of these techniques provide a cost function which evaluate edit operations (insert, delete and replace) required to convert one lexical string to another (Bilenko et al. 2003). Cohen et al. who did a performance evaluation of popular string distance metrics in the fields of name-matching (Cohen, Ravikumar and Fienberg 2003) and other sources such as (Bilenko et al. 2003) and (Ribeiro and Härder 2006) recommend the Jaro and the Jaro-Winkler metrics for the matching of short strings. More elaborate

Application of DICE in the Fields of EAA

approaches such as the *level two distance function* “Monge-Elkan distance” (Deza and Deza 2009) , a hybrid method combining token-based and edit-distance like functions perform slightly better in some cases but according to (Cohen, Ravikumar and Fienberg 2003) require up to a tenth of computational efforts in terms of computation time.

The characteristics of the strings are the most important factor when choosing the distance function. By comparing the names of building blocks, DICE has to deal with short strings. Archimate recommends either single words (nouns or verbs) or short sentences (verb-noun-combinations) for naming its building blocks. According to (Cohen, Ravikumar and Fienberg 2003), the Jaro similarity functions, which account for the number and order of common characters between the pair of two strings, perform best for short sentences and single words. The Jaro-Winkler similarity, which is an extension of the Jaro similarity, emphasizes matches in the first few characters (prefix), as Winkler recognized that typos more likely arise in the middle or at the end of a word as opposed to its beginning (Winkler 1999). However, for some of the building blocks, (namely for building blocks of the modelling classes “technology component” and “artefact”), this weighted rating is obstructive, as oftentimes these building blocks will comprise the reoccurring vendor name at the beginning of the strings, the more differentiating characters and designations in the middle and concrete version numbers at the end. Examples are: “Microsoft SQLserver 2012” vs. “Microsoft Server 2013” and “Apache Tomcat 7” vs. “Apache Tomcat 8” etc. Putting stronger emphasis on the prefix of words is inapplicable in these cases.

As is common for distance metrics, the score of the Jaro distance is normalized and will always be between 0 and 1 where 0 equates to a positive match and 1 is a bad match. In other words, the lower the distance for two strings is, the more similar the strings are. In general, scores higher than 0.2 are not considered useful. The Jaro distance is defined as follows (E. H. Porter and Winkler 1997):

$$d_j = \begin{cases} 0, & \text{if } f = 0 \\ \frac{1}{3} \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{|m|} \right), & \text{otherwise} \end{cases}$$

where m is the number of matching characters of the strings s_1 and s_2 and t is half the number of transpositions.

Application of DICE in the Fields of EAA

Furthermore, two characters from a given pair of strings are considered equal only if they are of the same character and not farther than

$$\left\lceil \frac{\max(|s_1|, |s_2|)}{2} \right\rceil - 1.$$

To obtain better results, pre-processing transformations are often performed on the strings before conducting the similarity analysis. Typical pre-processing transformations are: stop word removal (Fox 1989), white space removal and word stemming (M. F. Porter 1980). In cases where the building block names are not represented by single words but by short sentences (as is common for business processes, e.g. “Handle Claim” and “Pay Claim”), additional operations might be conducted. From the broad set of techniques from the fields of natural language processing (NLP), part-of-speech-tagging (POS) can be applied to filter out relevant text chunks only. A naïve but effective strategy would be to filter out verbs and nouns only and subsequently to assemble the text chunks alphabetically. See e.g. (Kao and Poteet 2007) for an introductory overview of NLP techniques and POS and the subsequent explanations in the guiding example.

Guiding example (continued): Take the business processes “Claim Registration” and “Register Claim” from the two EA models of Fig. 46 and Fig. 67.

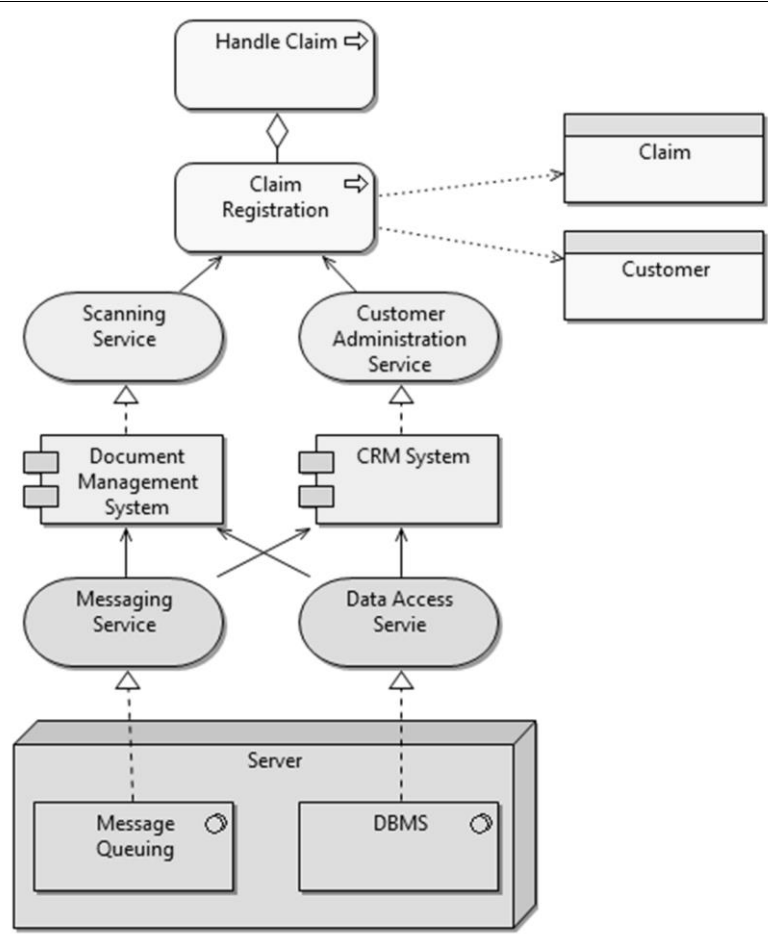


Fig. 67 Multi-layer viewpoint

Their Jaro Distance is 0,36 (Jaro Similarity of $1 - 0,36 = 0,64$). Based on this value, the enterprise architects are unlikely to consider the two business processes to be equal.

Applying POS tagging (based on the Penn Treebank tagset (Marcus, Marcinkiewicz and Santorini 1993) and subsequent word stemming using the Porter word stem algorithm, (Van Rijsbergen, Robertson and Porter 1980), the process names are represented as follows: “Register/VB Claim/NN” and “Claim/NN Registration/NN”. By reordering the text chunks alphabetically and removing white spaces and POS annotations, the enterprise architects obtain the following strings: “claimregistr” and “claimregis” with a respectable jaro distance of 0,06 (similarity of 0,94).

7.2.2.2 Semantic Similarity Analysis

Semantic similarity analysis is based on the assumption that the vocabulary that is used for naming the building blocks is taken from the given domain terminology (Deerwester et al. 1990). Thus, two building blocks are classified as similar if they use similar vocabulary based on a vector of synonyms (Kessentini et al. 2014) obtained from a thesaurus or domain ontology.

Since it is very likely that the dataset contains different concepts having the same semantics under different names (see common data quality issue: semantic similarity, section 6.2), solely identifying syntactic similarity of building blocks isolated from their semantics will not be sufficient in many cases. Take the following example: semantically similar building blocks, such as “client” and “customer”, both instantiated from Archimate’s modelling class “business object”, will obviously have a low similarity score applying the previously introduced Jaro distance which happens to be 0,51 in this case. Other similarity metrics such as the widely-used Levensthein distance will result in even lower similarity values (Cohen, Ravikumar and Fienberg 2003).

By applying semantic similarity analysis, we overcome this issue taking synonyms into account. The synonyms can be gathered from a general-purpose natural language ontology such as the lexical database WordNet (Miller 1995). Wordnet basically groups words into sets of synonyms called synsets. In other words, a synset represents a set of words in which all words have a similar meaning. For an overview on general-purpose ontologies, see (Bond and Paik 2012). For specific vocabulary, e.g. the set of technology components, specific ontologies and/or technical reference models will be helpful where available.

From the designations of two given building blocks, their synsets S and T are retrieved. In a next step the joint word set of the synsets S and T is formed:

$$J = S \cup T = (w_1, w_2 \dots w_n)$$

where J contains all the distinct words from S and T . Inflectional morphology (different word forms) can be accepted when the technique is combined with a preceding syntactic similarity analysis (see section 7.2.2.1) that will identify the similarity of morphologic words in a previous step.

Application of DICE in the Fields of EAA

The word order within the synsets is unimportant. Typically, token-based distance metrics are suitable. One simple and often quoted as being effective technique from the fields of token-based metrics is the Jaccard similarity. Between the word sets S and T , Jaccard similarity is defined as $\frac{|S \cap T|}{|S \cup T|}$, see e.g. (Deza and Deza 2009). Other prominent approaches are: the cosine similarity, the TF-IDF, the Euclidean distance and the Manhattan distance; see (Deza and Deza 2009) for these distance metrics and for many more. For the purposes at hand the cosine similarity is used, as it is very efficient especially for vectors such as the introduced synsets (Bilenko and Mooney 2003).

For calculating the cosine similarity, the name of each building block is represented in the form of a vector in n -dimensional space where n is the number of unique terms of a particular building block's names synset, i.e. the number of alternative denominations including the original name of the building block. By calculating the cosine similarity, the cosine of the angle between the two synsets (vectors based on the building block's names) is determined. For cosine similarities of value 0, the two compared vectors do not share any terms; the building blocks are not considered to be similar. In this case, the angle between two given vectors is 90 degrees. The mathematical equation is defined as follows:

$$similarity_{\cos}(S, T) = \frac{S \times T}{\|S\| \times \|T\|} = \frac{\sum_{i=1}^n S_i \times T_i}{\sqrt{\sum_{i=1}^n S_i^2} \times \sqrt{\sum_{i=1}^n T_i^2}}$$

where S_i and T_i are components of the vectors S and T and S and T represent the synsets of the names of the two observable units U_S and U_T . For a detailed discussion of the cosine similarity see, e.g. (Ye 2011).

In cases where the names of building blocks are not represented by single words but rather by short sentences, the algorithm has to be refined. Again, NLP techniques (see section 7.2.2.1) can be used to filter out relevant words only. Additionally, the mentioned classical pre-processing steps (stop word removal, word stemming etc.) can be applied to obtain better results.

Take the above example of the pair of business processes, “register client” and “record customer”. In a naïve approach one would obtain the synset of each designation by creating the superset of all synonyms of all contained words. The following synsets have been retrieved from wordnet (Miller 1995):

Application of DICE in the Fields of EAA

- $S_{register}(register, file, record, read, show, cross - file)$ and $S_{client}(client, customer, node)$
- $T_{record}(record, register, enter, put down, read, show)$ and $T_{customer}(client, customer)$.

The cosine similarity can be applied to these supersets resulting in a cosine similarity of 0.71 by comparing the vectors S and T where $S = S_{register} \cup S_{client}$ and $T = T_{record} \cup T_{customer}$.

An alternative approach is to generate the cross-products of the synsets first: $T_{recordCustomer} = T_{record} \times T_{customer}$ and $S_{registerClient} = S_{register} \times S_{client}$ and subsequently applying the cosine similarity.

7.2.2.3 Structural Similarity Analysis

This technique is based on the assumption that similar building blocks share the same neighbouring building blocks. Thus, two given building blocks which have the same neighbouring building blocks can be considered equivalent. The similarity calculation follows the same technique as the semantic similarity analysis. For each building block, the list of its immediate neighbouring building blocks is created. This list forms the word set of a given building block which has to be compared via similarity analysis. Again, cosine similarity for determining the similarity of given pairs of building blocks is applied. The cosine similarity does not penalize negative matches. This is important, as positive matches are far more important than negative matches because negative matches might simply result from the concept of viewpoints where not necessarily all relations and neighbouring building blocks are contained in a model.

Formally this is denoted as follows:

Let $N(U_u) = \{U_u^{neighbours}\}$ be the (set of names of) the neighbours of a building block U_u and consequently $N(U_x)$ be the neighbouring building blocks of building block U_x . The two building blocks can be considered similar where $distance_{cos}(N(U_u), N(U_x)) < threshold$ is given.

In DICE this type of similarity analysis is called structural similarity analysis.

Application of DICE in the Fields of EAA

Guiding example (continued): In the application usage model the direct neighbours of the business process “Register Claim” are the superordinated business process “Handle Claim”, the application services “Scanning service” and “Customer administration service”. Table 11 shows the neighbours categorised by relation class and modelling class.

Table 11 Categorized list of structural neighbours

Neighbouring BB	Connecting Relation class	Modelling class
Handle claim	Aggregation	Business process
Scanning service	Is used by	Application service
Customer administration service	Is used by	Application service

From the model in Fig. 67 the following neighbours of the business process “Claim Registration” can be obtained:

Table 12 Neighbours of the building block "Claim registration"

Neighbouring BB	Connecting Relation class	Modelling class
Handle claim	Aggregation	Business process
Scanning service	Is used by	Application service
Customer administration service	Is used by	Application service
Customer	Access	Business object
Claim	Access	Business object

$$\begin{pmatrix} \textit{Handle claim} \\ \textit{Scanning service} \\ \textit{Customer administration service} \\ \text{---} \\ \text{---} \end{pmatrix} \text{ and } \begin{pmatrix} \textit{Handle claim} \\ \textit{Scanning service} \\ \textit{Customer administration service} \\ \textit{Customer} \\ \textit{Claim} \end{pmatrix}$$

Fig. 68 Vectors on structural neighbours

Application of DICE in the Fields of EAA

Transferred into binary vectors one receives the following vectors:

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Fig. 69 Binary vectors on structural neighbours

The cosine similarity on the two vectors is 0,775.

In their work on semi-automatic schema matching algorithm for EDI/XML-based data, (Chukmol, Rifaieh and Benharkat 2005) present a similar approach classifying structural neighbours into ancestors, siblings, immediate children and leafs. This becomes possible due to the hierarchical structure of EDI/XML documents. Transferring this concept to DICE for EAA, the subset of structural neighbours have to be classified into neighbours based on the connecting relationship type. Relationship types such as “Specialization” and “Composition” make it possible to derive parent/child relationships of the building blocks. However, DICE refrains from this detailed segmentation as this would result in inadequate small subsets of neighbours to be compared. Furthermore, this approach would require high quality models as input datasets where proper utilization of the used relation types is guaranteed.

7.2.2.4 Attribute-based Similarity Analysis

This approach has to be understood as a variation on the previously discussed approaches. While up to now only the building block’s designations (the variable “name”) have been taken into account, the attribute-based similarity analysis considers multiple variables (attributes) of building blocks. A building block is considered as a vector with all its property values representing the components of the vector. Again the cosine distance can be used to calculate the similarity:

$$similarity_{cosine}(U_s, U_t) > threshold$$

where U_s and U_t (as has been defined) represent the vectors comprising all property values of selected variables of the observable units. To obtain better results one typically chooses variables of type categorical as the relevant variables to be compared.

Application of DICE in the Fields of EAA

Following the concept of (Gill and Qureshi 2015) who combined different similarity measures, the overall similarity can be calculated as follows:

$$\begin{aligned} similarity(U_s, U_t) = & similarity_{syn}(U_s, U_t) * Coef_{syn} + similarity_{sem}(U_s, U_t) * Coef_{sem} + \\ & similarity_{struc}(U_s, U_t) * Coef_{struc} + similarity_{attr}(U_s, U_t) * Coef_{attr} = \\ & \sum_{i \in \{syn, sem, struc, attr\}} similarity_i(U_s, U_t) * Coef_i \end{aligned}$$

where $\sum_{i \in \{syn, sem, struc, attr\}} Coef_i = 1$ and $0 \leq Coef_i \leq 1$.

The weight per coefficient must be adapted to the given situation and applied naming conventions (if any) of the building blocks.

An important precondition for calculating the similarity of building blocks is that the two building blocks do not have any relationships to one another. Take the following example: the business process “Claim Registration” is specialised into a business process “Claim Registration (online)”. As the processes have similar names, they will be considered as equivalent (depending on the defined threshold). From the specialisation relation between the two building blocks it is obvious that they have to be considered as different objects.

The following condition applies:

$$similarity(U_s, U_t) = \begin{cases} 0, & \text{if } U_s \in N(U_t) \\ \sum_{i \in \{syn, sem, struc, attr\}} similarity_i(U_s, U_t) * Coef_i, & \text{otherwise} \end{cases}$$

Thus, building blocks can only be considered similar as long as there is no relation defined between the building blocks. However, especially when it comes to specialisation relations between building blocks, it will be permissible to consolidate the specialised building block with its superior building block (see section 7.3).

7.2.3 Method Chunks Supporting Record Linkage

A typical transformation in the case of EA analysis is the addition of self-contained models to support cross-model analysis. In the guiding example, self-contained models are merged to support EA analysis. The required DICE transformation step (after restructuring the models) is the *addition* that will ultimately lead to duplicate observable units in the various datasets. In EAA the types of observable units typically correspond to the modelling classes defined in the metamodel. Performing addition on the models leads to a disconnected super graph where

Application of DICE in the Fields of EAA

each of the former models represents a graph component. Via record linkage, duplicate observable units are fuseded such that the independent graph components form a connected graph.

Fig. 70 schematically illustrates the required transformation based on the pair of two simplified EA models.

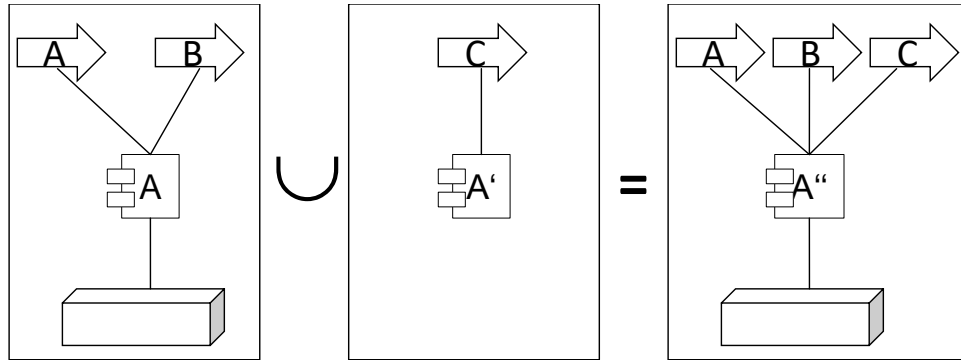


Fig. 70 Merging of EA models

The record linkage transformation is composed of the following atomic DICE transformation task types:

1. Restructuring the input models into data objects.
2. Addition of the data objects.
3. Blocking of building blocks.
4. Similarity analysis on building blocks.
5. Consolidation of equivalent observable units.

In the following sections, focus is placed on blocking strategies and the consolidation of the duplicate observable units. Restructuring of EA models has been discussed in section 7.2.1, and for addition of the datasets, the DICE transformation task type “addition” introduced in section 5.3.2.3 can be applied.

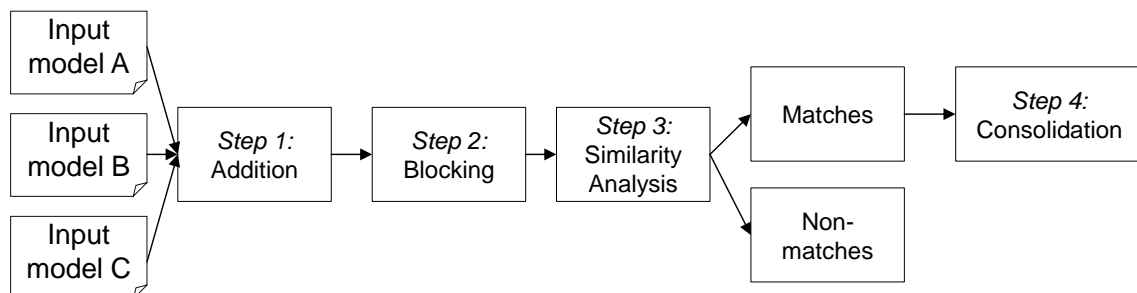


Fig. 71 Generic record linkage process

7.2.3.1 Perform Blocking

Record linkage requires the comparison of all building blocks within a dataset. The comparison space is the Cartesian product of all pairs of building blocks. With a given large population a naïve approach of comparing all building blocks is computationally prohibitive (Dusetzina et al. 2014). The number of required comparisons is $\frac{|\Omega| \times |\Omega| - |\Omega|}{2}$ where Ω is the population, i.e. the set of building blocks within the dataset. The concept of *blocking* makes the reduction of this wide search space possible by determining which record pairs to consider. It makes possible significant improvements in related processing efforts. To this end, the building blocks are grouped by so-called blocking attributes. Thus, the application of blocking strategies results in partitioning the initial set of building blocks into subsets. The matching of a given pair of building blocks will then be restricted to the building blocks residing in a block.⁷

The building blocks within these subsets must at least share one common attribute which characterises a block. Manifold blocking algorithms have recently been proposed. Prominent examples are: the standard blocking technique, the sorted neighbourhood technique, the q-gram indexing technique, the canopy clustering technique and the TFIDF, see (Baxter, Christen and Churches 2003) for an overview. Most of these blocking algorithms are based on the multi-pass approach first discussed in the work of (Hernández and Stolfo 1998) where candidate matches are generated using different attributes (and combinations of attributes) across independent matching passes.

For EAA the required partitioning into separate datasets can easily be performed by the DICE transformation task type “selection”. When applying a blocking strategy to any given dataset based on the Archimate modelling language, obviously the building block type (i.e. the modelling class) must be considered as a selection criterion. Depending on the given set of building blocks, and thus, on their different modelling classes, this strategy might lead to more than fifty subsets (blocks), as the Archimate 3.0 specification comes with more than fifty modelling classes. Fig. 72 illustrates the identified blocks of the guiding example’s

⁷ Note: the notions “building block” (the elements within the EA models) and “blocking” are coincidentally termed in a similar way and have nothing in common.

Application of DICE in the Fields of EAA

application usage diagrams. The initial table containing all the building blocks has been split into one table per modelling class (i.e. per business process, application service and application component). Each block is generated by performing a DICE selection task ($T_{selection}$) based on a selection criteria $P^{SEM>type} = [\text{modelling class}]$.

Guiding example (continued): After performing the blocking transformations (i.e. selection of building blocks per type), the enterprise architects receive four datasets: one table per modelling class and one additional table carrying the relations.

building BlockID	building Block
1	Scanning
2	Document Management System
...	...
5	CRM System

relationID	modelID	source Building BlockID	sink Building BlockID	relation class
1	1	1	2	aggregation
2	1	1	3	aggregation
3	1	1	4	aggregation
4	1	1	5	aggregation
5	1	6	2	aggregation
6	1	7	2	serves
7	1	8	3	serves
...

Fig. 72 Data after applying blocking transformation

The sketched approach is based on the blocking technique “standard blocking”; all building blocks of the modelling class are inserted into the same block (Peter Christen 2007). The major drawback of standard blocking is that errors in the blocking key values will lead to building blocks being assigned to the false block. This problem is mitigated by the assumption that the input format is based on Archimate’s Model Exchange Format. Under these circumstances, one can assume that all building blocks are typed correctly in regard to their underlying modelling class.

However, in case further blocking is required due to the high number of building blocks per block, additional blocking strategies can be applied. For an overview of blocking strategies, such as sorted neighbourhood, q-gram based blocking, canopy clustering and string map based blocking, see e.g. (Baxter, Christen and Churches 2003) and (Peter Christen 2007). In any case, the use of blocking strategies needs to be well thought through as errors in the blocking key values will lead to records being inserted into the false block, and thus, they will not be compared.

7.2.3.2 Similarity Analysis

Similar to Gill's and Qureshi's approach to aligning metamodels (Gill and Qureshi 2015), a multiple-step strategy (also referred to as iterative strategy to record linkage) (Dusetzina et al. 2014) based on all of the above similarity techniques is often most effective. To this end, the building blocks are matched in a sequence of steps. Building blocks that do not match syntactically are passed on to a second step for semantical analysis and so forth. The calculated similarities are collected in a node list: a dataset holding the pairs of compared building blocks along with the calculated similarity measures. Fig. 73 shows the structure of this dataset.

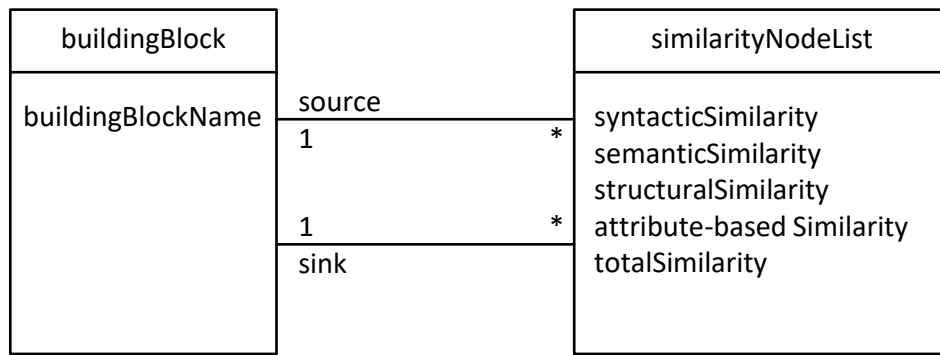


Fig. 73 DICE similarity node table

From a graph theoretical point of view, additional edges are created between the pairs of building blocks.

All of the proposed similarity functions are not applicable for all types of building blocks. In a semantical similarity analysis a thesauri or domain specific ontologies need to be available. This might be the case for building blocks of type business process and business objects since little domain-specific vocabulary is required. For building blocks of type application component where application components often carry fictional names or abbreviations, it is unlikely to find suitable general purpose ontologies. DICE considers this issue by introducing coefficients which make weighing the similarity measures possible (see section 7.2.2.4).

In the following section the required merge approach is presented in more detail.

7.2.3.3 Consolidation of Equivalent Building Blocks

Each of the four similarity analysis steps generates a so-called similarity matrix, a symmetric square matrix of distances between the building blocks. Each entry d_{ij} in such a matrix is the distance (or similarity) between building blocks i and j . The matrix is symmetric ($d_{ij}=d_{ji}$) and its diagonal has values of either 1 (one) in case of similarity or all 0's in case of distance. Hence, for reasons of symmetry only the left lower part of the matrix or likewise the right upper part of the matrix has to be considered. Fig. 74 illustrates this issue.

	U_1	U_2	...	U_n
U_1	•	d_{12}	...	d_{1n}
U_2	d_{21}	•	...	d_{2n}
...	•	...
U_n	d_{n1}	d_{n2}	...	•

Fig. 74 Symmetric distance matrix

As explained in section 7.2.2, only those candidate pairs of building blocks are relevant that have a distance lower than a defined threshold τ , that is to say a similarity measure beyond a given threshold. Two given building blocks with a distance $d_{ij} < \tau$ are considered to be equal and have to be consolidated.

As has been introduced in section 5.3.2.6, DICE provides two basic functions for consolidation of observable units: $T_{consolidationRecordLinkage}$ and $T_{consolidationAggregation}$. The first has to be applied in the case of duplicate observable units, i.e. observable units which have an equivalent observable unit within the relevant datasets. The latter is used as a consolidation step after variable removal. In this view, the combination of a variable removal transformation and a subsequently applied consolidation transformation represent a groupby transformation.

Consolidation of observable units, like any other transformation step, impacts the quality metadata. The applied similarity measures are incorporated into the quality indicators of the properties. Thus, the set of quality indicators on the property level is extended by the quality indicator $P^{QUAL>uniqueness}$.

7.3 Contraction of Edges and Vertices – Consolidation Strategies

The requirement of consolidating edges and vertices stems from the diverse characteristics of EA data, such as reflexive relations, hierarchical decomposition, types and instances, physical vs. logical view and variants of building blocks. In all cases, formerly discrete building blocks are merged into one new building block. In (Kurpjuweit and Aier 2009), the authors conclude that three categories of relations are of importance: association, aggregation and generalisation. Specialisation is considered as the opposite of generalisation. Thus, generalisations can be converted into aggregation and vice versa by converting the direction of the relation. Any other relationship types of an EA metamodel can be classified into these categories (Kurpjuweit and Aier 2009). For a better understanding Fig. 75 presents this idea for some of the Archimate relationship types.

Application of DICE in the Fields of EAA

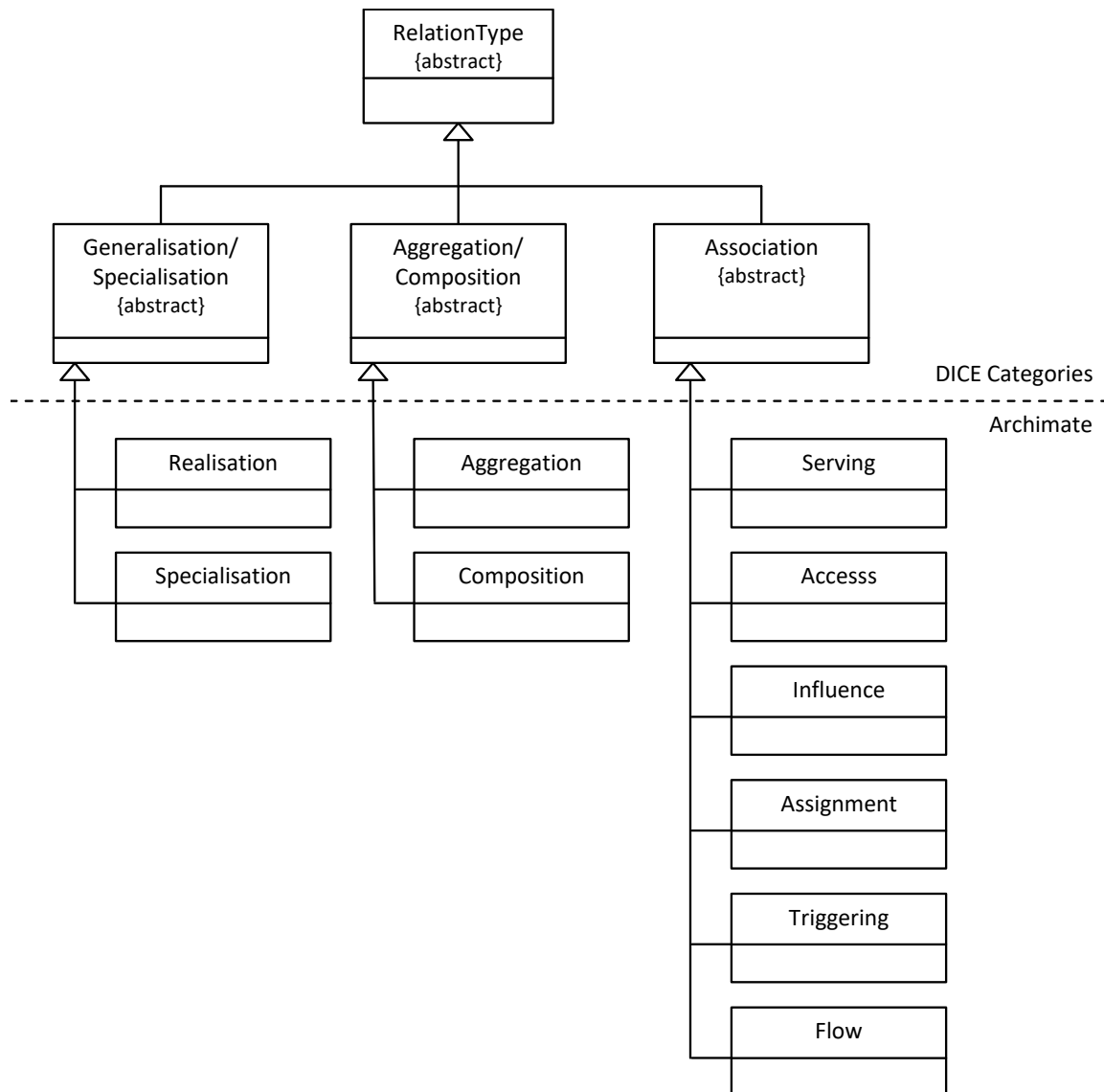


Fig. 75 Generic Relationship Types in DICE mapped onto Archimate relations

The following types of relations have to be considered:

- **Whole/part relations** where one building block can be considered to be part of another. The relationship types *aggregation* and *composition* are the typical representatives. Obviously, in such a relationship the component (the subordinated) building blocks can be consolidated into the super-ordinated building block but not vice-versa.
- The situation is different with **generalization/specialization** relationships. Subordinate building blocks are subsets of building blocks of a superior building block where specialization building blocks inherit the attributes of the generalization

Application of DICE in the Fields of EAA

building blocks. Per definition, generalization building blocks can be consolidated into their specialization building blocks. However, the data engineer can decide to consolidate specialization building blocks into generalization building blocks if he considers this a valid decision.

- The case of **association** is not that clear. Associations can be bidirectional, directed or undirected and do not provide inheritance and/or containment mechanisms per se. The desired behaviour in the case of record linkage has to be specified for the given situation. The similarity relation introduced in section 7.2.3.3 is a special case. Obviously, pairs of building blocks have to be consolidated in this case.

From the fields of graph theory, graph transformation techniques, such as *vertex*, *edge* and *path contraction* are adopted for DICE. In a graph G , contraction of an edge e with vertices x, y is the substitution of x and y with a new vertex such that edges incident to this new vertex are the edges that originally were incident to x or y . The resulting graph $G \setminus e$ has one less vertex than G (Wolfram Alpha 2017b). Vertex contraction is defined as a less restrictive transformation. In the case of vertex contraction, vertices can be substituted by a new vertex without originally being connected via an edge. Thus, vertex contraction may be applied to any pair of vertices.

7.3.1 Consolidation for Record Linkage

In DICE terms, edge and vertex contraction refers to the consolidation of building blocks. Take again the example section 7.2.3.2. Duplicate building blocks are consolidated. Duplicate building blocks are identified via their similarity measure stored in the similarity node list (see Fig. 73). Relations representing high similarity between a pair of building blocks are contracted. The direction of the association relationship is irrelevant in this case.

Application of DICE in the Fields of EAA

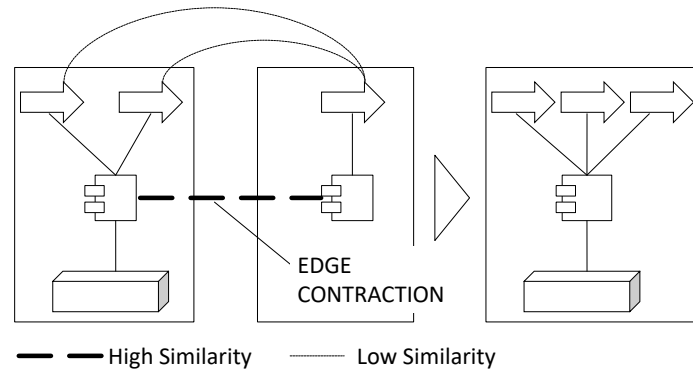


Fig. 76 Contradiction of equivalent building blocks

As has been defined in section 5.3.2.6, a consolidation transformation of type $T_{consolidateRecordlinkage}$ has to be performed. In the DICE super structure there is no extra handling of relationships. Thus, the method base has to be enriched by a transformation task type that is capable of handling the incidence relations (which connect the adjacent building blocks).

7.3.2 Consolidation of Hierarchically Structured Building Blocks

It is obvious that in the case of whole-part relations or specialisation/generalisation relations different building blocks will be retained. Thus, relationship type and direction are important. For a better understanding Fig. 77 illustrates an edge contraction based on the exemplary application usage model of Fig. 46. One of the level-1 processes is consolidated with the superior level-0 process.

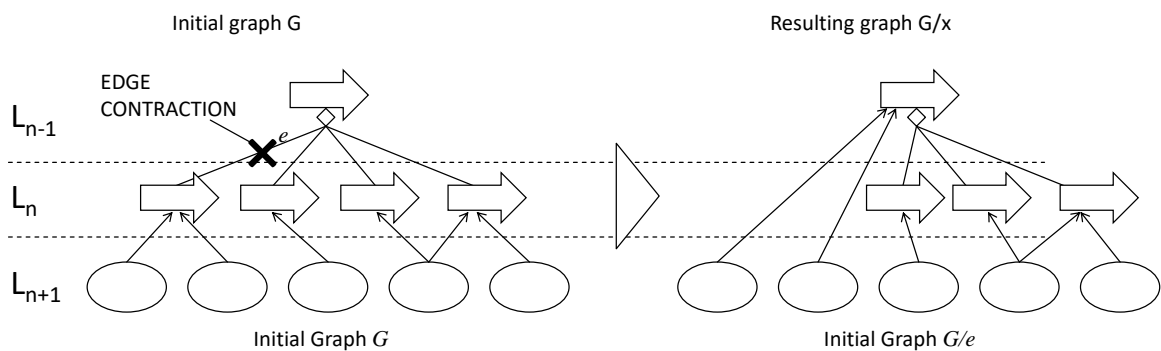


Fig. 77 Contraction of a building block in the case of reflexive relations (with aggregation relation)

Removal of all of the L_2 objects corresponds to vertex deletions of all L_2 objects followed by replacement of all previously adjacent edges with new relations. The cross product of the set of L_1 and L_2 elements where both sets contain the neighbours of a removed vertex $u \in L$ has

Application of DICE in the Fields of EAA

to be determined to create the relations. Formally this condition is written as follows:

$$N_G(u) = \{L_{n-1}, L_{n+1}\} \text{ and } L_{n-1} \times L_{n+1}$$

where $N_G(u)$ is the function which retrieves an observable unit's (u) neighbouring building block and L_{n-1} as well as L_{n+1} are the sets of neighbouring building blocks. Removal of all building blocks on L_2 results in a model as illustrated in Fig. 78.

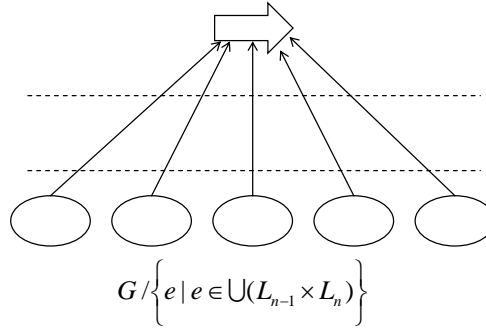


Fig. 78 One level entirely removed

7.3.3 Special Case of Reflexive Relations

Deciding which level to remove in order to acquire the required results clearly depends on the targeted dataset, in the guiding example the dataset required to create the envisioned cluster map. Usually, the determination of the required level will not be that straight forward. The EA might contain structures organised in different levels of hierarchy. A strategy for “normalising” the layers has to be defined. Fig. 79 illustrates an example with different levels of hierarchy in the case of reflexive relations.

Application of DICE in the Fields of EAA

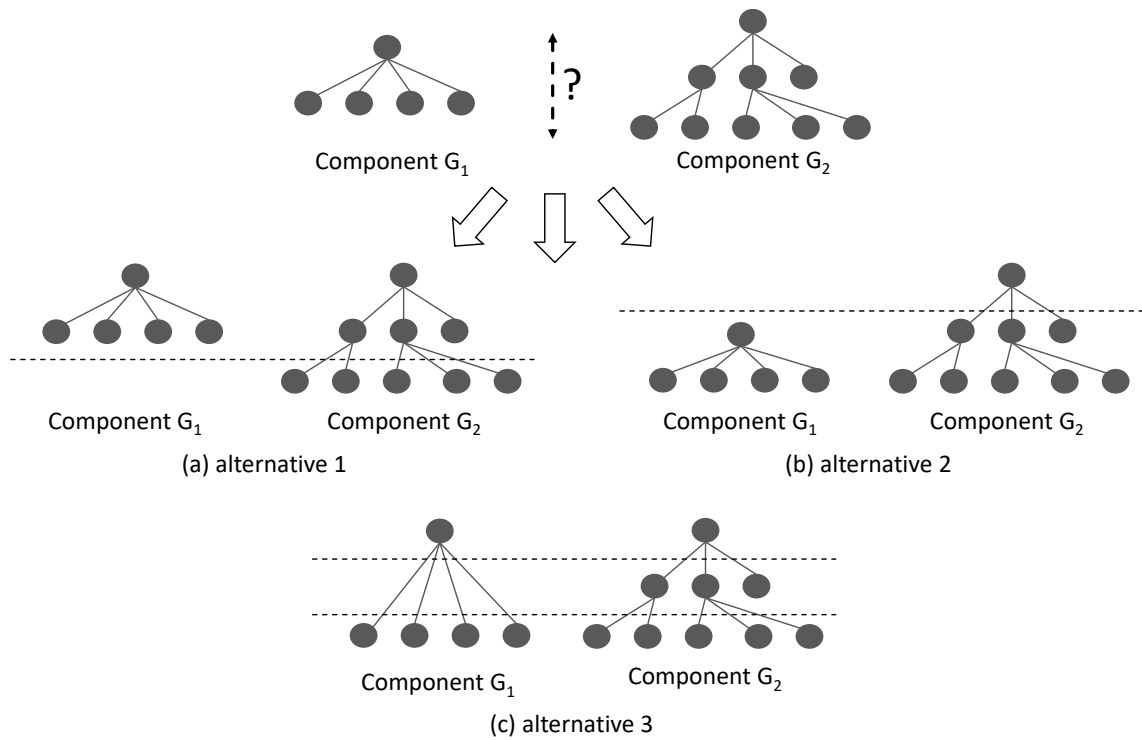


Fig. 79 Matching reflexively structured graph components

From Fig. 79 it becomes clear that determination of hierarchy levels is a non-trivial problem. The data engineers have to decide which levels to keep, remove or merge. As a general rule, one can state that siblings and cousins reside on the same level. Possible non-supervised strategies involve:

- vertical alignment to the top where the ‘top’ is defined by the height of the root node on the *longest path* from the root node to a leaf,
- vertical alignment to the bottom where ‘bottom’ is defined as the height of the leaf on the longest path from the root node the leaf,
- centered alignment where ‘center’ represents the middle level of the longest path
- or any individually defined situational strategies.

Using one of these strategies requires knowing the actual level where a building block resides in the hierarchy. These kinds of hierarchies comprise reflexive relations (see section 7.1.4.1) with the building blocks organized in a parent-child hierarchy (i.e. relations of super type specification, composition or aggregation). In graph theory, these hierarchies correspond to directed acyclic graphs (DAG) and the problem can be considered as an all-pair longest path problem. The longest path problem deals with discovering a simple path of maximum

Application of DICE in the Fields of EAA

length within the given graph. Simple paths are paths which do not contain repeated vertices. For the given problem statement, the graph can be considered as non-weighted, hence the length of a path can be calculated by simply counting its number of edges.

The given all-pairs longest path problem reduces to an all-pairs shortest path problem (Khan 2011). DICE adopts the Floyd–Warshall algorithm (Weisstein 2008), a general purpose algorithm for solving all-pairs shortest path problems. The algorithm uses each vertex (building block) in turn and computes the distance between every pair of building blocks in the model. The edges have no weights, i.e. a weight of 1. To adapt the algorithm (from shortest path to longest path) the edge weights need to be negated (multiplied by -1). Based on this transformation, the problem can be considered as a shortest path problem which is solved by applying the standard algorithm. Finally, the resulting distances have to be negated again to represent the longest paths.

```
% bb ... buildingblocks of the model %
let dist be a |bb|×|bb| matrix of min. distances initialized to -∞ (negative infinity)
for each bb
    dist[bb_v][bb_v] ← ∞
for each relation (bb_u, bb_v)
    dist[bb_u][bb_v] ← w(bb_u, bb_v) = -1 % the weight of the relation (u,v) is -1 %
for bb_k from 1 to |bb|
    for bb_i from 1 to |bb|
        for bb_j from 1 to |bb|
            if dist[bb_i][bb_j] > dist[bb_i][bb_k] + dist[bb_k][bb_j]
                dist[bb_i][bb_j] ← dist[bb_i][bb_k] + dist[bb_k][bb_j]
            end if

(|bb|×|bb|)* (-1) % negate the distances in the matrix %
dist[bb_v][bb_v] ← 0
```

Fig. 80 Longest path algorithm to determine height of building blocks in graph-based models

Like many of the shortest path algorithms (e.g. the Dijkstra's algorithm and the Bellman–Ford algorithm), see (Pallottino 1984)), the Floyd–Warshall algorithm does not compute the shortest paths themselves but rather the shortest distances between the pair of two nodes. From that the nodes and vertices defining the shortest path can easily be obtained. This of course also holds true for the longest distances computed with the introduced longest path algorithm.

For the problem at hand, the resulting matrix is used to define the height of the building blocks. Starting from the source nodes (level 0), all nodes with a distance of 1 represent the

level 1 nodes; nodes with distance of 2 represent level 2 nodes and so forth. On this basis the data engineer can decide on matching levels and levels to be contradicted.

7.3.4 Filtering and Contraction of Entire Paths

As in any BA endeavour, the input datasets will come in many different structures. Typically EA models contain complex dependencies between the building blocks. To serve as input for the subsequent EA analysis, non-relevant building blocks and their dependencies have to be filtered. Often entire paths within the EA models have to be filtered.

Take the example of the introduced guiding example. To produce the final analysis view, only building blocks of the types business process, application component and technology are required. The selection transformation task type supports the filtering out of the relevant building blocks and their relations. However, filtering is a non-trivial task, as important relations between the observable units might get lost. To keep all relevant dependencies between building blocks and at the same time allow the filtering of the datasets, DICE comes with the concept of path contradiction. It makes possible the deletion of unnecessary building blocks and at the same time the restoration of dependencies between the building blocks.

Archimate proposes a concept which supports object contraction transformations. Archimate comes with an abstraction rule called “derivation of relations” which states that *“two relationships that join at an intermediate building block can be replaced by the weaker of the two”* (The Open Group 2016), bypassing the intermediate building block. Formally this rule is specified as follows: *“If two structural relationships $r:R$ and $s:S$ are permitted between elements a , b and c such that $r(a,b)$ and $s(b,c)$, then a structural relationship $t:T$ is also permitted with $t(a,c)$ and type T being the weakest of R and S ”* (The Open Group 2016). Information on the formal derivation of this rule is provided in (van Buuren et al. 2004). DICE for EAA makes use of this concept, which makes it possible to bypass intermediate observable units and at the same time preserve the relations between the remaining building blocks.

Guiding example (continued): Take Fig. 81 as an example. The model presents paths from application components down to the technology level. For the envisioned clustermap only application components and technology components (system software) are of interest.

Application of DICE in the Fields of EAA

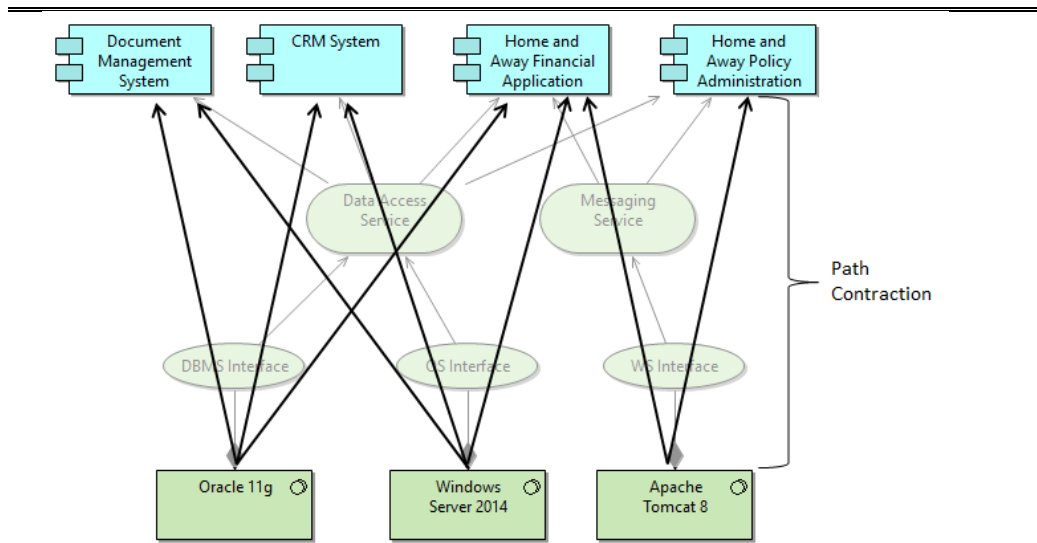


Fig. 81 Example of derived relationships

The enterprise architects make use of the derived relations concept to bypass all non-relevant building blocks and then delete these from the dataset.

Filtering the relevant paths and creating the derived relations between the remaining building blocks can be considered as *shortest path problem* in a directed graph. A path is defined as a graph whose vertices can be arranged in a sequence $v_1, v_2, v_3 \dots v_n$ such that the edge set is $E = \{v_i, v_{i+1} | i = 1, 2, \dots, n-1\}$. With regard to EA, model's paths connect building blocks. In section 7.1.1.3 EA models have been defined as directed cyclic graphs. It can be concluded that there can be more than one path connecting two building blocks within an EA model. DICE assumes that the greater the distance between a pair of building blocks, the weaker is the dependency on one another. Thus, the number of intermediate nodes within a graph plays an important role. Revisiting the concept of derived relationships of Archimate, one can see that Archimate goes one step further by assigning weights to its relations.

The Archimate relationships on which this rule is intended to be applied and their associated weights are the following: association (1), access (2), used by (3), realization (4), assignment (5), aggregation (6) and composition (7). Under the premise that two consecutive relations on a path within a model (dataset) point in the same direction, a new relation bypassing the intermediate object can be derived. A similar concept inspired by Archimate and universalized to be applicable for any EA metamodel has been introduced by (Kurpjuweit and Aier 2009). EA models and datasets not based on Archimate have to be extended by this

Application of DICE in the Fields of EAA

mechanism. Existing relation types have to be categorised into concepts, such as specialisation, generalisation and composition. Additionally, weights have to be assigned to the type of relations. Alternatively, the distance between two building blocks can be calculated simply by counting the nodes on the path between the two building blocks. In this case, simply the shortest path between vertices on a non-weighted graph has to be calculated. However, for these cases it is obvious that the resulting graph of shortest paths will not be that exact.

For EA analysis, one is usually interested in the most direct paths, i.e. in the shortest path, as it can be assumed that the greater the distance between two building blocks, the lower the reliability of estimates on their dependencies. The problem of finding the shortest paths between two nodes within a graph is well-studied in the fields of graph theory: “*Given a directed weighed graph, and a set of pairs of vertices, $\{(u_1, v_1), \dots, (u_n, v_n)\}$ where $u_i, v_i \in V$, the problem is to compute, for each i , a simple path in G from u_i to v_i (a list of vertices $u_i = s_{i0}, s_{i1}, \dots, s_{ik} = v_i$ such that for all $0 \leq j < k$ and $(s_{i,j}, s_{i,j+1}) \in E$ such that no other simple path in G from u_i to v_i has a lower total weight*” (Woburn C.I. Programming Enrichment Group (PEG) 2016).

In EA, one is usually not simply interested in the shortest path between two nodes, respectively between two building blocks. Moreover, a so-called *all-pairs shortest path* is required where the shortest paths between two sets of building blocks can be calculated. The two sets are typically sets of building blocks each of a certain type. In the guiding example the enterprise architects are interested in all shortest paths connecting business processes to application components, any intermediary building blocks such as building blocks of type “business service” have to be omitted.

DICE again adopts the Floyd–Warshall algorithm (Weisstein 2008) for the given problem. To this end the Archimate weights on relation types (see above) have to be inverted such that a small weight reflects a strong relation: association (7), access (6), used by (5), realization (4), assignment (3), aggregation (2) and composition (1).

Fig. 82 shows the Floyd-Warshall algorithm adapted to support the DICE path contradiction.

Application of DICE in the Fields of EAA

```
% bb ... buildingblocks of the model %
let dist be a |bb| × |bb| matrix of minimum distances initialized to ∞ (infinity)
for each bb
    dist[bb_v][bb_v] ← 0
for each relation (bb_u, bb_v)
    dist[bb_u][bb_v] ← w(bb_u, bb_v) // the weight of the relation (u,v)
for bb_k from 1 to |bb|
    for bb_i from 1 to |bb|
        for bb_j from 1 to |bb|
            // weakest relation within the path bb_i-bb_k-bb_j is inserted between bb_i and bb_j if ...
            if dist[bb_i][bb_j] > min(dist[bb_i][bb_k], dist[bb_k][bb_j]) then
                dist[bb_i][bb_j] ← min(dist[bb_i][bb_k], dist[bb_k][bb_j])
            end if
```

Fig. 82 Adoption of the Floyd-Warshall algorithm to detect shortest paths within EA models

As already mentioned, the Floyd-Warshall algorithm does not compute the shortest paths themselves but rather the shortest distances between the pair of two nodes from where the shortest paths can be obtained easily.

The resulting adjacency matrix holds the distances between two edges, respectively between two building blocks of the model. The distance measurements then can be transferred into EA relationships. The distance directly indicates the relation type. Relations not lying on the shortest path are discarded.

The above algorithm is embedded into a new transformation task type: **Contradict Paths**. Input parameters are two sets of building blocks. The set of building blocks typically represents building blocks on different layers of the EA. All building blocks residing on the path between the two sets of building blocks are contradicted and removed from the dataset by bypassing the building blocks first and then by removing these building blocks including their incoming and outgoing relations. This type of transformation task is not a specialisation of the existing transformation tasks of the DICE method base. Thus, it has to be directly specialised from the top level transformation task type.

DICE for EAA keeps the information on used relationships and considers the information for assessing the quality of the dataset. This is not only true for derived relationships but also for relationships existing in the original datasets.

7.4 Heterogeneous Metamodels and EA Data Schemas

In case of heterogeneous metamodels (and EA schemas), the metamodels/schemas but also the EA data inferred from them have to be aligned. The approach is similar to the one

Application of DICE in the Fields of EAA

introduced in section 7.2.2 where strategies to calculate the equivalency of building blocks (instance level) have been introduced. The introduced equivalency functions also work for the metamodels constituting concepts, i.e. the modelling classes, the relation classes and their attributes. In cases where no metamodel mappings such as the TOGAF-Archimate-Mapping of Fig. 61 exist, equivalency functions support the generation of such mappings. With the introduction of the addition function (see section 5.3.2.3), the calculation of variable similarity has already been brought up. Taking this requirement one step further, equivalency functions need to be in place for all metamodel elements. Thus, these are required for modelling classes as well as relation classes. (Gill and Qureshi 2015) introduce such an approach and demo the feasibility by merging the partial metamodels of BPMN and Archimate. Adapted for DICE, in contrast to the record linkage on building block level (see section 7.2.2), the metadata of the datasets have to be aligned. Main input is the semantic definitions of the observable (U^{SEM}) units and the variables (V^{SEM}). Two datasets carry equivalent observable units when their semantic metadata are equivalent:

$U_{datasetA}^{SEM>name} \approx U_{datasetB}^{SEM>name}$ where similarity is calculated via syntax, semantics and structural analysis. The same holds true for variables. The equivalency of two variables from the different data objects is also calculated using the variables metadata: $V^{SEM>name}$. To obtain more reliable results, besides the name, additional metadata such as $V^{PROC>measureUnit}$ might be considered to obtain the equivalency measures by calculating the cosine distance as introduced in section 7.2.2. To sum up, the following techniques are propagated in DICE for EAA:

- Syntactical Analysis: The metamodel concept's names are compared. In the case of modelling classes the metadata $U^{SEM>NAME}$ are compared. For attributes and relations the variable names $V^{SEM>NAME}$ are used for calculation of equivalency.
- Semantical Analysis: Again, the concept names are the input for the equivalency calculations. Synonyms are generated using adequate ontologies in a first step. Token-based similarity functions calculate the similarity in a next step.
- Structural Analysis: The concept's neighbouring concepts are considered. Neighbours are all modelling classes within the metamodel adjacent to a certain modelling class. Again, a token-based similarity function is used to calculate the similarity.

Application of DICE in the Fields of EAA

- **Attribute-based Analysis:** In the case of attribute-based analysis, besides the name of a modelling class, the names of its variables are taken into account for the token-based similarity calculation.

Table 13 illustrates such a mapping on level of modelling classes for the metamodels of Archimate and BPMN. The overall similarity is calculated by a function weighing the structural, semantical and syntactical similarity measurements obtained from the concepts names and their set of variables.

Table 13 Excerpt of overall similarity of Archimate and BPMN concepts (Gill and Qureshi 2015)

Archimate	BPMN	Overall Similarity
Data Object	Data Object	0,6
Artifact	Artifact	0,63
Business Role	Partner Role	0,3
Business Event	Boundary Event	0,3
Business Collaboration	Collaboration	0,18

Where no mappings are available, such a transformation task type is valuable for performing the required addition transformations. DICE presents the final outcome in a metamodel comparison sheet. Based on this input, the data engineer decides on the next steps. DICE restrains from an automated addition based on the calculated mapping, as the results from (Gill and Qureshi 2015) show that semantic interpretation and decision on these mappings in most cases will be required to avoid biased results.

7.5 Summary

In this section the characteristics of EA data are discussed. The classical EA data is object-oriented. EA data in the form of EA models can typically be described in the form of a directed graph. Consequently, many of the EA-situational transformation tasks types deal with graph-based transformations. Examples are: hierarchical dependencies of architecture elements caused by type-instance relations or by whole-part relations that are used to describe building blocks in more detail, i.e. on different strata. In addition, peculiarities of EA data, such as time-aspects, versioning issues, etc. are addressed.

In the data preparation phase of an EAA endeavour, the EA descriptions (models) need to be “normalised”. The data typically requires the same level of granularity. Time-aspects need to be clearly defined. The required transformation task types to create a sound data basis are

Application of DICE in the Fields of EAA

discussed. All of these EAA situational transformation task types are integrated into DICE by specialising its foundational transformation task types.

In this vein, the section serves a twofold purpose. Firstly, the EAA-specific requirements on method chunks are described from the nature of EA data and from the requirements of the EA analysis techniques. Adequate method chunks fulfilling these requirements are introduced. Secondly, it is shown how the scenario-agnostic foundational DICE concepts can be specialised to support situational method engineering.

8 Evaluation Based on Prototypical Implementation

DICE is a design science artefact. Thus, it can be evaluated against the broad spectrum of evaluation criteria presented by (Prat, Comyn-Wattiau and Akoka 2014). The criteria: structural **consistency**, **efficacy** and **validity** gain priority in the evaluation. The evaluation is structured in accordance with these criteria into three parts.

Firstly, the structure of DICE is evaluated with a focus on the evaluation criteria of structural **consistency**. According to (Prat, Comyn-Wattiau and Akoka 2014), “*consistency of structure is internal consistency*”. Internal consistency of DICE is evaluated with a strong focus on the DICE metamodel. To this end, the DICE metamodel is implemented on the metamodeling platform ADOxx, building the basis for the DICE modeller, a modelling environment that makes the design and transformation processes possible and holds the metadata defined in the structural part of the DICE metamodel.

Secondly, the DICE algorithms are implemented on ADOxx and *R statistics* to demonstrate the DICE **efficacy**. Efficacy is “*the degree to which the artefact produces its desired effect*”, (Prat, Comyn-Wattiau and Akoka 2014). The prototypical implementation is used to demonstrate the feasibility of DICE. The DICE macro level algorithms (see section 5.3.1) are implemented directly in ADOxx, whereas the micro level algorithms (see section 5.3.2) are implemented on R statistics. Together they allow for concurrent transformation of data and metadata (the “desired effect”). To this end, a significant part of the DICE transformation tasks and algorithms has been implemented.

Finally, **validity** of DICE is demonstrated by an illustrative scenario. *Validity* is defined as “*the degree to which the artefact works correctly*” (Prat, Comyn-Wattiau and Akoka 2014). In this connection a standard EA dataset is used. The dataset comprises Archimate models residing in the Archimate Model Exchange File Format (see section 7.1.3). The data is loaded and transformed executing DICE workflows designed in the DICE modeller for the purpose of evaluation.

8.1 Structural Consistency and Efficacy - DICE Prototype Based on ADOxx

ADOxx is a metamodeling platform provided by the OMiLab (Open Models Laboratory), a collaborative environment for modelling method engineering (Götzinger, Miron and Staffel

Evaluation Based on Prototypical Implementation

2016). Based on ADOxx, the DICE modelling environment is implemented. The modelling environment “DICE modeller” is intended to design executable DICE workflows in a graphical manner. Through its integration with *R statistics*, the required transformations are conducted on the datasets and the corresponding metadata are generated. *R statistics* is a programming language and environment to statistically explore data (Dalgaard 2008).

The implementation of the DICE modeller follows the principles of agile modelling method engineering (Karagiannis 2015). In the course of implementing a modelling method, a number of design decisions have to be taken. Typically, in a first step the platform-independent metamodel defined in the conceptional base (see section 5.2) has to be translated into a platform-specific metamodel. The metamodel represents the core of the modelling language. The implemented modelling classes have to be extended with a graphical notation and must be embedded in a modeltype (model kind, see section 2.2). In a second step, the required mechanisms and algorithms for machine-processing of the models have to be implemented.

In the course of this, how the features of the chosen metamodeling platform support the implementation of the method has to be considered. This aspect is denoted as the “*conceptualisation of a modelling method*” in (Karagiannis 2015). The OMiLab lifecycle introduced by (Visic et al. 2015) defines the method conceptualisation lifecycle in five phases: (1) create, (2) design, (3) formalize, (4) develop and (5) deploy/validate. Fig. 83 provides an overview.

Phase (1) focuses on knowledge acquisition and requirements analysis (see sections 4, 6.1 and 7.1). The subsequent design phase covers the specification of the metamodel, the procedure model and the required mechanisms and algorithms (see section 5). The *formalise phase* targets the unambiguous definition of the method and makes it possible to share the results with a scientific community, see e.g. (Grossmann and Moser 2016). The *develop phase* concentrates on the implementation of the modelling method, often based on a metamodeling platform. The *deployment phase* usually involves domain experts who evaluate the modelling method and its implementation. The following sections focus on *development* and *deployment* of DICE.

Evaluation Based on Prototypical Implementation

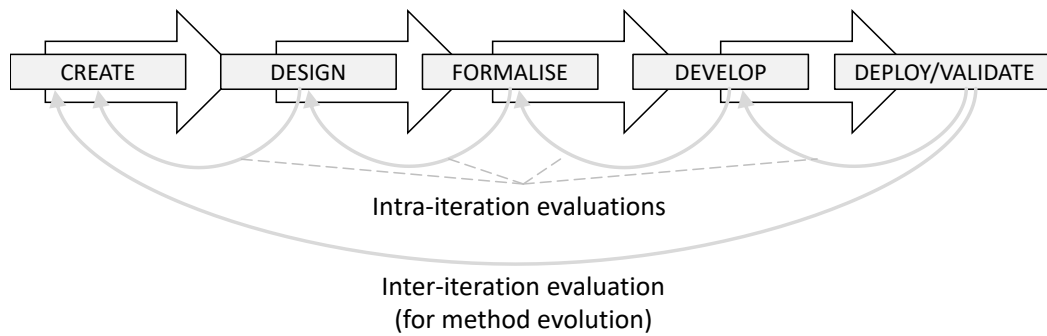


Fig. 83 Method conceptualisation lifecycle, adapted from (Visic et al. 2015)

8.1.1 Architecture

The DICE 2.1 prototype, a revised version of the DIBA prototype⁸, consists of a three-layer architecture. The top-layer represents the modelling environment providing the user interface for the data engineer. It is made up of the **modelling component**, which makes it possible to graphically design the DICE workflows. The main functions of the modelling environment are: (1) to design and manage DICE workflows, (2) to specify the metadata and transformation parameters, (3) to generate the transformation statements to be executed on the input datasets/data objects and (4) to store and visualise the calculated metadata. The modelling classes offered to design a DICE workflow are derived from the DICE metamodel, i.e. the conceptualisation base which is discussed in section 5.2. The implementation makes use of the predefined concepts defined in the meta2model of ADOxx.

Via the **external coupling component** of ADOxx, *R statistics*, an environment for statistical computing, is integrated. *R statistics* represents the BI-tier. Utilising AdoScript, the ADOxx-internal DSL (domain specific programming language), the modelling environment communicates with the BI-tier. The modelling environment uses the designed workflow and the user-inputted parameters to create the executable (platform-specific) code and triggers the required transformations/calculations, which are performed in R. R, transforms the input datasets and calculates the metadata. The latter are returned to the modelling environment for analysis by the data engineer.

The data layer is the bottom layer. It holds the datasets, the processing information (DICE workflows) and the metadata. More precisely, the workflow definitions and the metadata are

⁸ See <http://austria.omilab.org/psm/content/diba/info>, access: 02.04.2017

Evaluation Based on Prototypical Implementation

managed in the ADOxx database; the input datasets (data level) are initially retrieved from various source systems, and after transformation, stored in flat files from where they can be accessed for further processing. Via its access functions, the DICE prototype is capable of accessing data from social media platforms (e.g. from twitter, see www.twitter.com), from cloud storages such as google drive (<https://www.google.com/intl/de/drive/>) or from local shares.

Fig. 84 illustrates the DICE architecture. For reasons of overview, central components of the ADOxx metamodeling platform, such as the ADOxx web interface, the ADOxx simulation component and the ADOxx model analysis component are intentionally omitted. The ADOxx modelling component (1) is used for graphically designing the DICE workflows. Within the modelling subsystem (CORE), the models are interpreted and the platform-specific R code is assembled automatically from the DICE workflows and the parameters of its constituent transformation tasks. The generated R code (3) is embedded in the execution code (implemented in AdoScript) and triggers the execution (4) of the code in R statistics. R statistics retrieves the input data and performs the calculations using R packages (libraries of reproducible R code). Via R packages, additional application components (6) such as the WordNet 2.1 application, a lexical database of English delivering the synsets introduced in section 7.2.3.2 are integrated. R delivers the calculated metadata (7a), charts on the metadata (7b) and the transformed datasets (7c), which are stored in flat files (csv) for further usage.

Evaluation Based on Prototypical Implementation

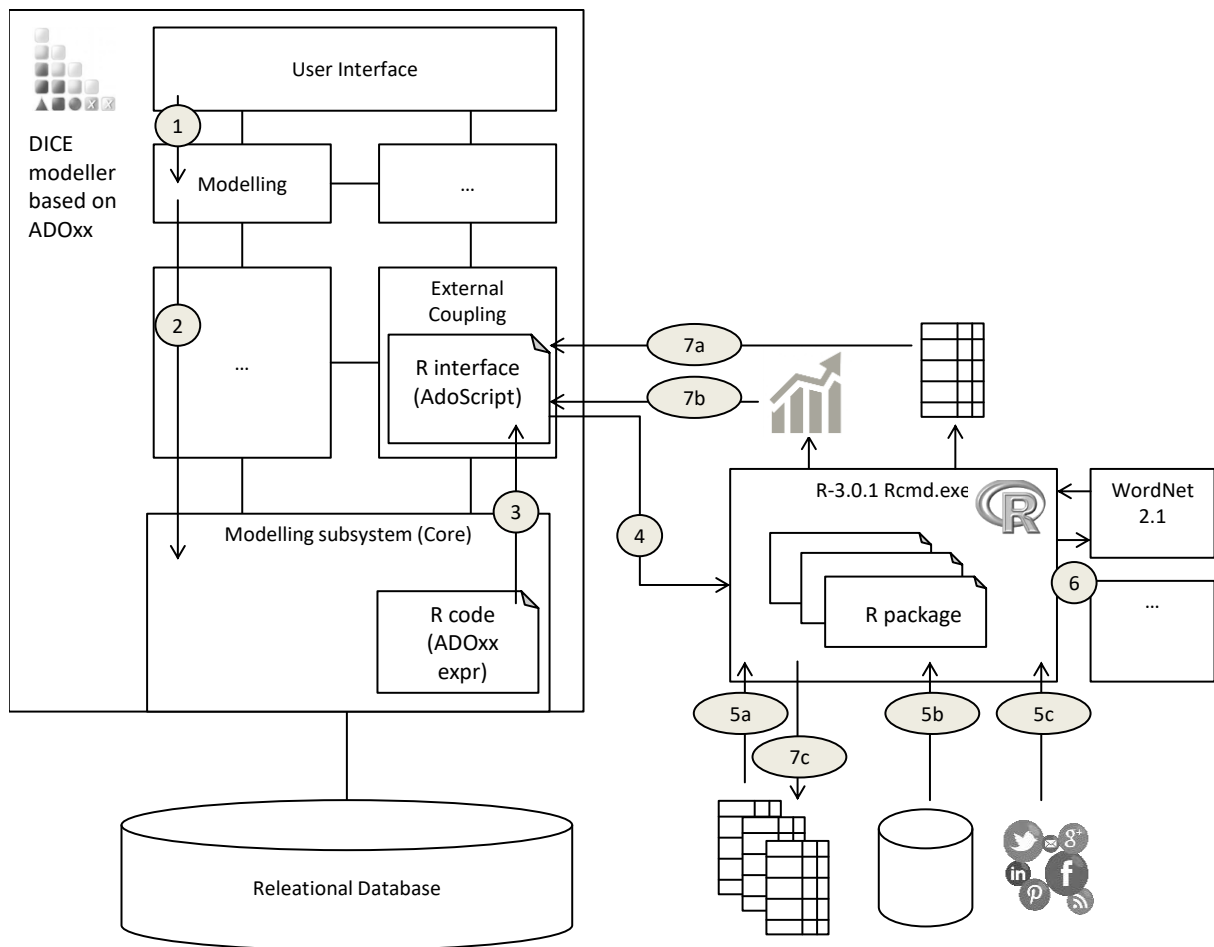


Fig. 84 The DICE architecture

8.1.2 Development

Central to the implemented modelling method is the modeltype "DICE workflow". Modeltypes correspond to the concept "model kind" as introduced in section 2.2. It contains all relevant modelling classes and relation classes required to: (1) design the DICE workflow, (2) specify the required data transformations and (3) visualize the generated metadata. The metamodel of the DICE workflow is represented in Fig. 85. The DICE concepts are specialised from the meta2model foundational constructs of the ADOxx meta2model, which is discussed in detail in (Karagiannis et al. 2016).

Evaluation Based on Prototypical Implementation

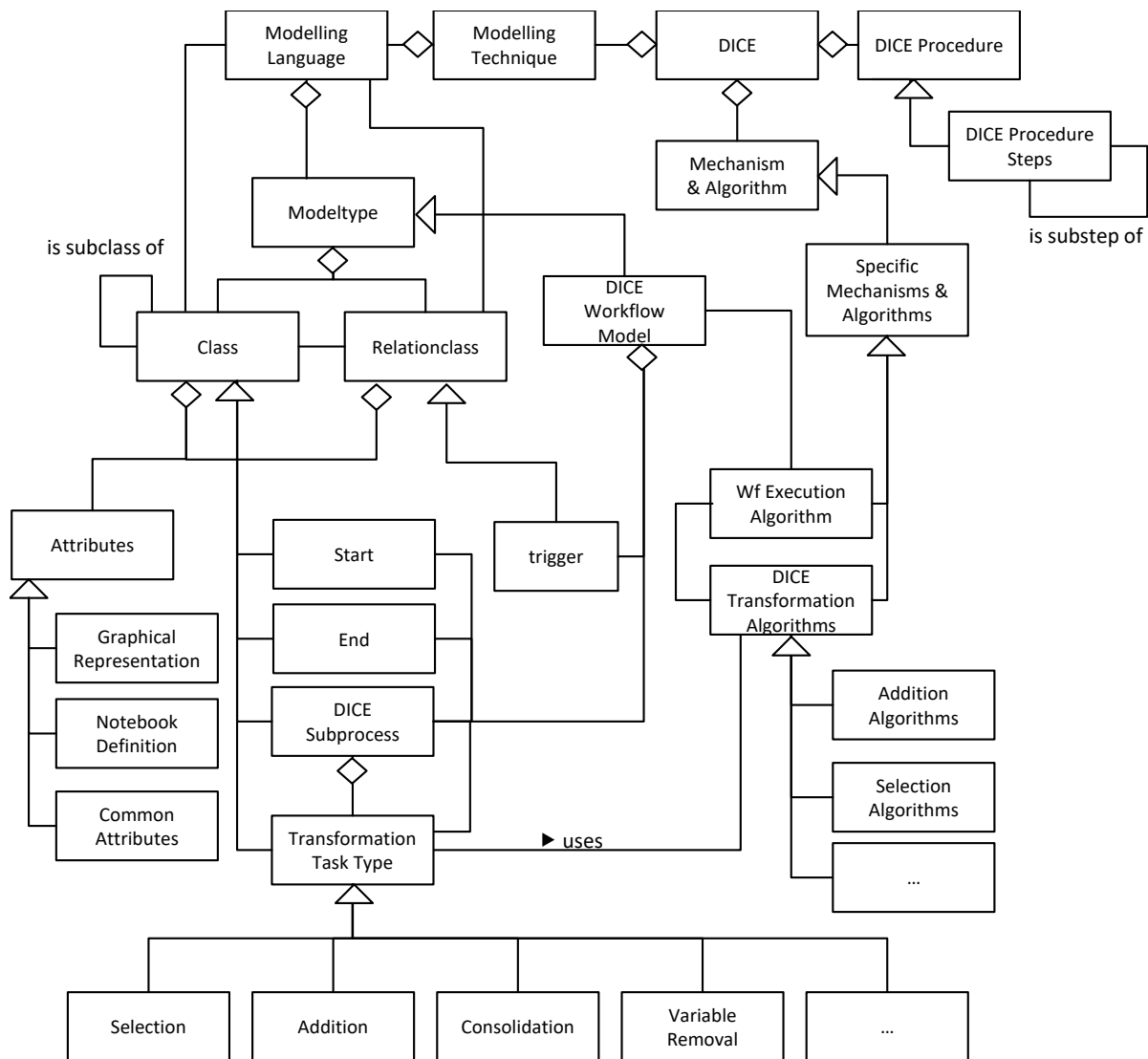


Fig. 85 Metamodel of the DICE workflow (excerpt)

The implemented metamodel deviates from the metamodel of the conceptualisation base, as it had to be transferred into the ADOxx platform concepts. From the foundational concept “class”, method-specific classes are specialised. On the top-most level one can find the modelling classes of the DICE meta structure (of utmost importance, the transformation task types). From these task types EAA-specific transformation task types are derived.

As can be seen in Fig. 85, the separation of behaviour and structural concepts has been abandoned. In the platform-specific metamodel, the DICE transformation tasks also carry the metadata of the product fragments. This design decision has been made for reasons of usability. In this way, the DICE workflows remain clear and the modeller does not have to bother about maintaining consistency between transformation tasks and input/output datasets.

Evaluation Based on Prototypical Implementation

The DICE modeltype composes the modelling classes: start event, end event, DICE subprocess, the atomic transformation task types and the situational transformation task types supporting EAA scenarios. In addition, the EAA-situational transformation task type “Load Archimate Models” is illustrated. The transformation task types correspond to the concept of DICE method chunks of the DICE meta structure. The relation class “triggers” is instantiated from the foundational concept “relation class” and makes the connection of the transformation tasks possible, thus, defining the order in which the transformation tasks are processed in runtime.

For each of the modelling classes a graphical notation is defined. This graphical notation is defined in foundational attribute of type “GraphRep” utilising the ADOxx-specific DSL. Fig. 86 illustrates an example.

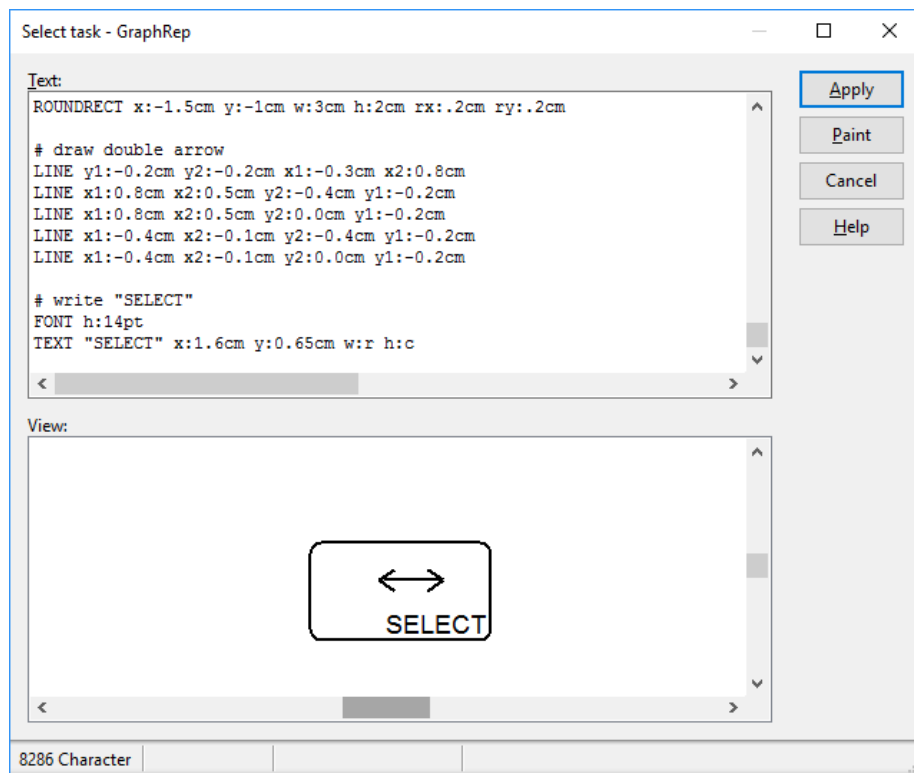


Fig. 86 GraphRep definition of the transformation task type “Selection”

In addition, each modelling class comprises a set of attributes. “Attributes” are defined as *“properties attached to the semantic definition of modelling concepts”* (Karagiannis et al. 2016). Attributes are made visible and editable via the so-called *notebook* assigned to the modelling classes. A notebook represents the property box of a modelling object. Fig. 87 shows an example screenshot of the transformation task type *selection* and the corresponding

Evaluation Based on Prototypical Implementation

notebook definition based on the AttrRep configuration. The AttrRep attribute is one of the foundational attributes which carries the notebook definitions of a modelling class. It defines the chapter structure and the set of visible attributes of the class. The AttrRep configuration is interpreted during runtime of the modelling tool and creates the notebooks according to the configurations.

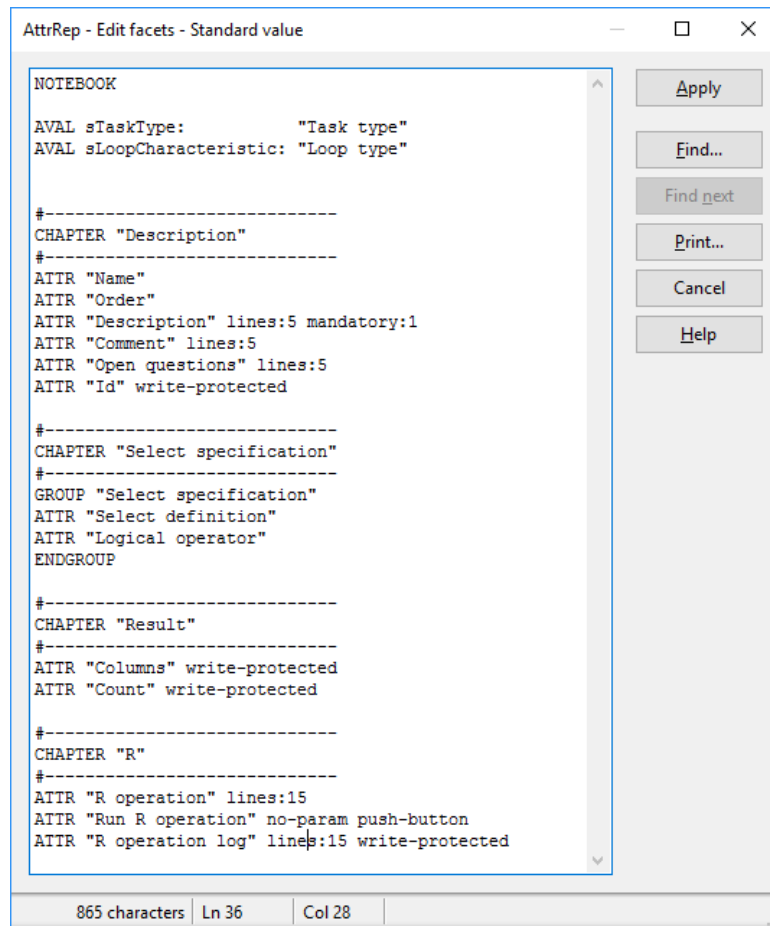


Fig. 87 AttrRep - ADOxx notebook definition

In ADOxx, attributes can be of a simple type, such as string, integer and double but also of more complex types such as records which are classes of their own and in turn compose attributes. Records are presented in the form of tables within the notebooks. An example can be seen in Fig. 88 where the record attribute "Metadata" carries some of the DICE metadata of the data object.

Evaluation Based on Prototypical Implementation

(1) Get Business Objects from CSV (GET) (Get task) - Metadata Check

	Variable	Check Type	Check Definition	Result	
1	Confidentiality	Syntax	^(very high high medium low very low)\$	0,714286	^
2	Objects	Completeness		1,000000	
3	Confidentiality	Completeness		1,000000	
4	Integrity	Completeness		0,857143	
5	Availability	Completeness		0,857143	
6	Description	Completeness		0,857143	
7	Objects	Currentness	2017-10-07	1,000000	
8	DataSet	Uniqueness	Objects,Classes	1,000000	

Fig. 88 DICE processing metadata and quality metadata (per variable)

The concept of ADOxx *expressions* is used to assemble the PSM (platform-specific model) code which can be executed in R statistics. Expressions are used to collect all required input data from the attributes and to integrate the attribute values into the generic R code held directly in the expression attribute. ADOxx expressions are comparable to the concept of formulas within spreadsheet software and in DICE serve as the code generator for generating the R-specific transformation code. During run-time, it delivers the (platform-specific) code to be executed on the BI-tier. To this end, the DICE algorithms (see e.g. section 5.3.2) have to be implemented in platform-specific R code where the parameters (e.g. path to source files) specified in the attributes (edited via the ADOxx notebooks) are dynamically embedded. Fig. 89 illustrates an example of an expression.

Evaluation Based on Prototypical Implementation

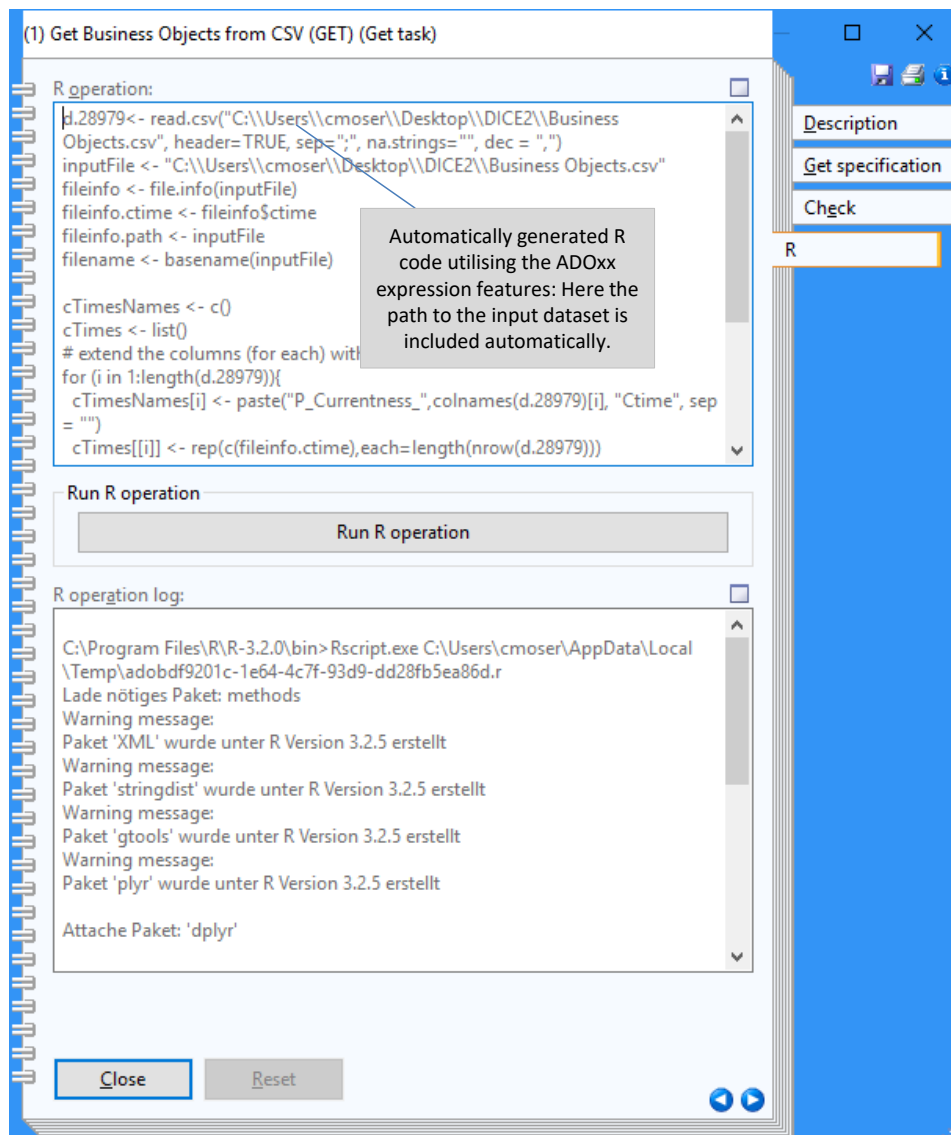


Fig. 89 Example of an ADOxx expression assembling R code

AdoScript is used to execute the DICE workflow. It triggers the required calculations by integrating with the BI-tier (via external coupling APIs). Via AdoScript, the calculations are triggered in R statistics and the results (more precisely, the calculated metadata and graphical charts) are written back into the attributes of the transformation tasks.

Basically, objects of three types are created by execution of a single transformation task: the output dataset, the corresponding metadata object and charts depicting the metadata. Whereas the metadata and supporting graphical charts are stored in the modelling environment (returned via API from R), the output datasets are stored in flat files structured in the defined tabular layout in csv-format.

Evaluation Based on Prototypical Implementation

To sum up, the prototypical implementation of DICE covers the following part of the DICE method chunks implemented in the form of transformation task types:

- **Load datasets:** This transformation task type is capable of loading datasets residing in csv-format structured according to the DICE main structure, respectively in tabular form.
- **Load Archimate Models:** The transformation task type makes it possible to load files residing in the Archimate Model Exchange File Format. Via this transformation, task type files can be loaded and transferred into the DICE standard format as discussed in section 7.2.1. It creates two tables: one holding the observable units represented by the various Archimate objects which can be of different types (e.g. business process, application component, technology component) and a second one holding all relations between the objects including the type of the relations.
- **Selection:** From a given dataset, observable units can be selected by specifying selection criteria that can be based on data and metadata of observable units. In the context of EA, the selection transformation task is capable of selecting observable units based on their attribute values but also based on metadata, such as total quality (U^{QUAL}), currentness ($U^{QUAL>currentness}$), accuracy ($U^{QUAL>accuracy}$) and completeness ($U^{QUAL>completeness}$).
- **Variable Removal:** From a given dataset, variables and their metadata can be removed.
- **Addition:** The transformation task type makes the concatenation of two datasets possible. The input datasets are horizontally integrated. Columns that are not present in all input datasets are added to the output datasets.

The **initialization function** is a fixed component of all the mentioned transformation task types. In its current version (DICE 2.1), it calculates the quality indicators:

- completeness,
- accuracy,
- currentness,
- uniqueness and
- total quality

Evaluation Based on Prototypical Implementation

of properties, variables and the entire dataset. For each of the named quality aspects, at minimum one of the DICE atomic quality indicators is implemented. See the following section for more details and for a concrete example.

8.2 Evaluating Validity - Illustrative Scenario Based on the DICE Prototype

The illustrative scenario presented in the subsequent sections shows an example where architecture descriptions are combined with operational data. For reasons of overview and understandability, the focus is to demo the DICE features. Real world examples will be more complex and challenging to be integrated and cleansed.

8.2.1 Input Datasets

For the illustrative example, three input datasets are considered. The first dataset is based on the Archimate 2.1 example files, officially issued by the Opengroup to demo the Archimate Model Exchange File Format⁹. Table 14 Quantity structures of the input datasets provides an overview of the quantity structures of contained building blocks and relation classes. The overall structure of this file format has been discussed in section 7.2.1.

Table 14 Quantity structures of the input datasets

Number of modelling classes	12
Number of building blocks	156
Number of relation class types	23
Number of relations	574

The second dataset is a table which comprises objects of modelling class “business objects”, i.e. observable units of type “business objects”. Besides a variable specifying the names of the observable units, it comprises the three typical variables for specifying security requirements: (1) “Confidentiality”, (2) “Integrity” and (3) “Availability”. All of these variables are categorical variables with the admissible values “very high”, “high”, “medium”, “low” and

⁹ The file can be downloaded from <http://www.opengroup.org/xsd/archimate>, access: 08.03.2017.

Evaluation Based on Prototypical Implementation

“very low”. The property values of these variables are set randomly. Some of the values do not comply with these syntactical requirements, such that $P^{QUAL>accuracy>syntax}$ and $V^{QUAL>accuracy>syntax}$ will calculate insufficient quality values. Table 16 illustrates a subset of this dataset. Some of the contained observable units have syntactically similar names and others have equal names as compared to the data objects contained in the formerly mentioned dataset.

The third dataset holds operational data. Table 15 shows a snapshot of the dataset. The dataset holds a list of transactions, the related customer and the transaction date. As can be seen, the contained transaction types have the same name as the business objects in the previous datasets.

Table 15 Operational Data – Simplified Dataset

Objects	FirstName	LastName	Date
[...]	[...]	[...]	[...]
Car Insurance Policy	Henry	Fonda	13.12.2016
Insurance Request	Harrison	Ford	14.12.2016
Insurance Policy	John	Wayne	15.12.2016
Damage Claim	Lex	Barker	16.12.2016
Car Insurance Policy	Gary	Cooper	17.12.2016
Car Insurance Policy	Dean	Martin	18.12.2016
Car Insurance Policy	Ronald	Reagan	19.12.2016
Insurance Request	Gregory	Beck	20.12.2016
Insurance Policy	Clint	Eastwood	21.12.2016
[...]	[...]	[...]	[...]

8.2.2 DICE Process Instance

One major advantage of DICE is that each of the conducted transformation steps results in datasets that can be examined in detail. For purposes of demonstration, the two datasets are loaded into the DICE prototype, transformed, integrated and cleansed. Fig. 90 depicts the DICE process.

Evaluation Based on Prototypical Implementation

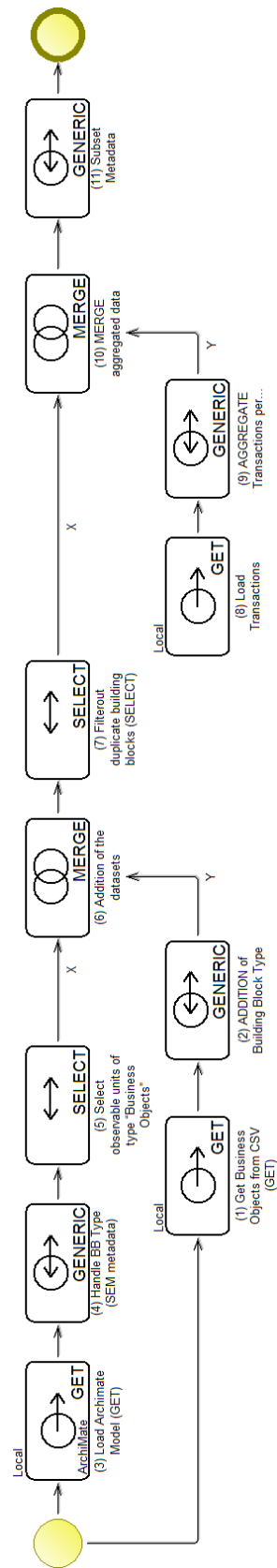


Fig. 90 Exemplary DICE workflow mapped in the DICE modeller

Evaluation Based on Prototypical Implementation

(1) Get business objects from CSV (GET): In this transformation task, the Dataset 2, which is already structured in the DICE preferred data structure, is loaded and subsequently initialized. Table 16 shows the initial structure of the input dataset including some of its observable units.

Table 16 Initial structure of input dataset

Objects	Description	Confidentiality	Integrity	Availability
Customer	A person or organization using the services of Archisurance.	high	medium	low
Car Insurance Policy	NA	medium	High	
Insurance Request	NA	123	High	low
Insurance Policy	A document detailing the terms and conditions of a contract of insurance.	high	medium	low
Customer File	NA	medium	High	very low
Damage Claim	Formal notification of a loss or damage that might be covered by the policy.	no entry		low
Client	A person or organization using the services of Archisurance.	medium	high	very low
...

After performing the initialisation ($T^{initialise}$), the dataset is enriched with the following metadata (columns):

- P_LOG_ChangeDate_Objects,
P_LOG>ChangeDate_Description,
P_LOG_ChangeDate_Confidentiality,
P_LOG>ChangeDate_Integrity,
P_LOG_ChangeDate_Availability:

Each of these columns holds the last change date of properties. As initially no information on the last change date on property level is available, the change date is

Evaluation Based on Prototypical Implementation

retrieved from the input file's metadata (provided by the operating system in use) and imputed on property level. This metadata needs to be stored on the level of single properties because due to merge operations, tractability of the values will suffer if held on dataset level only.

- P_LOG>**Sourcepath_Objects**,
P_LOG>**Sourcepath_Description**,
P_LOG>**Sourcepath_Confidentiality**,
P_LOG>**Sourcepath_Integrity**,
P_LOG>**Sourcepath_Availability**:

Each of these columns holds the storage location of the input datasets. The storage location is specified by the data engineer via the DICE notebook.

- P_LOG>**Filename_Objects**,
P_LOG>**Filename_Description**,
P_LOG>**Filename_Confidentiality**,
P_LOG>**Filename_Integrity**,
P_LOG>**Filename_Availability**:

Each of these columns holds the filename of the source file.

- P_QUAL>**completeness>missingValue_Objects**,
P_QUAL>**completeness>missingValue_Description**,
P_QUAL>**completeness>missingValue_Confidentiality**,
P_QUAL>**completeness>missingValue_Integrity**,
P_QUAL>**completeness>missingValue_Availability**:

Each of these columns holds the atomic quality indicator on property level required for calculation of completeness of a variable. In $\forall^{PROC>valueDomain>missingValue}$ the syntactical patterns for missing values are defined. Missing values in the given case are specified via a regular expression: $\wedge(|no\ entry)\$$

The exemplarily specified regular expression evaluates whether a property value is entirely empty or carries the value “no entry”.

- P_QUAL>**accuracy>syntax_Confidentiality**,
P_QUAL>**accuracy>syntax_Integrity**,

Evaluation Based on Prototypical Implementation

P_QUAL>accuracy>syntax_Availability:

In the example given, syntax compliance has to be ensured for the variables confidentiality, integrity and availability only. To this end, the syntax definition is defined in the form of the following regular expression:

^(very high|high|medium|low|very low)\$

which has to be defined by the data engineer in

V_PROC>valueDomain>Syntax_Confidentiality,

V_PROC>valueDomain>Syntax_Integrity,

V_PROC>valueDomain>Syntax_Availability.

Thus, the three variables are considered as categorical variables with the admissible values “very high”, “high”, “medium”, “low” and “very low”.

- **P_QUAL>currentness>maximumAge_Objects, etc.:**

Each of these columns holds information on whether the property values fulfil the currentness criteria "maximumAge" which is specified in the metadata **V_PROC>currentness>maximumAge_Objects, etc.**

All properties evaluated with “pass” fulfil the criteria:

P_LOG>changeDate ≥

V_PROC>currentness>maximumAge_[variableName].

- **D_QUAL_currentness,**
D_QUAL_accuracy,
D_QUAL_completeness

Each of these columns holds the averaged quality values obtained from the variable quality indicators of the respective quality category. As an example:

D_QUAL_completeness is obtained from the averaged quality values of

D_QUAL_completeness_Objects,

D_QUAL_completeness_ObjectIDs,

D_QUAL_completeness_Description,

D_QUAL_completeness_Criticallity,

D_QUAL_completeness_Integrity and

D_QUAL_completeness_Availability.

- Finally, **U_QUAL** holds the total quality per observable unit.

Evaluation Based on Prototypical Implementation

Fig. 91 shows a screenshot of the DICE modeller. It illustrates parts of the V_PROC and V_QUAL metadata.

(1) Get Business Objects from CSV (GET) (Get task)

Metadata Check:			
	Variable	Check Type	Check Definition
1	Confidentiality	Syntax	^(very high high medium low very low)\$
2	Objects	Completeness	
3	Confidentiality	Completeness	
4	Integrity	Completeness	
5	Availability	Completeness	

Result: 0,714286, 1,000000, 1,000000, 0,857143, 0,857143

Quality:

Completeness: 0,914286

Syntax: 0,714286

Currentness: 1,000000

Uniqueness: 1,000000

Columns:			
	Column name	Data type	Description
1	Classes	integer	
2	Objects	integer	
3	Description	integer	
4	Confidentiality	integer	
5	Integrity	integer	

Count: 7

Close Reset

Fig. 91 Example of DICE metadata specification in ADOxx

(2) Addition of building block type (ADDITION)

In this transformation step the column “U_SEM_type” ($U^{SEM>type}$) is added. It specifies the type of the contained building blocks (business objects). In addition, the resulting data object is initialised.

(3) Load Archimate model (GET) is of type “Load Archimate model”. It is capable of loading datasets residing in the Archimate Exchange File Format and transforms it into the required tabular structure. In its finalisation step, the resulting dataset is initialised, i.e. the metadata are calculated and assigned to the data object. The required parametrisation to conduct the transformation and the subsequent initialisation is done in the same way as in (1).

Evaluation Based on Prototypical Implementation

For a better understanding, Fig. 92 illustrates some of the building blocks contained in the dataset in the form of an Archimate viewpoint.

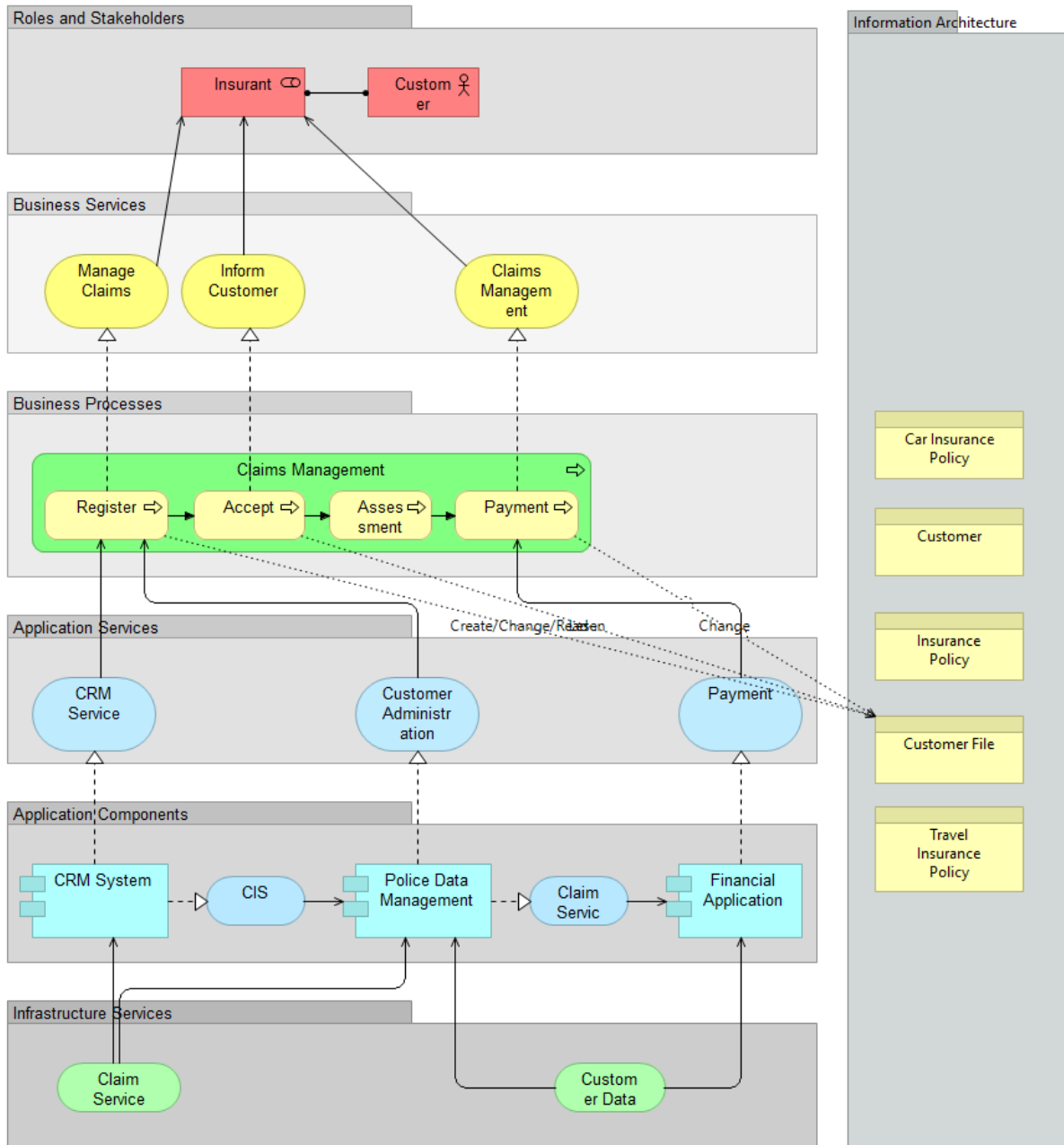


Fig. 92 Exemplary subset of the building blocks in the dataset

(4) Handle building block type (SEM metadata): The input data object already carries the $U^{SEM>type}$ (see section 5.2.2) of the contained building blocks. The column only has to be renamed to correspond to the DICE metamodel and to be identified as DICE metadata.

Evaluation Based on Prototypical Implementation

(5) Select observable units of type “Business Objects”: The building blocks catalogued within the Archimate file are of different types (modelling classes). In the case at hand, the emphasis lies on building blocks (observable units) of type “business objects”. To this end, the input dataset delivered from the preceding transformation task is reduced to hold observable units of modelling class “business objects” only. This is achieved by selection of those observable units which are typed as business objects (variable “Classes”). Again, as a last step the dataset is initialized. The input parameters for conducting the initialisation do not change, as solely observable units have been removed.

(6) Addition of the datasets: In this step both data objects are concatenated via vertical integration and initialized. By appending one data object to the other, the superset of their variables is created. Equal variables of the two datasets are bound together. In the DICE prototype, equivalency is established by comparing the variable’s names, i.e. the metadata ($V^{SEM>name}$). Variables only existing in one of the data objects are added (including the variable metadata).

(7) Filter out duplicate building blocks: Due to the fact that the sample input datasets (specified in section 8.2.1) carried similar/equal observable units, the data object resulting from the previous append transformation has to be cleansed by consolidating equal/similar observable units. From a pair of similar observable units, the one with the better overall quality is kept, whereas the other one is deleted from the data object. In section 7.2.3.2, four different types of similarity analysis are introduced: syntactical, semantical, neighbouring and attribute-based. In its initial version the DICE prototype handles only the syntactical equivalency check, which is conducted during data object initialisation, after each of the performed transformations. In the course of this, equal building blocks are labelled being equal, more precisely: the observable unit with the lower overall quality indicator from a pair of equal observable units is labelled as duplicate. Thus, duplicate observable units can be filtered out easily by conducting a selection on the metadata $U^{QUAL>uniqueness>inconsistency}$ and $U^{QUAL>uniqueness>redundancy}$, both metadata indicating duplicates.

(8) Load Transactions: In this step again data from csv is loaded. This time, it is transactional data, i.e. data from an operative source. After loading the data it is initialised as in any transformation step.

Evaluation Based on Prototypical Implementation

(9) Aggregate Transactions per Type: In this step the individual transactional observable units are consolidated per type. Each type represents a business object such as customer, policy, insurance policy, etc. as defined in the aforementioned datasets. During aggregation of the data the frequency of the individual transactions is calculated. In a final step the dataset is initialised.

(10) Merge aggregated data: This transformation is a perfect example, where operational data and architecture description are merged. It holds the final data object. With the available metadata, together with the DICE process itself, data provenance can easily be traced back to the initial input datasets.

Fig. 93 again depicts the DICE process. This time it additionally shows the resulting metadata $D^{QUAL>syntax}$, $D^{QUAL>currentness}$, $D^{QUAL>uniqueness}$ and $D^{QUAL>completeness}$.

Evaluation Based on Prototypical Implementation

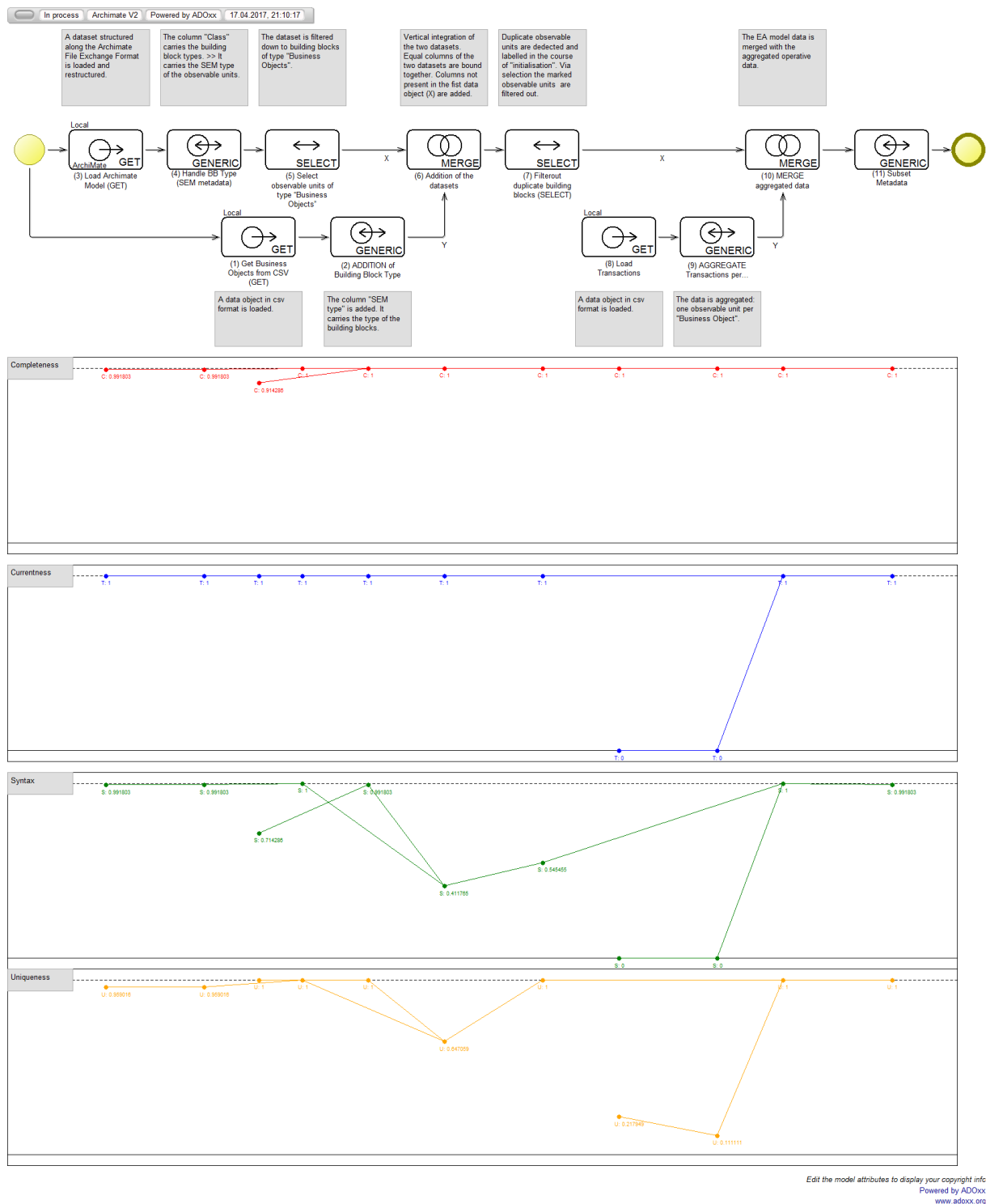


Fig. 93 DICE process incl. quality line charts

The charts show the development of the summative qualities currentness, completeness, uniqueness and syntax of the resulting data objects after performing the transformations. In

Evaluation Based on Prototypical Implementation

this way, data engineers obtain an overview of quality issues and the quality impact after performing the transformations. Fig. 94 shows a concrete example.

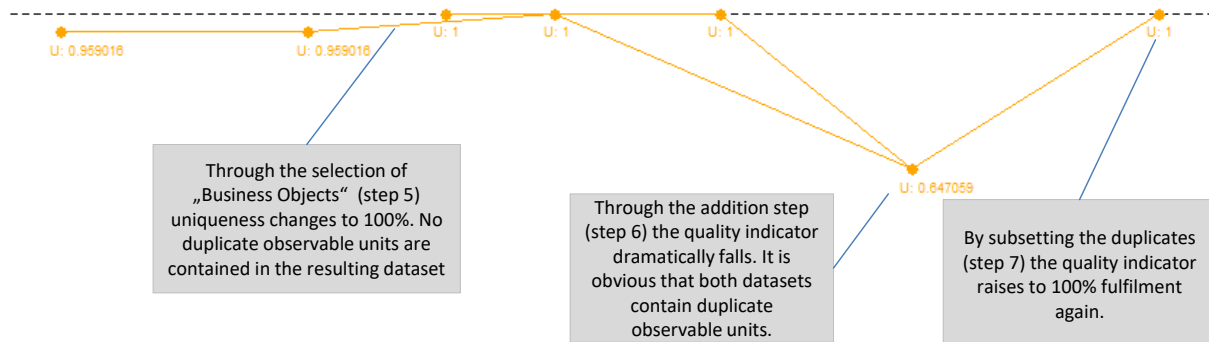


Fig. 94 Development of the quality indicator "uniqueness" throughout the process

The quality indicator “uniqueness” changes after performing the transformation step. Initially the quality is about 95% for the first dataset. After subsetting the business objects, the quality rises to 100%. Obviously, there have not been any duplicate “business objects” in the initial dataset. Through addition to the second dataset the quality indicator falls dramatically. This is an indication that there are many duplicates in the set of business objects of both datasets. By subsetting the unique business objects in the last step of the DICE process the quality rises again.

8.2.3 Examination

The DICE prototype covers a representative subset of the DICE capabilities. In the following section the most important components of the DICE prototype are summarised.

Modeltype “DICE process”: The modeltype makes modelling of an entire DICE process possible. It is used to demonstrate the DICE workflow capabilities. DICE transformation tasks can be assembled to executable DICE workflows as discussed in section 5.3.1.

DICE method base: The method base is made up of the core DICE transformation tasks: load, restructure, selection, addition, variable removal, reclassification and consolidation. From these atomic transformation tasks, some specialised method chunks are derived to demonstrate the DICE capabilities of derivation of specialised method chunks, which is considered to be a core feature of situational methods. An example is the method chunk “Merge Transformation”, which is an assembly of the atomic transformation task types “addition” and “selection”. An example of a tailored (situational) EA specific transformation

Evaluation Based on Prototypical Implementation

task is the “Load Archimate model” transformation, where the DICE method base is enhanced with a situational method chunk, providing features to transform input datasets residing in the Archimate Exchange File Format into the DICE horizontal layout.

DICE initialisation: The DICE initialisation is an example of a process fragment, which is reused in all method chunks (transformation task types). With these process fragments the prototypical implementation places strong emphasis on the metadata aspects of DICE.

From the set of introduced meta data, representative examples are implemented. Some of them are based directly on the raw data on data level; others require logistical or processing metadata as an input.

- $V^{QUAL>completeness}$: This quality metadata scores the completeness of a variable within a dataset.
- $V^{QUAL>currentness}$: This quality indicator requires *logistical* metadata as an input. More precisely, information on the latest update of the property values is taken into account for its calculation. The *logistical* metadata is retrieved from the input file’s metadata (provided from the operating system in use). Besides last change date, other logistical meta data such as source path and file names of input files are retrieved and subsequently propagated onto property level of the resulting output data object. Keeping records on these meta data strongly supports data provenance; at any step within the transformation process the source stays traceable.
- $V^{QUAL>accuracy}$: For demonstrating the DICE features in the context of accuracy, quality indicators are calculated using *processing* metadata as an input. In the concrete case, the accuracy is calculated from the atomic quality indicators $V^{QUAL>accuracy>syntax}$ and $V^{QUAL>accuracy>dataType}$. Both of these atomic quality indicators require processing metadata as an input that has to be specified by the data engineer.
- In addition, $D^{QUAL>uniqueness}$ is calculated (based on $U^{QUAL>uniqueness>redundancy}$ and $U^{QUAL>uniqueness>inconsistency}$) to demonstrate the DICE record linkage capabilities.

8.3 Summary

The evaluation of DICE has been performed in a two-step approach. (1) DICE has been implemented based on a metamodeling platform, a typical environment for implementing situational methods such as DICE. In a second step, the validity of DICE has been

Evaluation Based on Prototypical Implementation

demonstrated by applying DICE in a concrete case: a representative dataset from the application domain of enterprise architecture management was loaded and transformed for upcoming BI modelling phases. The demo revealed that the compulsory simultaneous transformation of datasets and their metadata is possible, leading to meaningful data objects as an input for further analysis and decision making.

9 Conclusion and Outlook

9.1 Conclusion

Acknowledging the significance of data integration and cleansing mechanism in the fields of enterprise architecture management and with a wider perspective of management domains based on modelling, this thesis introduces DICE. DICE stands for Data Integration and Cleansing Environment. It is a domain-agnostic method for data integration and cleansing and is intended to support the data preparation phases of business analytics endeavours.

To this end, it borrows from the fields of workflow management, statistical metadata management and data mining. Concepts from the fields of workflow management are used to design executable workflows that allow for design and execution of data transformations. Concepts from the fields of statistical metadata management, with particular focus on processing and quality metadata, are incorporated to keep the conducted transformations traceable. The quality of the resulting datasets is known and quality improvement potentials can easily be derived. The same holds true for data provenance aspects; sources of the data remain fully transparent. Comprehensibility is also assisted through incorporation of concepts from the fields of workflow management. By graphically designing the DICE workflows, the conducted data transformations are fully documented.

Concepts from workflow management do not only contribute in terms of clarity. Their excitability grants reproducibility. In this way, DICE workflows represent deployable production processes that perform the actual data transformations and calculate the metadata with minimal human interaction.

To conduct the actual data transformation, DICE draws from the abundance of algorithms and techniques from the fields of data mining. It defines and implements the atomic transformation tasks immanent in many of these techniques and enriches these transformation tasks with concepts of statistical metadata. According to this design principle, the transformation task types represent method chunks comprising features for data transformation and concurrent calculation of metadata.

Conceptualised as a situational method, DICE can be adapted to the given situation by means of metamodelling, i.e. it can be tuned to domain/project specific situations. Enterprise architecture management is considered as such a domain specific situation. By analysing the

Conclusion and Outlook

peculiarities of EA models and EA related data, requirements for the specific method are drawn. The DICE method base (initially holding generic method chunks) is extended with domain specific method chunks. In the fields of EA, examples such as the loading of models residing in typical EA formats, restructuring of the EA data, filtering of EA data etc. are introduced. The aim of this action was twofold: (1) the adaptability of DICE by means of meta-modelling is demonstrated and (2) powerful transformation tasks to be applied in EA scenarios are presented.

DICE and DICE for EAA are evaluated in a three-staged approach. Firstly, the metadata calculations are demonstrated based on an implementation of the structural part of the DICE metamodel. To this end, parts of the metamodel and some of the algorithms to calculate quality indicators are implemented. The metamodel is instantiated into a model representing a common dataset, its constituent parts and some of the quality metadata, which are calculated automatically.

DICE is evaluated in regard to efficacy by implementing DICE based on the metamodeling platform *ADOxx* and *R statistics*, an environment for statistical computation. With this prototypical DICE modeller, the feasibility of DICE can be proven. Some of the transformation task types (the main method chunks) have been implemented, making possible the design and execution of DICE workflows. It could be demonstrated that the concurrent transformation of data instances and metadata is possible with a minimum of human intervention.

To assess the resulting data quality (the outcome of the transformations), an illustrative scenario is presented. For this purpose, a typical EA dataset, initially residing in an EA standard format, is loaded and transformed using the EA-situational transformation task types.

9.2 Outlook

In this section the possible directions of future work are briefly discussed.

In general, inter-iteration evaluation as recommended by (Visic et al. 2015) and (Karagiannis 2015) has to be performed to enrich the DICE method base with reusable method fragments and to possibly streamline the existing method fragments. In the context of DICE for EAA, additional EA specific method fragments can be defined. This obviously has to be done in the context of specific EA projects ensuring applicability and foremost

Conclusion and Outlook

reusability of the transformation task types. To this end, the prototypical implementation of DICE has to be extended to cover the full set of transformation task types and metadata.

DICE places a strong focus on data integration and cleansing of structured data. Taking a wider perspective, the preceding steps of data loading and restructuring, which are only fractionally covered in this thesis, need to be explored. Especially when it comes to DICE for EAA, the data loading and restructuring of semi-structured data gains importance. Thus, extending DICE by derivation of method chunks that focus on data restructuring of semi-structured data is one important direction for future work. During the course of writing this thesis, some promising results have been obtained by analysing and restructuring text corpora of ISO standards in the fields of security management. The main aim was to extract and highlight the most important compliance requirements, i.e. processes and tasks to be performed to ensure compliance. The resulting tasks and processes are intended to be used to ensure and measure compliance of EAs. However, neither atomic transformation tasks extracted nor the required metadata supporting the results on data level have been defined.

In addition, it has to be noted that EAA the integration of EA and BA is still in its infancy. Thus, besides the data driven-approach discussed in this thesis, research has to be conducted on the actual management scenarios in these fields. What kind of data is required for adequate management decision making in the context of the phases of EA management cycles such as the TOGAF ADM has to be investigated in more detail.

10 Annex

10.1 DICE Metamodel

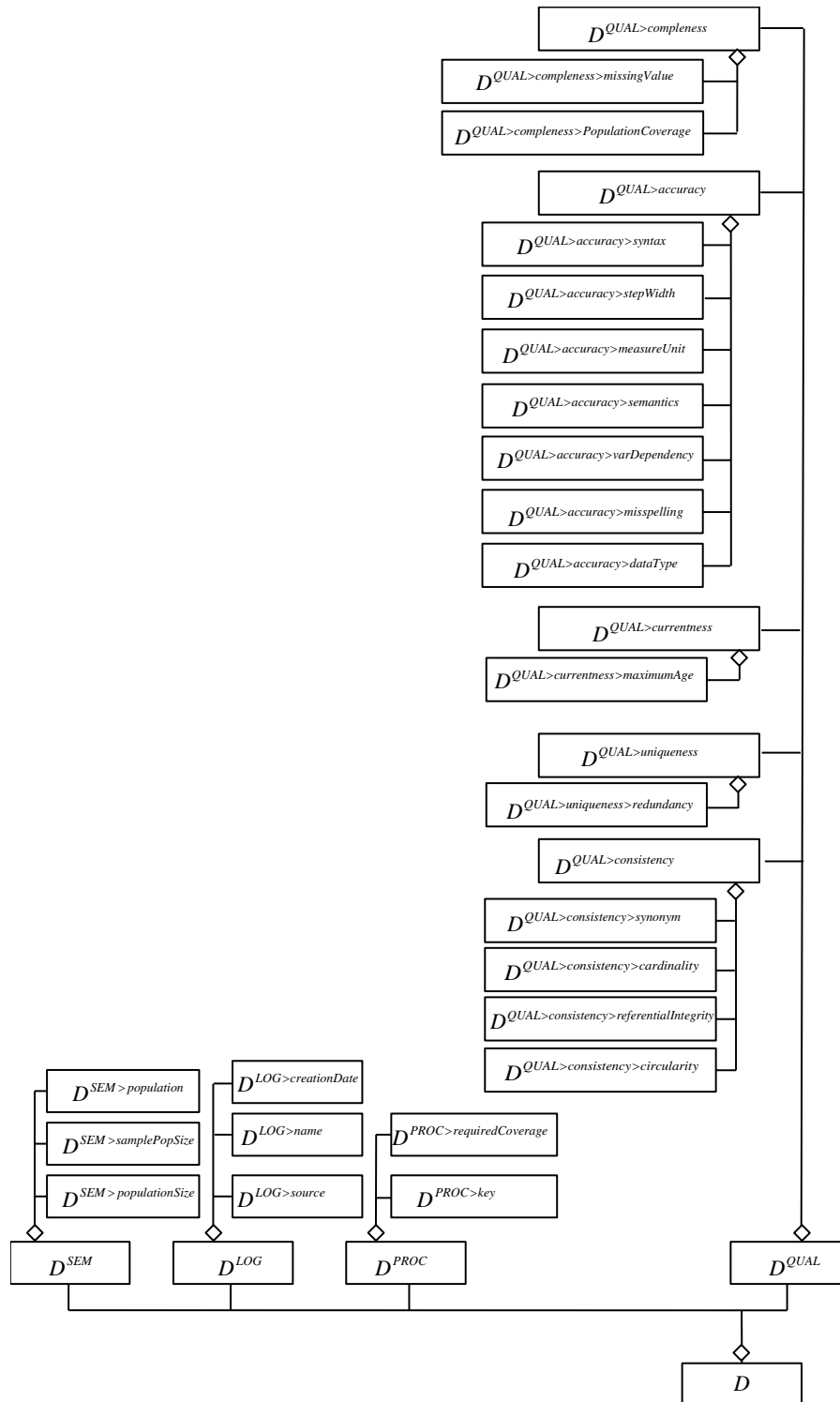


Fig. 95 DICE - Structural part of the metamodel - subset "Dataset"

Annex

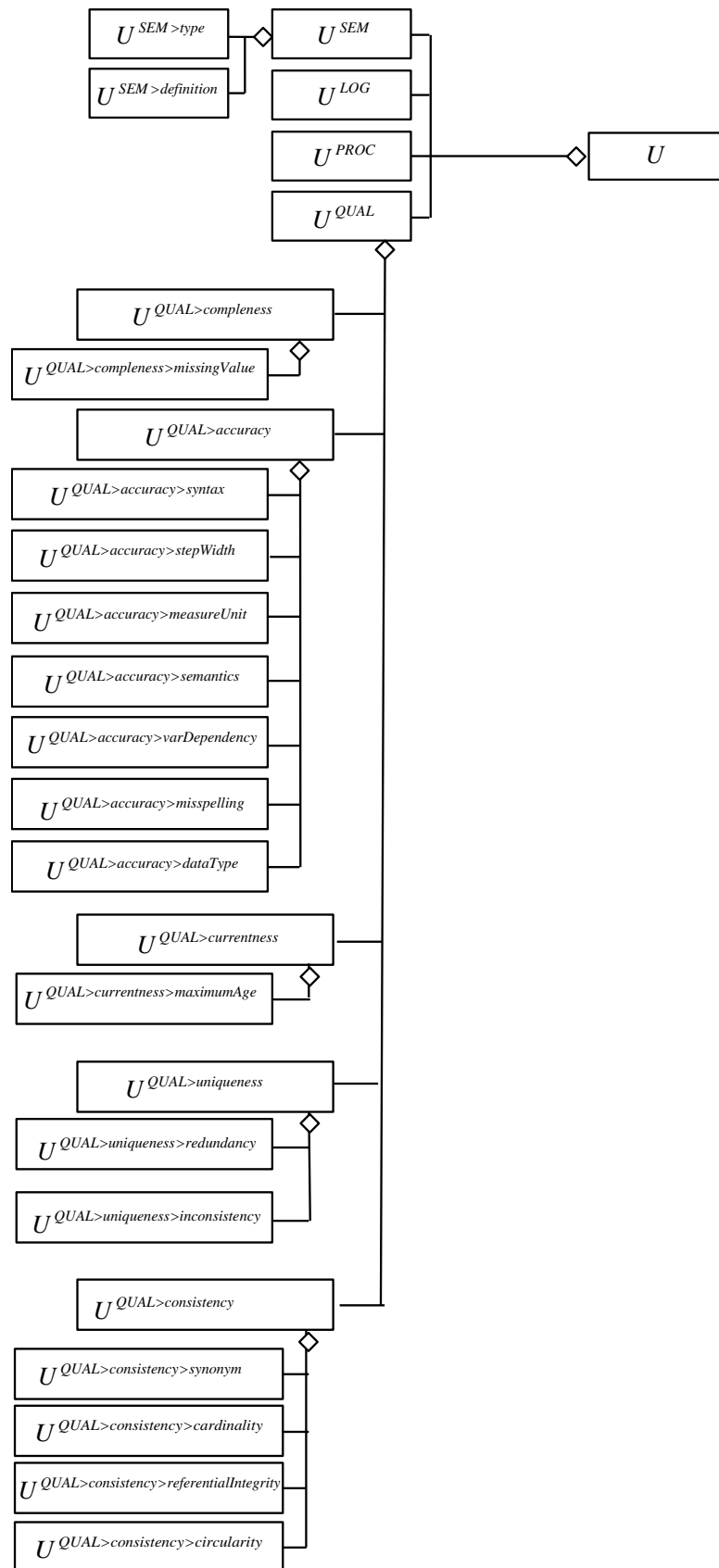


Fig. 96 DICE - Structural part of the metamodel - subset "Observable Unit"

Annex

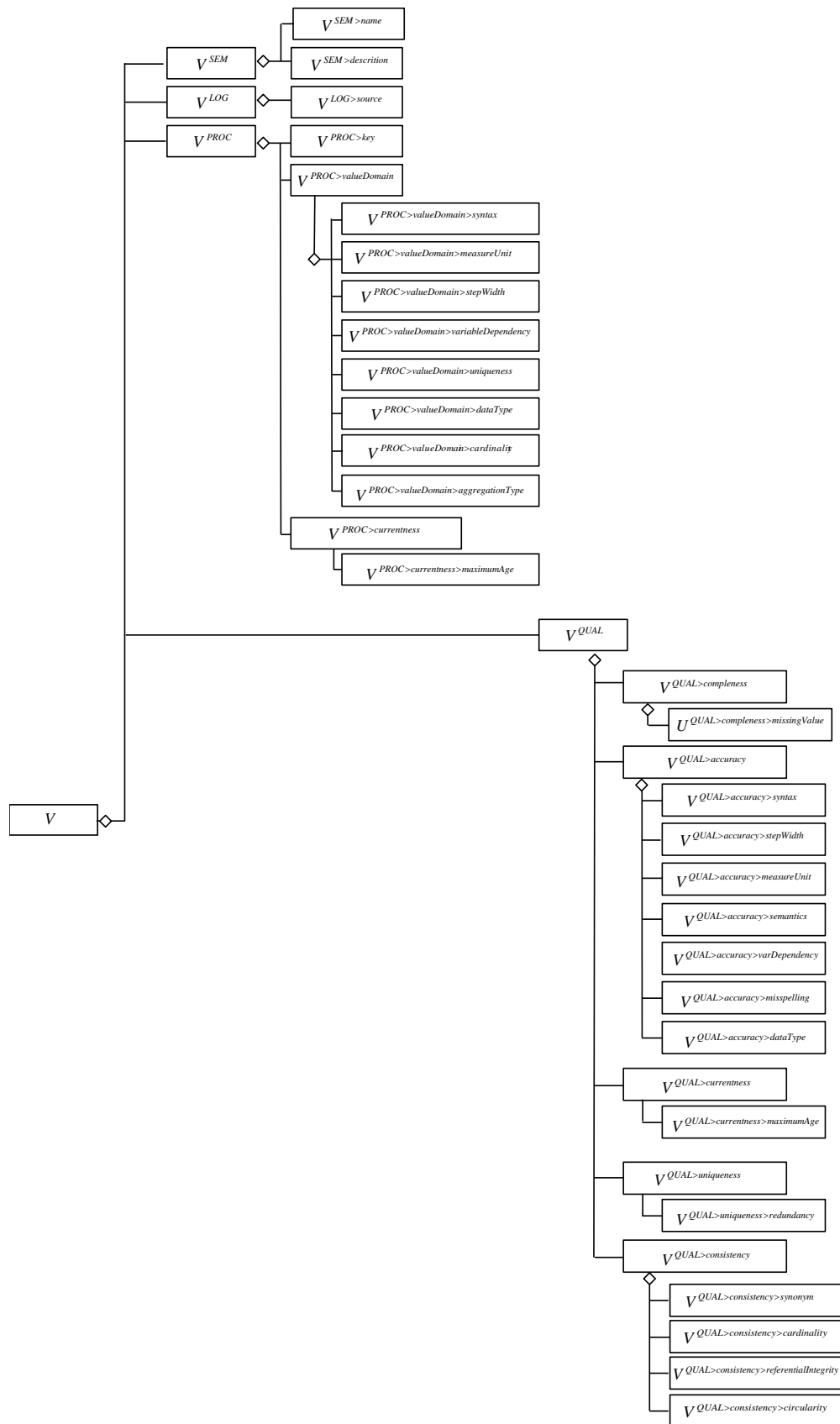


Fig. 97 DICE - Structural part of the metamodel - subset "Variable"

Annex

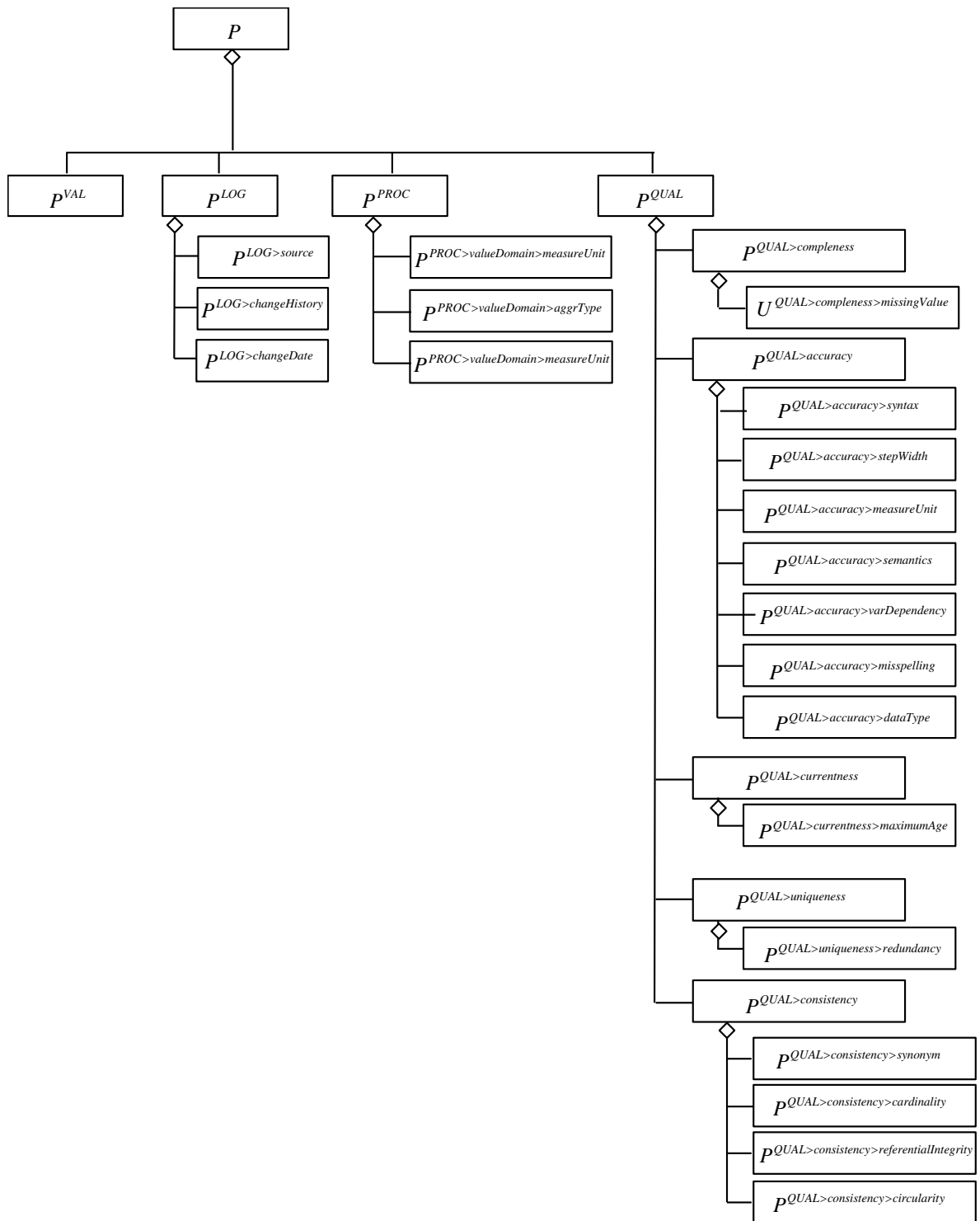


Fig. 98 DICE - Structural part of the metamodel - subset "Property"

10.2 DICE Transformations

10.2.1 Initialisation

```

input:
   $D^{A,(i)}$  or  $O^{(i)}$            % a raw dataset or a composite input data
                                % object which requires recalculation %

output:
   $O^{(o)}$                        % the composite output data object %

Begin
  if initial input is  $D^{A,(i)}$            % if raw data  $O^{(i)}$  is
                                          % constructed first %

  input  $D^{SEM}$ ,  $D^{LOG}$ ,  $D^{PROC}$            % manual input %
  input  $U^{SEM}$                            % manual input %
  input  $V^{SEM}$ ,  $V^{PROC}$                    % manual input %
  propagate  $D^{LOG}$  to  $P^{LOG}$ 

  % iterate through all variables & through their quality indicators %
  for  $v = 1$  to  $|V|$ 
    for  $indicator = 1$  to  $k$ 
      retrieve indicator-relevant requirements from  $V_v^{PROC}$ 
      % iterate through observable units %
      for each  $u = 1$  to  $|U|$ 
        % match property value against requirement %
        if calculable calculate  $P_{uv}^{QUAL>indicator}$ 
          else input  $P_{uv}^{QUAL>indicator}$  manually
        % calculate  $V_v^{QUAL>indicator}$  from the set of qualities
        % per observable unit %
        calculate average value from  $P_v^{QUAL>indicator}$ 

      % iterate through observable units %
      for  $u = 1$  to  $|U|$ 
        for  $v = 1$  to  $|V|$ 
          % average quality per variable of observable unit %
          calculate  $P_{uv}^{QUAL>total}$  from set of  $P_{uv}^{QUAL}$ 

        % calculate total quality of  $U_u$ : average of  $P_{uv}^{QUAL>total}$  %
        calculate  $U^{QUAL>total}$ 

```

Annex

```

% total quality per indicator on dataset level  $D^{QUAL>indicator}$  %

calculate average  $V^{QUAL>indicator}$ 

% total quality of dataset %

calculate  $D^{QUAL}$ 

return  $O^{(i)}$ 

end

```

10.2.2 Selection

```

input:
  selection criteria
   $O^{(i)}$  % the composite input data object %

output:
   $O^{(o)}$  % the composite output data object %

begin
% an empty output data object is created %
create empty  $O^{(o)}$ 
% iterate through all observable units %
for  $u = 1$  to  $|U|$ 
  % selection criteria can refer to data & metadata %
  if selection criteria match
    % transfer metadata of  $U_u$ , property values of  $U_u$ 
    incl. property metadata: %
    copy  $U_u^{(i)}$  into  $O^{(o)}$ 
  % transfer metadata %
  copy  $V^{(i),SEM}$ ,  $V^{(i),LOG}$ ,  $V^{(i),PROC}$  into  $O^{(o)}$ 

  % conduct reinitialisation: %
  initialize  $O^{(o)}$ 
return  $O^{(o)}$ 

end

```

10.2.3 Addition

```

input:
 $O^{(i_k)}$  and  $O^{(i_l)}$            % two data objects %
output:
 $O^{(o)}$                        % the composite output data object %

begin
  create empty  $O^{(o)}$ 
  %  $O^{(i_k)}$  holds the leading metadata %
  copy  $O^{(i_k)}$  to  $O^{(o)}$ 
  for  $v = 1$  to  $|V^{(i_k)}|$  in  $O^{(i_k)}$ 
    for  $z = 1$  to  $|V^{(i_l)}|$  in  $O^{(i_l)}$ 
      calculate similarity degree of  $\langle V_v^{(i_k)}, V_z^{(i_l)} \rangle$ 
      if variable similar
        % append all properties of  $V_z^{(i_l)}$  and their metadata %
        append  $P_z^{(i_l)}$  to corresponding variable in  $O^{(o)}$ 
      else
        append  $V_z^{(i_l)}$  as new variable
      % conduct reinitialisation %
    initialize  $O^{(o)}$ 
  return  $O^{(o)}$ 
end

```

10.2.4 Variable Removal

```

input:
 $V_r^{(i)}$            % the to be removed variable %
 $O^{(i)}$            % the composite input data object %
output:
 $O^{(o)}$            % the composite output data object %

begin
  create empty  $O^{(o)}$ 
  for each  $V_v^{(i)}$  in  $O^{(i)}$ 

```

Annex

```

    % transfer all remaining variables incl. their properties %
    if  $V_v^{(i)} \neq V_r^{(i)}$  copy  $V_v^{(i)}$  to  $O^{(o)}$ 

    copy  $D^{(i),SEM}$ ,  $D^{(i),PROC}$ ,  $D^{(i),LOG}$ 

    % conduct reinitialisation %
    initialize  $O^{(o)}$ 

    return  $O^{(o)}$ 

end

```

10.2.5 Reclassification

```

input:
     $O^{(i)}$  % composite input data object %
     $V_r^{(i),SEM>name}$  % name of variable which requires reclassification %
    %
    reclassification rules % mapping table or conversion function %
output:
     $O^{(o)}$  % the composite output data object %

begin
    create empty  $O^{(o)}$ 
    for all  $V_v^{(i)}$  in  $O^{(i)}$ 
        if  $V_v^{(i)} \neq V_r^{(i)}$  copy  $V_v^{(i)}$  to  $O^{(o)}$ 
        for  $u = 1$  to  $|U|$ 
            if  $P_{ur}^{(i),VAL}$  does not comply with  $V_r^{PROC>accuracy>syntax}$ 
                % after reclassification processing meta data are adapted %
                reclassify  $P_{ur}^{(i),VAL}$  && adapt  $P_{ur}^{(i),PROC}$ 
            % conduct reinitialisation %
        end
        initialize  $O^{(o)}$ 
    end
    return  $O^{(o)}$ 

end

```

10.2.6 Consolidation of Observable Units

```

input:
 $O^{(i)}$            % composite input data object %
 $\{V^{(i),SEM>name}\}$  % array on variables holding properties to be compared %
similarity threshold

output:
 $O^{(o)}$            % the composite output data object %

begin
  create empty  $O^{(o)}$ 
  % compare all observable units within the dataset with each other %
  for  $u = 1$  to  $|U|$ 
    for  $i = 1$  to  $|U|$ 
      calculate similarity between  $U_u$  and  $U_i$ 
      if similarity > threshold
        mark  $U_u$  and  $U_i$  for consolidation
      consolidate similar pairs and copy to  $O^{(o)}$ 
      copy non-similar observable units to  $O^{(o)}$ 
      copy variable metadata to  $O^{(o)}$ 
      copy dataset metadata to  $O^{(o)}$ 

    % conduct reinitialisation %
  initialize  $O^{(o)}$ 

  return  $O^{(o)}$ 
end

```

10.3 Publications by the Author

Table 17 provides an overview of the articles published by the author. Most of the articles have been written with the influence and help of the Research Group Knowledge Engineering, led by Prof. Dr. Dimitris Karagiannis.

Table 17 Publications by the author

Author	Title	Year	Outlet
Christoph Moser, Franz Bayer, Dimitris Karagiannis	ITIL: Modellgestützte Umsetzung mit ADOIT	2004	Optimiertes IT-Management mit ITIL. Victor, F., Günther, H., Vieweg (2004)
Christoph Moser, Franz Bayer	IT Architecture Management: A Framework for IT- Services.	2007	In: Proceedings of the Workshop on Enterprise Modelling and Information Systems Architectures, Desel J., Frank U. (eds.) Lecture Notes in Informatics – Gesellschaft für Informatik (GI), Klagenfurt, Austria
Christoph Moser, Matthias Winklhofer, Christian Kuplich	Business Objectives Compliance Architecture Framework	2008	Proceedings of Modellierung 2008, Kühne T., Reisig W., Steimann F. (eds.) Lecture Notes in Informatics– Gesellschaft für Informatik (GI), Berlin, Germany.
Christoph Moser, Franz Bayer	Einführung von ISO 20000 - ein prozessbasierter Ansatz	2008	In: ISO 20000: Praxishandbuch für Servicemanagement und IT- Governance, Andenmatten M. (eds.), Symposion Publishing.
Christoph Moser, Stefan Junginger,	Some Process Patterns for Enterprise Architecture	2009	In: Conference: Software Engineering 2009 -

Annex

Mathias Brückmann, Klaus-Manfred Schöne	Management.		Workshopband, Fachtagung des GI-Fachbereichs Softwaretechnik, Münch J., Liggesmeyer P. (eds.), Kaiserslautern, Germany.
Robert Winter, Jan vom Brocke, Peter Fettke, Peter Loos, Stefan Junginger, Christoph Moser, Wolfgang Keller, Florian Matthes, Alexander Ernst	Patterns in der Wirtschaftsinformatik	2009	In: Wirtschaftsinformatik 51, no. 6
Christoph Moser, Daniel Fürstenau, Stefan Junginger	A Method for Integrating EAM and BPM	2010	In: 2nd European Workshop on Patterns for Enterprise Architecture Management (PEAM2010), Paderborn, Germany.
Christoph Moser, Lutz Kirchner	Integration von Prozessmanagement und Unternehmensarchitektur- Management – Konzepte und Vorgehensweisen zum Business IT Alignment	2013	In: Prozessmanagement für Experten, Bayer F., Kühn H. (eds), Springer Gabler Verlag.
Tobias Rausch, Michael Puncochar, Kai-Helmut Eckert, Christoph Moser	Technische Umsetzung von Geschäftsprozessen	2013	In: Prozessmanagement für Experten, Bayer F., Kühn H. (eds), Springer Gabler Verlag.
Karagiannis,	Compliance evaluation	2012	In: International Conference on

Annex

Dimitris, Christoph Moser, Arash Mostashari	featuring heat maps (CE-HM): a meta-modeling-based approach		Advanced Information Systems Engineering. Springer Berlin Heidelberg, Germany.
Wilfried Grossmann, Christoph Moser	Big Data—Integration and Cleansing Environment for Business Analytics with DICE	2016	In: Domain-Specific Conceptual Modeling - Concepts, Methods and Tools, Karagiannis D., Mayr H. C., Mylopoulos J., Springer

11 Bibliography

- Aalst, Wil MP van der. 1996. "Structural Characterizations of Sound Workflow Nets." *Computing Science Reports* 96 (23): 18–22.
- Abiteboul, Serge. 1997. "Querying Semi-Structured Data." In *Database Theory—ICDT'97*. Springer.
- Abraham, Ralf, José Tribolet, and Robert Winter. 2013. "Transformation of Multi-Level Systems—theoretical Grounding and Consequences for Enterprise Architecture Management." In *Advances in Enterprise Engineering VII*, 73–87. Springer.
- Addicks, Jan Stefan, and Hans-Jürgen Appelrath. 2010. "A Method for Application Evaluations in Context of Enterprise Architecture." In *Proceedings of the 2010 ACM Symposium on Applied Computing*, 131–36. ACM.
- Ahmed, Rafi, and Shamkant B Navathe. 1991. "Version Management of Composite Objects in CAD Databases." In *ACM SIGMOD Record*, 20:218–27. ACM.
- Aier, Stephan, Christian Riege, and Robert Winter. 2008a. "Classification of Enterprise Architecture Scenarios-An Exploratory Analysis." *Enterprise Modelling and Information Systems Architectures* 3 (1): 14–23.
- . 2008b. "Unternehmensarchitektur - Literaturüberblick Und Stand Der Praxis." *Wirtschaftsinformatik* 50 (4): 292–304.
- Aier, Stephan, and Robert Winter. 2009. "Virtual Decoupling for IT/Business Alignment—conceptual Foundations, Architecture Design and Implementation Example." *Business & Information Systems Engineering* 1 (2): 150–63.
- Allison, Paul D. 2002. "Missing Data: Quantitative Applications in the Social Sciences." *British Journal of Mathematical and Statistical Psychology* 55 (1): 193–96.
- Andersen, Peter, and Andrea Carugati. 2014. "Enterprise Architecture Evaluation." In *Proceedings of the 8th Mediterranean Conference on Information Systems*.
- Antunes, Gonçalo, Marzieh Bakhshandeh, Rudolf Mayer, José Borbinha, and Artur Caetano. 2013. "Using Ontologies for Enterprise Architecture Analysis." In *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*, 361–68. IEEE.
- Baker, Michael J. 2000. "Writing a Literature Review." *The Marketing Review* 1 (2): 219–47.
- Band, Iver, Henk Jonkers, Erik Proper, Dick Quartel, and Mike Turner. 2015. "Using the TOGAF 9.1 Framework with the ArchiMate 2.1 Modeling Language."
- Barone, Daniele, Eric Yu, Jihyun Won, Lei Jiang, and John Mylopoulos. 2010. "Enterprise Modeling for Business Intelligence." In *IFIP Working Conference on The Practice of Enterprise Modeling*, 31–45. Springer.
- Baxter, Rohan, Peter Christen, and Tim Churches. 2003. "A Comparison of Fast Blocking Methods for Record Linkage." In *ACM SIGKDD*, 3:25–27. Citeseer.
- Berka, Christopher, Stefan Humer, Manuela Lenk, Mathias Moser, Henrik Rechta, and Eliane Schwerer. 2016. "A Quality Framework for Statistics Based on Administrative Data Sources Using the Example of the Austrian Census 2011." *Austrian Journal of Statistics* 39 (4): 299–308.

Bibliography

- Berti-Équille, L. 2007. "Quality Awareness for Data Managing and Mining." *Habilitation À Diriger Les Recherches, Université de Rennes 1*.
- Beyer, Dirk, Andreas Noack, and Claus Lewerentz. 2005. "Efficient Relational Calculation for Software Analysis." *IEEE Transactions on Software Engineering* 31 (2).
- Biggs, Bill. 2005. "Ministry of Defence Architectural Framework (Modaf)."
- Bilenko, Mikhail, Raymond Mooney, William Cohen, Pradeep Ravikumar, and Stephen Fienberg. 2003. "Adaptive Name Matching in Information Integration." *IEEE Intelligent Systems*, no. 5: 16–23.
- Bilenko, Mikhail, and Raymond J Mooney. 2003. "Adaptive Duplicate Detection Using Learnable String Similarity Measures." In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 39–48. ACM.
- Blaha, Michael. 2010. *Patterns of Data Modeling*. Vol. 1. CRC Press.
- Boer, Frank S de, Marcello M Bonsangue, Joost Jacob, Andries Stam, and L Van der Torre. 2005. "Enterprise Architecture Analysis with Xml." In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, 222b–222b. IEEE.
- Bond, Francis, and Kyonghee Paik. 2012. "A Survey of Wordnets and Their Licenses." *Small* 8 (4): 5.
- Brinkkemper, Sjaak. 1996. "Method Engineering: Engineering of Information Systems Development Methods and Tools." *Information and Software Technology* 38 (4): 275–80.
- Brockmans, Saartje, Peter Haase, and Rudi Studer. 2006. "A MOF-Based Metamodel and UML Syntax for Networked Ontologies." In *Intl. Semantic Web Conf. Georgia, US*.
- Brückmann, Matthias, Klaus-Manfred Schöne, Stefan Junginger, and Diana Boudinova. 2009. "Evaluating Enterprise Architecture Management Initiatives-How to Measure and Control the Degree of Standardization of an IT Landscape." In *EMISA*, 155–68.
- Bucher, Tobias, and Robert Winter. 2008. "Dissemination and Importance of the Method Artifact in the Context of Design Research for Information Systems." *Proceedings of the Third International Conference on Design Science Research in Information Systems and Technology (DESRIST 2008)*.
- Buckl, Sabine, Markus Buschle, Pontus Johnson, Florian Matthes, and Christian M Schweda. 2011. "A Meta-Language for Enterprise Architecture Analysis." In *Enterprise, Business-Process and Information Systems Modeling*, 511–25. Springer.
- Buckl, Sabine, Alexander M Ernst, Josef Lankes, Kathrin Schneider, and Christian M Schweda. 2007. "A Pattern Based Approach for Constructing Enterprise Architecture Management Information Models." *Wirtschaftsinformatik Proceedings 2007*, 65.
- Buckl, Sabine, Alexander M Ernst, Florian Matthes, René Ramacher, and Christian M Schweda. 2009. "Using Enterprise Architecture Management Patterns to Complement TOGAF." In *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, 34–41. IEEE.
- Buckl, Sabine, Jens Gulden, and Christian M Schweda. 2010. "Supporting Ad Hoc Analyses on Enterprise Models." In *EMISA*, 69–83.
- Buckl, Sabine, Florian Matthes, and Christian M Schweda. 2009. "Classifying Enterprise Architecture Analysis Approaches." In *Enterprise Interoperability*, 66–79. Springer.

Bibliography

- Buschle, Markus, Mathias Ekstedt, Sebastian Grunow, Matheus Hauder, Florian Matthes, and Sascha Roth. 2012. "Automating Enterprise Architecture Documentation Using an Enterprise Service Bus."
- Buuren, René van, Henk Jonkers, Maria-Eugenia Iacob, and Patrick Strating. 2004. "Composition of Relations in Enterprise Architecture Models." In *ICGT*, 3256:39–53. Springer.
- Caetano, Artur. 2016. "An Application of Semantic Techniques to the Analysis of Enterprise Architecture Models." In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 4536–45. IEEE.
- Chakrabarti, Soumen, Martin Ester, Usama M. Fayyad, Johannes Gehrke, Jiawei Han, Shinichi Morishita, Gregory Piatetsky-Shapiro, and Wei Wang. 2006. "Data Mining Curriculum: A Proposal (Version 1.0)." *Intensive Working Group of ACM SIGKDD Curriculum Committee*, 140.
- Chapman, Pete, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. 2000. "CRISP-DM 1.0." *CRISP-DM Consortium*.
- Choi, Seung-Seok, Sung-Hyuk Cha, and Charles C Tappert. 2010. "A Survey of Binary Similarity and Distance Measures." *Journal of Systemics, Cybernetics and Informatics* 8 (1): 43–48.
- Chukmol, Uddam, Rami Rifaieh, and Nabila Aicha Benharkat. 2005. "Exsmal: Edi/Xml Semi-Automatic Schema Matching Algorithm." In *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*, 422–25. IEEE.
- Coenen, Frans, Graham Goulbourne, and Paul Leng. 2004. "Tree Structures for Mining Association Rules." *Data Mining and Knowledge Discovery* 8 (1): 25–51.
- Cohen, William W, Pradeep D Ravikumar, and Stephen E Fienberg. 2003. "A Comparison of String Distance Metrics for Name-Matching Tasks." In *Proceedings of IJCAI-03 Workshop on Information Integration*, 2003:73–78.
- Conradi, Reidar, and Bernhard Westfechtel. 1998. "Version Models for Software Configuration Management." *ACM Computing Surveys (CSUR)* 30 (2): 232–82.
- Council, CIO. 1999. "Federal Enterprise Architecture Framework Version 1.1." Retrieved from 80: 3–1.
- Cowie, Jim, and Wendy Lehnert. 1996. "Information Extraction." *Communications of the ACM* 39 (1): 80–91.
- Czarnecki, Krzysztof, and Simon Helsen. 2003. "Classification of Model Transformation Approaches." In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 45:1–17. USA.
- Dalgaard, Peter. 2008. *Introductory Statistics with R*. Springer Science & Business Media.
- Daneva, Maya. 2004. "ERP Requirements Engineering Practice: Lessons Learned." *Software, IEEE* 21 (2): 26–33.
- Dardenne, Anne. 1993. *On the Use of Scenarios in Requirements Acquisition*. Department of Computer and Information Science, University of Oregon.
- Davenport, Thomas, H. 2007. "Competing on Analytics." HBR Press.
- Davoudi, Mahsa Razavi, Fereidoon Shams Aliee, and Kambiz Badie. 2011. "An AHP-Based Approach toward Enterprise Architecture Analysis Based on Enterprise Architecture Quality Attributes." *Knowledge and Information Systems* 28 (2): 449–72.

Bibliography

- Davoudi, Mahsa Razavi, and K Sheikhvand. 2012. "An Approach towards Enterprise Architecture Analysis Using AHP and Fuzzy AHP." *International Journal of Machine Learning and Computing* 2 (1): 46.
- Deerwester, Scott, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. "Indexing by Latent Semantic Analysis." *Journal of the American Society for Information Science* 41 (6): 391.
- Denk, Michaela, Karl Anton Froeschl, and Wilfried Grossmann. 2002. "Statistical Composites: A Transformation-Bound Representation of Statistical Datasets." In *SSDBM '02 Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, 217–26. IEEE.
- DeWitt, David J, Jeffrey F Naughton, and Donovan A Schneider. 1991. "An Evaluation of Non-Equijoin Algorithms." In *Proceedings of the 17th International Conference on Very Large Data Bases*, 443–52. Morgan Kaufmann Publishers Inc.
- Deza, Michel Marie, and Elena Deza. 2009. "Encyclopedia of Distances." In *Encyclopedia of Distances*, 1–583. Springer.
- Dietz, JLG. 2006. "Enterprise Ontology: Theory and Methodology." *Berling: Springer*.
- DoD, CIO. 2010. "DoDAF Architecture Framework Version 2.02." *Website, August*.
- Dublin Core Metadata Initiative. 2012. "Dublin Core Metadata Element Set, Version 1.1."
- Dupont, William D, and Walton D Plummer. 1990. "Power and Sample Size Calculations: A Review and Computer Program." *Controlled Clinical Trials* 11 (2): 116–28.
- Dusetzina, Stacie B, Seth Tyree, Anne-Marie Meyer, Adrian Meyer, Laura Green, and William R Carpenter. 2014. "An Overview of Record Linkage Methods."
- Duval, Erik. 2001. "Metadata Standards: What, Who & Why." *Journal of Universal Computer Science* 7 (7): 591–601.
- Ehrig, Hartmut, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. 2005. "Formal Integration of Inheritance with Typed Attributed Graph Transformation for Efficient VL Definition and Model Manipulation." In , 71–78. IEEE.
- Ekstedt, Mathias, Ulrik Franke, Pontus Johnson, Robert Lagerström, Teodor Sommestad, Johan Ullberg, and Markus Buschle. 2009. "A Tool for Enterprise Architecture Analysis of Maintainability." In , 327–28.
- Eltinge, John L, Paul P Biemer, and Anders Holmberg. 2013. "A Potential Framework for Integration of Architecture and Methodology to Improve Statistical Production Systems." *Journal of Official Statistics* 29 (1): 125–45.
- Falleri, Jean-Rémy, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. 2008. "Metamodel Matching for Automatic Model Transformation Generation." In *Model Driven Engineering Languages and Systems*, 326–40. Springer.
- Farwick, Matthias, Ruth Breu, Matheus Hauder, Sascha Roth, and Florian Matthes. 2013. "Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation." In *Proceedings of the 2013 46th Hawaii International Conference on System Sciences*, 3868–77. IEEE.
- Farwick, Matthias, Christian M Schweda, Ruth Breu, and Inge Hanschke. 2016. "A Situational Method for Semi-Automated Enterprise Architecture Documentation." *Software & Systems Modeling* 15 (2): 397–426.

Bibliography

- Fatolahi, Ali, and Fereidoon Shams. 2006. "An Investigation into Applying UML to the Zachman Framework." *Information Systems Frontiers* 8 (2): 133–43.
- Faust, Katherine. 1997. "Centrality in Affiliation Networks." *Social Networks* 19 (2): 157–91.
- Fayyad, Usama M., Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. "From Data Mining to Knowledge Discovery in Databases." *AI Magazine* 17 (3): 37.
- Fayyad, Usama M., Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. 1996. "Advances in Knowledge Discovery and Data Mining."
- FEAPO. 2013. "Common Perspectives on Enterprise Architecture." *Architecture and Governance Magazine* Issue 9-4 (November).
- Fill, Hans-Georg, and Florian Johannsen. 2016. "A Knowledge Perspective on Big Data by Joining Enterprise Modeling and Data Analyses." In *System Sciences (HICSS), 2016 49th Hawaii International*, 4052–61. IEEE.
- Fischer, Ronny, Stephan Aier, and Robert Winter. 2007. "A Federated Approach to Enterprise Architecture Model Maintenance." *Enterprise Modelling and Information Systems Architectures* 2 (2): 14–22.
- Fischer, Ronny, and Robert Winter. 2007. "Ein Hierarchischer, Architekturbasierter Ansatz Zur Unterstützung Des IT/Business Alignment." *Wirtschaftsinformatik Proceedings 2007*, 66.
- Fisher, Amit, Mike Nolan, Sanford Friedenthal, Michael Loeffler, Mark Sampson, Manas Bajaj, Lonnie VanZandt, Krista Hovey, John Palmer, and Laura Hart. 2014. "3.1. 1 Model Lifecycle Management for MBSE." In *INCOSE International Symposium*, 24:207–29. Wiley Online Library.
- Florez, Hector, Mario Sánchez, and Jorge Villalobos. 2014. "Extensible Model-Based Approach for Supporting Automatic Enterprise Analysis." In *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*, 32–41. IEEE.
- . 2016. "Analysis of Imprecise Enterprise Models." In *International Workshop on Business Process Modeling, Development and Support*, 349–64. Springer.
- Fox, Christopher. 1989. "A Stop List for General Text." In *ACM SIGIR Forum*, 24:19–21. ACM.
- Frank, Ulrich. 2002. "Multi-Perspective Enterprise Modeling (Memo) Conceptual Framework and Modeling Languages." In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, 1258–67. IEEE.
- Franke, Ulrik, David Höök, Johan König, Robert Lagerström, Per Närman, Johan Ullberg, Pia Gustafsson, and Mathias Ekstedt. 2009. "EAF2-a Framework for Categorizing Enterprise Architecture Frameworks." In *SNPD '09 Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, 327–32. IEEE.
- Friedman, Nir, Lise Getoor, Daphne Koller, and Avi Pfeffer. 1999. "Learning Probabilistic Relational Models." In *IJCAI*, 99:1300–1309.
- Fröschl, K. A., and Wilfrid Grossmann. 2001. "Deciding Statistical Data Quality." *New Techniques and Technologies for Statistics/Exchange of Technology and Know-How, Pre-Proc.*, no. 1.
- Gale, Thornton, and James Eldred. 1996. *Getting Results with the Object-Oriented Enterprise Model*. Sigs Books New York, NY, USA.

Bibliography

- Gentner, Dedre. 1983. "Structure-Mapping: A Theoretical Framework for Analogy*." *Cognitive Science* 7 (2): 155–70.
- Giachetti, Ronald E. 2010. *Design of Enterprise Systems: Theory, Architecture, and Methods*. CRC Press.
- Gill, Asif Qumer, and Muhammad Atif Qureshi. 2015. "Adaptive Enterprise Architecture Modelling." *Journal of Software* 10 (5): 628–38.
- Glazner, Christopher G. 2011. "Enterprise Transformation Using a Simulation of Enterprise Architecture." *Journal of Enterprise Transformation* 1 (3): 231–60.
- Gomes, Cláudio, Bruno Barroca, and Vasco Amaral. 2014. "Classification of Model Transformation Tools: Pattern Matching Techniques." In *International Conference on Model Driven Engineering Languages and Systems*, 619–35. Springer.
- Götzinger, David, Elena-Teodora Miron, and Franz Staffel. 2016. "OMiLAB: An Open Collaborative Environment for Modeling Method Engineering." In *Domain-Specific Conceptual Modeling*, 55–76. Springer.
- Gough, Paul A, Filip T Fodemski, Stewart A Higgins, and SJ Ray. 1995. "Scenarios-an Industrial Case Study and Hypermedia Enhancements." In *Proceedings of the Second IEEE International Symposium on*, 10–17. IEEE.
- Grossmann, Wilfried. 2009. "A Conceptual Approach for Data Integration in Business Analytics." *International Journal of Software and Informatics* 4: 53–68.
- . 2015. "Metadata." *Wiley StatsRef: Statistics Reference Online*. doi:10.1002/9781118445112.
- Grossmann, Wilfried, and Christoph Moser. 2016. "Big Data—Integration and Cleansing Environment for Business Analytics with DICE." In *Domain-Specific Conceptual Modeling*. Springer.
- Grunow, Sebastian, Florian Matthes, and Sascha Roth. 2013. "Towards Automated Enterprise Architecture Documentation: Data Quality Aspects of SAP PI." In *Advances in Databases and Information Systems*, 103–13. Springer.
- Gschwandtner, Theresia, Johannes Gärtner, Wolfgang Aigner, and Silvia Miksch. 2012. "A Taxonomy of Dirty Time-Oriented Data." In *International Conference on Availability, Reliability, and Security*, 58–72. Springer.
- Gudas, Saulius, Audrius Lopata, and Tomas Skersys. 2005. "Approach to Enterprise Modelling for Information Systems Engineering." *Informatica* 16 (2): 175–92.
- Han, Jiawei, and Micheline Kamber. 2000. "Data Mining: Concepts and Techniques (the Morgan Kaufmann Series in Data Management Systems)." Morgan Kaufmann.
- Handley, Holly AH, and Robert J Smillie. 2008. "Architecture Framework Human View: The NATO Approach." *Systems Engineering* 11 (2): 156–64.
- Harmsen, Anton Frank. 1997. "Situational Method Engineering." University of Twente, Twente, The Netherlands.
- Harmsen, Anton Frank, Jacobus Nicolaas Brinkkemper, and JL Han Oei. 1994. *Situational Method Engineering for Information System Project Approaches*. Citeseer.
- Hause, Matthew, Daniel Brookshier, and Graham Bleakley. 2012. "UPDM-Unified Profile for DoDAF/MODAF." DTIC Document.

Bibliography

- Henderson-Sellers, Brian, Cesar Gonzalez-Perez, and Jolita Ralyté. 2008. "Comparison of Method Chunks and Method Fragments for Situational Method Engineering." In , 479–88. IEEE.
- Henderson-Sellers, Brian, and Jolita Ralyté. 2010. "Situational Method Engineering: State-of-the-Art Review." *J. UCS* 16 (3): 424–78.
- Hernández, Mauricio A, and Salvatore J Stolfo. 1998. "Real-World Data Is Dirty: Data Cleansing and the Merge/Purge Problem." *Data Mining and Knowledge Discovery* 2 (1): 9–37.
- Hevner, AR, Salvatore T March, Jinsoo Park, and Sudha Ram. 2004. "Design Science in Information Systems Research." *MIS Quarterly* 28 (1): 75–105.
- Hinkelmann, Knut, Michaela Maise, and Barbara Thönssen. 2013. "Connecting Enterprise Architecture and Information Objects Using an Enterprise Ontology." In , 1–11. IEEE.
- Iacob, Maria-Eugenia, and Henk Jonkers. 2006. "Quantitative Analysis of Enterprise Architectures." In *Interoperability of Enterprise Software and Applications*, 239–52. Springer.
- ISO/IEC/IEEE 25012. 2008. "Software Engineering — Software Product Quality Requirements and Evaluation (SQuaRE) — Data Quality Model."
- ISO/IEC/IEEE 42010. 2011. "Systems and Software Engineering—Architecture Description." ISO/IEC/IEEE 42010.
- Israel, Glenn D. 1992. *Determining Sample Size*. University of Florida Cooperative Extension Service, Institute of Food and Agriculture Sciences, EDIS Gainesville.
- Jeners, Simona, Horst Lichter, and Elena Pyatkova. 2012. "Metric Based Comparison of Reference Models Based on Similarity." *International Journal of Digital Content Technology and Its Applications* 6 (21): 50.
- Johannsen, Florian, and Hans-Georg Fill. 2014. "RUPERT: A Modelling Tool for Supporting Business Process Improvement Initiatives." In *International Conference on Design Science Research in Information Systems*, 418–22. Springer.
- Johnson, Pontus, Lars Nordström, and Robert Lagerström. 2007. "Formalizing Analysis of Enterprise Architecture." In *Enterprise Interoperability*, 35–44. Springer.
- Jonkers, Henk, Harmen van den Berg, Maria-Eugenia Iacob, and Dick Quartel. 2010. "ArchiMate® Extension for Modeling the TOGAF™ Implementation and Migration Phases." *Reading, Berkshire: Whitepaper, The Open Group*.
- Kao, Anne, and Steve R Poteet. 2007. *Natural Language Processing and Text Mining*. Springer Science & Business Media.
- Karagiannis, Dimitris. 2015. "Agile Modeling Method Engineering." In , 5–10. ACM.
- Karagiannis, Dimitris, Robert Andrei Buchmann, Patrik Burzynski, Ulrich Reimer, and Michael Walch. 2016. "Fundamental Conceptual Modeling Languages in OMiLAB." In *Domain-Specific Conceptual Modeling*, 3–30. Springer.
- Karagiannis, Dimitris, and Harald Kühn. 2002. "Metamodelling Platforms." In *EC-Web*, 2455:182.
- Karagiannis, Dimitris, Christoph Moser, and Arash Mostashari. 2012. "Compliance Evaluation Featuring Heat Maps (CE-HM): A Meta-Modeling-Based Approach." In *Advanced Information Systems Engineering*, 414–28. Springer.

Bibliography

- Kazman, Rick, Gregory Abowd, Len Bass, and Paul Clements. 1996. "Scenario-Based Analysis of Software Architecture." *Software, IEEE* 13 (6): 47–55.
- Kent, Jean-Pierre, and Maarten Schuerhoff. 1997. "Some Thoughts about a Metadata Management System." In *SSDBM '97 Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, 97:174–85. Citeseer.
- Kern, Heiko, Axel Hummel, and Stefan Kühne. 2011. "Towards a Comparative Analysis of Meta-Metamodels." In *Proceedings of the Compilation of the Co-Located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPEs'11, NEAT'11, & VMIL'11*, 7–12. ACM.
- Kessentini, Marouane, Ali Ouni, Philip Langer, Manuel Wimmer, and Slim Bechikh. 2014. "Search-Based Metamodel Matching with Structural and Syntactic Measures." *Journal of Systems and Software* 97: 1–14.
- Khan, Mumit. 2011. "Longest Path in a Directed Acyclic Graph (DAG)."
- Kim, Won, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. 2003. "A Taxonomy of Dirty Data." *Data Mining and Knowledge Discovery* 7 (1): 81–99.
- Klösgen, Willi. 2002. "Types and Forms of Data." *Handbook of Data Mining and Knowledge Discovery*, Oxford University Press, New York, USA, 33–44.
- Klosterboer, Larry. 2007. *Implementing ITIL Configuration Management*. Pearson Education.
- Korhonen, Janne J, Mehmet Yildiz, and Juha Mykkanen. 2009. "Governance of Information Security Elements in Service-Oriented Enterprise Architecture." In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, 768–73. IEEE.
- Kühn, Harald, Franz Bayer, Stefan Junginger, and Dimitris Karagiannis. 2003. "Enterprise Model Integration." In *International Conference on Electronic Commerce and Web Technologies*, 379–92. Springer.
- Kumar, Kuldeep, and Richard J Welke. 1992. "Methodology Engineering: A Proposal for Situation-Specific Methodology Construction." In *Challenges and Strategies for Research in Systems Development*, 257–69. John Wiley & Sons, Inc.
- Kurpjuweit, Stephan, and Stephan Aier. 2009. "Ein Allgemeiner Ansatz Zur Ableitung von Abhängigkeitsanalysen Auf Unternehmensarchitekturmodellen." In *Wirtschaftsinformatik*, 129–38.
- Lagerström, Robert, Pontus Johnson, and David Höök. 2010. "Architecture Analysis of Enterprise Systems Modifiability—models, Analysis, and Validation." *Journal of Systems and Software* 83 (8): 1387–1403.
- Lapkin, Anne, Phillip Allega, Brian Burke, Betsy Burton, R Scott Bittler, Robert A Handler, Greta A James, Bruce Robertson, David Newman, and Deborah Weiss. 2008. "Gartner Clarifies the Definition of the Term Enterprise Architecture." *Research G00156559, Gartner*.
- Laverdure, Leo, and Alex Conn. 2012. "SEA Change: How Sustainable EA Enables Business Success in Times of Disruptive Change." *Journal of Enterprise Architecture* 8 (1).
- Lee, Wei-Nchi, Nigam Shah, Karanjot Sundlass, and Mark A Musen. 2008. "Comparison of Ontology-Based Semantic-Similarity Measures." In *Annual Symposium Proceedings. AMIA Symposium*.

Bibliography

- Leist, Susanne, and Gregor Zellner. 2006. "Evaluation of Current Architecture Frameworks." In *Proceedings of the 2006 ACM Symposium on Applied Computing*, 1546–53. ACM.
- Lenzerini, Maurizio. 2002. "Data Integration: A Theoretical Perspective." In , 233–46. ACM.
- Li, Lin, Taoxin Peng, and Jessie Kennedy. 2014. "A Rule Based Taxonomy of Dirty Data." *GSTF Journal on Computing (JoC)* 1 (2).
- Liberatore, Matthew J, and Wenhong Luo. 2010. "The Analytics Movement: Implications for Operations Research." *Interfaces* 40 (4): 313–24.
- Lúcio, Levi, Moussa Amrani, Jürgen Dingel, Leen Lambers, Rick Salay, Gehan MK Selim, Eugene Syriani, and Manuel Wimmer. 2014. "Model Transformation Intents and Their Properties." *Software & Systems Modeling*, 1–38.
- Manzur, Laura, Jorge Mario Ulloa, Mario Sánchez, and Jorge Villalobos. 2015. "xArchiMate: Enterprise Architecture Simulation, Experimentation and Analysis." *Simulation* 91 (3): 276–301.
- Marbán, Óscar, Gonzalo Mariscal, and Javier Segovia. 2009. "A Data Mining & Knowledge Discovery Process Model." *Data Mining and Knowledge Discovery in Real Life Applications. IN-TECH* 2009: 8.
- Marcus, Mitchell P, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. "Building a Large Annotated Corpus of English: The Penn Treebank." *Computational Linguistics* 19 (2): 313–30.
- Matthes, Dirk. 2011. *Enterprise Architecture Frameworks Kompendium*. Springer-Verlag.
- McHugh, Jason, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. 1997. "Lore: A Database Management System for Semistructured Data." *SIGMOD Record* 26 (3): 54–66.
- Mendling, Jan, Hajo A Reijers, and Jan Recker. 2010. "Activity Labeling in Process Modeling: Empirical Insights and Recommendations." *Information Systems* 35 (4): 467–82.
- Merriam-Webster, I. 2007. "Merriam-Webster's Dictionary and Thesaurus."
- Miller, George A. 1995. "WordNet: A Lexical Database for English." *Communications of the ACM* 38 (11): 39–41.
- Mirbel, Isabelle, and Jolita Ralyté. 2006. "Situational Method Engineering: Combining Assembly-Based and Roadmap-Driven Approaches." *Requirements Engineering* 11 (1): 58–78.
- Mishra, Priti, and Margaret H Eich. 1992. "Join Processing in Relational Databases." *ACM Computing Surveys (CSUR)* 24 (1): 63–113.
- Monahov, Ivan, Christopher Schulz, Alexander Schneider, and Florian Matthes. 2012. "EAM KPI Catalog."
- Moser, Christoph, Robert Buchmann Andrei, Wilfrid Utz, and Dimitris Karagiannis. 2017. "CE-SIB: A Modelling Method Plug-in for Managing Standards in Enterprise Architectures."
- Moser, Christoph, Daniel Fürstenau, and Stefan Junginger. 2010. "A Method for Integrating EAM and BPM." In *Software Engineering (Workshops), 2nd European Workshop on Patterns for Enterprise Architecture Management (PEAM2010)*, 231–42. Paderborn, Germany.

Bibliography

- Moser, Christoph, Stefan Junginger, Matthias Brückmann, and Klaus-Manfred Schöne. 2009. "Some Process Patterns for Enterprise Architecture Management." In *Workshopband, Fachtagung Des GI-Fachbereichs Softwaretechnik*, 19–30. Kaiserlautern, Germany: Citeseer.
- Moser, Christoph, Mathias Winklhofer, and Christian Kuplich. 2008. "Business Objectives Compliance Architecture Framework." In *Proceedings of Modellierung 2008*. Berlin, Germany: Springer.
- Mourya, Maya, and Preeti Saxena. 2015. "Survey of Xml to Relational Database Mapping Techniques." *Adv. Comput. Sci. Inf. Technol.(ACSIT)* 2: 162–66.
- Murata, Tadao. 1989. "Petri Nets: Properties, Analysis and Applications." *Proceedings of the IEEE* 77 (4): 541–80.
- Mykhashchuk, Mariana, Sabine Buckl, Thomas Dierl, and Christian M Schweda. 2011. "Charting the Landscape of Enterprise Architecture Management." In *Wirtschaftsinformatik*, 83.
- Närman, Per, Markus Buschle, Johan König, and Pontus Johnson. 2010. "Hybrid Probabilistic Relational Models for System Quality Analysis." In *Enterprise Distributed Object Computing Conference (EDOC), 2010 14th IEEE International*, 57–66. IEEE.
- Närman, Per, Ulrik Franke, Johan König, Markus Buschle, and Mathias Ekstedt. 2014. "Enterprise Architecture Availability Analysis Using Fault Trees and Stakeholder Interviews." *Enterprise Information Systems* 8 (1): 1–25.
- Närman, Per, Hannes Holm, Pontus Johnson, Johan König, Moustafa Chenine, and Mathias Ekstedt. 2011. "Data Accuracy Assessment Using Enterprise Architecture." *Enterprise Information Systems* 5 (1): 37–58.
- Närman, Per, M Schonherr, Pontus Johnson, Mathias Ekstedt, and Moustafa Chenine. 2008. "Using Enterprise Architecture Models for System Quality Analysis." In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, 14–23. IEEE.
- Neaga, Elena I, and Jennifer A Harding. 2005. "An Enterprise Modeling and Integration Framework Based on Knowledge Discovery and Data Mining." *International Journal of Production Research* 43 (6): 1089–1108.
- Niemann, Klaus D. 2006. *From Enterprise Architecture to IT Governance*. Vol. 1. Springer.
- Oliveira, Paulo, Fátima Rodrigues, Pedro Henriques, and Helena Galhardas. 2005. "A Taxonomy of Data Quality Problems." In *2nd Int. Workshop on Data and Information Quality*, 219–33. Citeseer.
- OMG. 2015. "Meta Object Facility (MOF) Core, Version 2.5." Object Management Group. <http://www.omg.org/spec/MOF/2.5>.
- Ordóñez, Carlos. 2010. "Statistical Model Computation with UDFs." *IEEE Transactions on Knowledge and Data Engineering* 22 (12): 1752–65.
- Ordóñez, Carlos, and Zhibo Chen. 2012. "Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis." *IEEE Transactions on Knowledge and Data Engineering* 24 (4): 678–91.
- Österlind, Magnus, Pontus Johnson, Kiran Karnati, Robert Lagerström, and Margus Välja. 2013. "Enterprise Architecture Evaluation Using Utility Theory." In *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*, 347–51. IEEE.

Bibliography

- Pallottino, Stefano. 1984. "Shortest-path Methods: Complexity, Interrelations and New Propositions." *Networks* 14 (2): 257–67.
- Papageorgiou, Haralambos, Fragkiskos Pentaris, Eirini Theodorou, Maria Vardaki, and Michalis Petrakos. 2001. "A Statistical Metadata Model for Simultaneous Manipulation of Both Data and Metadata." *Journal of Intelligent Information Systems* 17 (2–3): 169–92.
- Papageorgiou, Haralambos, Maria Vardaki, and Fragkiskos Pentaris. 2000. "Data and Metadata Transformations." *Research in Official Statistics* 3 (2): 27–43.
- Parent, Christine, Stefano Spaccapietra, and Esteban Zimányi. 1999. "Spatio-Temporal Conceptual Models: Data Structures+ Space+ Time." In , 26–33. ACM.
- Peppers, Ken, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. 2007. "A Design Science Research Methodology for Information Systems Research." *Journal of Management Information Systems* 24 (3): 45–77.
- Pereira, Carla Marques, and Pedro Sousa. 2005. "Enterprise Architecture: Business and IT Alignment." In *Proceedings of the 2005 ACM Symposium on Applied Computing*, 1344–45. ACM.
- Peter Christen. 2007. "Towards Parameter-Free Blocking for Scalable Record Linkage."
- Porter, Edward H, and William E Winkler. 1997. "Approximate String Comparison and Its Effect on an Advanced Record Linkage System." In *Advanced Record Linkage System. US Bureau of the Census, Research Report*. Citeseer.
- Porter, Martin F. 1980. "An Algorithm for Suffix Stripping." *Program* 14 (3): 130–37.
- Potts, Chris. 2010. "Using Structural Performance Ratios to Guide Investments in Enterprise Architecture." *Journal of Enterprise Architecture* 6 (4): 14–18.
- Pouya Aleatrati Khosroshahi, Alexander W. Schneider, Matheus Hauder, and Florian Matthes. 2015. "Enterprise Architecture Management Pattern Catalog," no. Version 2.0.
- Prat, Nicolas, Isabelle Comyn-Wattiau, and Jacky Akoka. 2014. "Artifact Evaluation in Information Systems Design-Science Research-a Holistic View." In *PACIS 2014 Proceedings - Pacific Asia Conference on Information Systems*, 23. Citeseer.
- Rahm, Erhard, and Hong Hai Do. 2000. "Data Cleaning: Problems and Current Approaches." *IEEE Data Eng. Bull.* 23 (4): 3–13.
- Ralyté, Jolita, and Colette Rolland. 2001. "An Assembly Process Model for Method Engineering." In *International Conference on Advanced Information Systems Engineering*, 267–83. Springer.
- Ralyté, Jolita, Colette Rolland, and Rébecca Deneckère. 2004. "Towards a Meta-Tool for Change-Centric Method Engineering: A Typology of Generic Operators." In *International Conference on Advanced Information Systems Engineering*, 202–18. Springer.
- Ribeiro, Leonardo, and Theo Härder. 2006. "Entity Identification in XML Documents." In *Grundlagen von Datenbanken*, 130–34.
- Rolland, Colette, and Naveen Prakash. 1996. "A Proposal for Context-Specific Method Engineering." In *Method Engineering*, 191–208. Springer.

Bibliography

- Roszczyk, Marcin. 2015. "Enterprise Architecture - A System Engineering Discipline." August 3. <https://www.linkedin.com/pulse/enterprise-architecture-system-engineering-marcin-roszczyk>.
- Roth, Sascha. 2014. "Federated Enterprise Architecture Model Management." Dissertation. Munich University.
- Roth, Sascha, and Florian Matthes. 2014. "Visualizing Differences of Enterprise Architecture Models." In *Proceedings of International Workshop on Comparison and Versioning of Software Models (CVSM) at Software Engineering (SE)*.
- Rumbaugh, James, Ivar Jacobson, and Grady Booch. 2004. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education.
- Rusinowska, Agnieszka, Rudolf Berghammer, Harrie De Swart, and Michel Grabisch. 2011. "Social Networks: Prestige, Centrality, and Influence." In , 22–39. Springer.
- Saha, Pallab. 2004. "Analyzing The Open Group Architecture Framework from the GERAM Perspective." http://www.opengroup.org/architecture/wp/saha/TOGAF_GERAM_Mapping.htm.
- . 2007. "A Synergistic Assessment of the Federal Enterprise Architecture Framework against GERAM (ISO15704: 2000)."
- Santana, Alixandre, Kai Fischbach, and Hermano Moura. 2016. "Enterprise Architecture Analysis and Network Thinking: A Literature Review." In *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, 4566–75. IEEE.
- Schekkerman, Jaap. 2004a. "Extended Enterprise Architecture Framework (E2AF) Essentials Guide." *Institute For Enterprise Architecture Developments*.
- . 2004b. *How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating or Choosing an Enterprise Architecture Framework*. Trafford Publishing.
- Schoonjans, Anthony. 2016. "Social Network Analysis Techniques in Enterprise Architecture Management."
- Sciore, Edward. 1994. "Versioning and Configuration Management in an Object-Oriented Data Model." *The VLDB Journal—The International Journal on Very Large Data Bases* 3 (1): 77–106.
- Sharma, Rajeev, Peter Reynolds, Rens Scheepers, Peter B Seddon, and Graeme G Shanks. 2010. "Business Analytics and Competitive Advantage: A Review and a Research Agenda." In , 187–98.
- Shu, Nan C. 1987. "Automatic Data Transformation and Restructuring." In , 173–80. IEEE.
- Simon, Daniel, Kai Fischbach, and Detlef Schoder. 2013. "An Exploration of Enterprise Architecture Research." *Communications of the Association for Information Systems* 32 (1): 1–72.
- Singh, Prince M, and Marten J van Sinderen. 2015. "Lightweight Metrics for Enterprise Architecture Analysis." In *International Conference on Business Information Systems*, 113–25. Springer.
- Smith, John R, and Peter Schirling. 2006. "Metadata Standards Roundup." *IEEE MultiMedia* 13 (2): 84–88.
- Sommestad, Teodor, Mathias Ekstedt, and Hannes Holm. 2013. "The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures." *IEEE Systems Journal* 7 (3): 363–73.

Bibliography

- Sommestad, Teodor, Mathias Ekstedt, and Pontus Johnson. 2010. "A Probabilistic Relational Model for Security Risk Analysis." *Computers & Security* 29 (6): 659–79.
- Sprinkle, Jonathan, Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai. 2010. "Metamodelling - State of the Art and Research Challenges." Springer.
- Staab, Steffen, and Rudi Studer. 2013. *Handbook on Ontologies*. Springer Science & Business Media.
- Stephan, Eva-Maria, Andrej Vckovski, and Felix Bucher. 1993. "Virtual Data Set-An Approach for the Integration of Incompatible Data." In , 93–93. ASPRS.
- Stravinskienė, Auksė, and Saulius Gudas. 2011. "Enterprise Knowledge Modeling and Data Mining Integration."
- Sundgren, Bo. 1996. "Making Statistical Data More Available." *International Statistical Review/Revue Internationale de Statistique*, 23–38.
- Sunkle, Sagar, Vinay Kulkarni, and Suman Roychoudhury. 2013. "Analyzing Enterprise Models Using Enterprise Architecture-Based Ontology." In *International Conference on Model Driven Engineering Languages and Systems*, 622–38. Springer.
- Svejvig, Per, and Peter Andersen. 2015. "Rethinking Project Management: A Structured Literature Review with a Critical Look at the Brave New World." *International Journal of Project Management* 33 (2): 278–90.
- Syriani, Eugene, Jeff Gray, and Hans Vangheluwe. 2013. "Modeling a Model Transformation Language." In *Domain Engineering*, 211–37. Springer.
- Szegheo, Orsolya. 2000. "Introduction to Enterprise Modeling." In *Enterprise Modeling*, 21–32. Springer.
- Ter Hofstede, Arthur HM, and TF Verhoef. 1997. "On the Feasibility of Situational Method Engineering." *Information Systems* 22 (6): 401–22.
- The Open Group. 2011. "TOGAF Version 9.1."
- . 2015. "Archimate Model Exchange File Format."
- . 2016. "ArchiMate 3.0 Specification." Van Haren Publishing.
- Theodoulidis, Charalampos I, and Pericles Loucopoulos. 1991. "The Time Dimension in Conceptual Modelling." *Information Systems* 16 (3): 273–300.
- TOGAF BIAN Collaboration Work Group. 2012. "TOGAF BIAN White Paper."
- Torokhti, Anatoli, and Phil Howlett. 2000. *Theory of Hierarchical, Multilevel, Systems*. Vol. 68. Elsevier.
- Urbaczewski, Lise, and Stevan Mrdalj. 2006. "A Comparison of Enterprise Architecture Frameworks." *Issues in Information Systems* 7 (2): 18–23.
- Van der Aalst, Wil MP. 1998. "The Application of Petri Nets to Workflow Management." *Journal of Circuits, Systems, and Computers* 8 (01): 21–66.
- Van Der Aalst, Wil MP, Arthur HM Ter Hofstede, and Mathias Weske. 2003. "Business Process Management: A Survey." In *International Conference on Business Process Management*, 1–12. Springer.
- Van Rijsbergen, Cornelis J, Stephen Edward Robertson, and Martin F Porter. 1980. *New Models in Probabilistic Information Retrieval*. British Library Research and Development Department.

Bibliography

- Vardaki, Maria, Haralambos Papageorgiou, and Fragkiskos Pentaris. 2009. "A Statistical Metadata Model for Clinical Trials' Data Management." *Computer Methods and Programs in Biomedicine* 95 (2): 129–45.
- Vasconcelos, André, Pedro Sousa, and José Tribolet. 2007. "Information System Architecture Metrics: An Enterprise Engineering Evaluation Approach." *The Electronic Journal Information Systems Evaluation* 10 (1): 91–122.
- . 2015. "Enterprise Architecture Analysis-An Information System Evaluation Approach." *Enterprise Modelling and Information Systems Architectures* 3 (2): 31–53.
- Veneberg, RKM, Maria-Eugenia Iacob, MJ van Sinderen, and Lianne Bodestaff. 2014. "Enterprise Architecture Intelligence: Combining Enterprise Architecture and Operational Data." In *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*, 22–31. IEEE.
- Vernadat, François. 1996. *Enterprise Modeling and Integration*. Boom Koninklijke Uitgevers.
- . 2014. "Enterprise Modeling in the Context of Enterprise Engineering: State of the Art and Outlook." *International Journal of Production Management and Engineering* 2 (2): 57–73.
- Visic, Niksa, Hans-Georg Fill, Robert Andrei Buchmann, Dimitris Karagiannis, Thang Le Dinh, Tim Rickenberg, Michael H Breitner, Florian Johannsen, Hans-Georg Fill, and Claudiu Vasile Kifor. 2015. "A Domain-Specific Language for Modeling Method Definition: From Requirements to Grammar." In *IEEE Ninth International Conference on Research Challenges in Information Science*.
- Vom Brocke, Jan, Alexander Simons, Bjoern Niehaves, Kai Riemer, Ralf Plattfaut, and Anne Clevén. 2009. "Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process." In , 9:2206–17.
- Wegmann, Alain. 2002. "The Systemic Enterprise Architecture Methodology (SEAM). Business and IT Alignment for Competitiveness."
- Weisstein, Eric W. 2008. "Floyd-Warshall Algorithm."
- White, Stephen A. 2004. "Introduction to BPMN." *IBM Cooperation* 2 (0): 0.
- Winkler, William E. 1999. "The State of Record Linkage and Current Research Problems." In *Statistical Research Division, US Census Bureau*.
- Winter, Robert, and Ronny Fischer. 2006. "Essential Layers, Artifacts, and Dependencies of Enterprise Architecture." In *Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International*, 30–30. IEEE.
- Woburn C.I. Programming Enrichment Group (PEG). 2016. "Shortest Path." http://wcipeg.com/wiki/Shortest_path#Single-source_shortest_paths.
- Wolfram Alpha. 2017a. "Numerical Precision." August 2. <http://reference.wolfram.com/language/tutorial/NumericalPrecision.html>.
- . 2017b. "Vertex Contraction." August 2. <http://reference.wolfram.com/language/tutorial/NumericalPrecision.html>.
- Ye, Jun. 2011. "Cosine Similarity Measures for Intuitionistic Fuzzy Sets and Their Applications." *Mathematical and Computer Modelling* 53 (1): 91–97.
- Zachman, John A. 1987. "A Framework for Information Systems Architecture." *IBM Systems Journal* 26 (3): 276–92.

Bibliography

- Zimmermann, Alfred, Kurt Sandkuhl, Michael Pretz, Michael Falkenthal, Dierk Jügel, and Matthias Wissotzki. 2013. "Towards an Integrated Service-Oriented Reference Enterprise Architecture." In *Proceedings of the 2013 International Workshop on Ecosystem Architectures*, 26–30. ACM.
- Zrnec, Aljaz, Marko Bajec, and Marjan Krisper. 2001. "Enterprise Modelling with UML." *Electro Tech. Rev* 68 (2–3): 109–14.

List of Tables

12 List of Tables

Table 1 Concepts for architecture descriptions	32
Table 2 Overview and assessment of EA analysis approaches	61
Table 3 Archetypical transformation task types – behaviour concepts in DICE	93
Table 4 Archetypical types of data analysis objects in DICE	94
Table 5 Example data object after initialisation.....	105
Table 6 Characteristics of semi-structured data based on (Abiteboul 1997)	111
Table 7 Common quality indicators organised along the DICE metamodel.....	119
Table 8 Exemplary sources for EA data.....	151
Table 9 Exemplary excerpt of CMDB report.....	160
Table 10 Requirements for supporting EAA.....	172
Table 11 Categorized list of structural neighbours	187
Table 12 Neighbours of the building block "Claim registration"	187
Table 13 Excerpt of overall similarity of Archimate and BPMN concepts (Gill and Qureshi 2015).....	207
Table 14 Quantity structures of the input datasets	220
Table 15 Operational Data – Simplified Dataset	221
Table 16 Initial structure of input dataset.....	223
Table 17 Publications by the author	246

13 List of Figures

Fig. 1 Continuously evolving as-is and target architectures	14
Fig. 2 Preparing EA data for EA decision making.....	16
Fig. 3 From heterogeneous datasets to a sound basis for EA analysis.....	22
Fig. 4 Design Science Research Methodology Process Model, adapted from (Peffer et al. 2007).....	24
Fig. 5 Thesis Outline	26
Fig. 6 Context of architecture descriptions (ISO/IEC/IEEE 42010 2011).....	30
Fig. 7 Conceptual model for architecture descriptions (ISO/IEC/IEEE 42010 2011)	32
Fig. 8 Model kinds defined by TOGAF (The Open Group 2011)	35
Fig. 9 CRISP-DM process (Chapman et al. 2000).....	37
Fig. 10 The main elements of a modelling method.....	40
Fig. 11 Simplified Archimate Metamodel (Iacob and Jonkers 2006).....	41
Fig. 12 Structure of Archimate 3.0 (The Open Group 2016).....	43
Fig. 13 The process for configuring a situational method, marginally adjusted from (Harmsen 1997).....	46
Fig. 14 Method fragments in DICE.....	47
Fig. 15 Interaction of BA and EA	51
Fig. 16 Main steps of the literature research (Andersen and Carugati 2014).....	54
Fig. 17 Model transformation terminology, adapted from (Syriani, Gray and Vangheluwe 2013).....	65
Fig. 18 Composite Data Analysis Object	69
Fig. 19 Example of a composite analysis object – data level.....	70
Fig. 20 Data-level concepts and related metadata concepts.....	71
Fig. 21 Example a data object and annotated metadata	72
Fig. 22 Horizontal vs. vertical integration of two datasets.....	74
Fig. 23 Simultaneous transformations for data and metadata objects.....	75
Fig. 24 Core components of situational methods.....	78

List of Figures

Fig. 25 Hierarchical breakdown of method chunks through specialisation/refinement.....	79
Fig. 26 Constitutional elements of transformation task types	80
Fig. 27 Multi-level hierarchy of DICE	80
Fig. 28 Modelling procedure – Phase “Business understanding”, adapted from (Chapman et al. 2000)	83
Fig. 29 Modelling procedure – Phase “Data understanding”, adapted from (Chapman et al. 2000)	84
Fig. 30 Modelling procedure – Phase “Data preparation”, adapted from (Chapman et al. 2000)	87
Fig. 31 DICE Metamodel Overview	90
Fig. 32 Complete MM of the DICE structural part	97
Fig. 33 Petri net representation of an exemplary DICE workflow	101
Fig. 34 Nested loop join on data level	102
Fig. 35 Merge algorithm extended with fragments for calculation of meta objects	103
Fig. 36 Determine quality indicators for properties	129
Fig. 37 Levels and structure of DICE quality indicators.....	130
Fig. 38 Heatmapped Clustermap – Technology support of business processes	135
Fig. 39 Gantt, portfolio and box plot views on catalogues	137
Fig. 40 The data structure of EA matrices	139
Fig. 41 Heatmapped matrix (Moser et al. 2017)	139
Fig. 42 Structure of node/edge directed graph template in UML, adapted from (Blaha 2010)	140
Fig. 43 Node edge template, adapted from (Blaha 2010)	140
Fig. 44 Example system/technology matrix	141
Fig. 45 Nested box (The Open Group 2016) and node/edge diagrams (The Open Group 2011)	142
Fig. 46 Application Usage Viewpoint (The Open Group 2016)	145
Fig. 47 Node edge representation of application usage models	146

List of Figures

Fig. 48 Exemplary meta model snippet supporting PRM analysis (Buckl et al. 2011)	146
Fig. 49 Exemplary metamodel for an indicator-based approach (Vasconcelos, Sousa and Tribolet 2015).....	148
Fig. 50 Node edge undirected graph template, adapted from (Blaha 2010)	149
Fig. 51 Reflexive relations – example in Archimate notation.....	153
Fig. 52 EA strata, adapted from (Fischer and Winter 2007)	155
Fig. 53 Application usage model with contracted application services	156
Fig. 54 ABBs and SBBs	157
Fig. 55 Different representations of logical and physical interfaces in Archimate	158
Fig. 56 Types and Instances - Usage of BA data in EAA	161
Fig. 57 As-is, transition and target architectures.....	162
Fig. 58 Application usage model incl. plateaus depicting transition architectures	164
Fig. 59 Varying depths of versioning for applications and technologies	166
Fig. 60 Alternative Architectures	169
Fig. 61 Metamodel mapping: Archimate vs. TOGAF content metamodel (Band et al. 2015)	170
Fig. 62 Structure of datasets on data-level to represent EA data	175
Fig. 63 Structure of datasets on data-level to represent EA data including time aspects.....	176
Fig. 64 Exemplary snippet of an Archimate model in the Archimate Model Exchange Format	177
Fig. 65 The DICE “Restructure Archimate Model” transformation task type.....	178
Fig. 66 Extending the initialisation transformation task type	180
Fig. 67 Multi-layer viewpoint	183
Fig. 68 Vectors on structural neighbours	187
Fig. 69 Binary vectors on structural neighbours	188
Fig. 70 Merging of EA models.....	190
Fig. 71 Generic record linkage process	190
Fig. 72 Data after applying blocking transformation	192

List of Figures

Fig. 73 DICE similarity node table	193
Fig. 74 Symmetric distance matrix	194
Fig. 75 Generic Relationship Types in DICE mapped onto Archimate relations	196
Fig. 76 Contradiction of equivalent building blocks.....	198
Fig. 77 Contraction of a building block in the case of reflexive relations (with aggregation relation).....	198
Fig. 78 One level entirely removed.....	199
Fig. 79 Matching reflexively structured graph components	200
Fig. 80 Longest path algorithm to determine height of building blocks in graph-based models	201
Fig. 81 Example of derived relationships	203
Fig. 82 Adoption of the Floyd-Warshall algorithm to detect shortest paths within EA models	205
Fig. 83 Method conceptualisation lifecycle, adapted from (Visic et al. 2015)	211
Fig. 84 The DICE architecture	213
Fig. 85 Metamodel of the DICE workflow (excerpt).....	214
Fig. 86 GraphRep definition of the transformation task type “Selection”	215
Fig. 87 AttrRep - ADOxx notebook definition	216
Fig. 88 DICE processing metadata and quality metadata (per variable).....	217
Fig. 89 Example of an ADOxx expression assembling R code	218
Fig. 90 Exemplary DICE workflow mapped in the DICE modeller	222
Fig. 91 Example of DICE metadata specification in ADOxx	226
Fig. 92 Exemplary subset of the building blocks in the dataset	227
Fig. 93 DICE process incl. quality line charts.....	230
Fig. 94 Development of the quality indicator "uniqueness" throughout the process	231
Fig. 95 DICE - Structural part of the metamodel - subset "Dataset"	237
Fig. 96 DICE - Structural part of the metamodel - subset "Observable Unit"	238
Fig. 97 DICE - Structural part of the metamodel - subset "Variable"	239

List of Abbreviations

Fig. 98 DICE - Structural part of the metamodel - subset "Property" 240

14 List of Abbreviations

ABB	Architecture Building Block
ADM	Architecture Development Method
AHP	Analytical Hierarchy Process
BA	Business Analytics
BI	Business Intelligence
BB	Building Block
BPM	Business Process Management
BPMN	Business Process Management Notation
CRISP-DM	CRoss-Industry Standard Process for Data Mining
D	Dataset
DAG	Directed acyclic graph
DICE	Data Integration and Clearance Environment
DoDAF	Department of Defence Architecture Framework
GERAM	Generalised Enterprise Reference Architecture and Methodology
EA	Enterprise Architecture
EAA	Enterprise Architecture Analytics
EAM	Enterprise Architecture Management
FEAF	Federal Enterprise Architecture Framework
ITIL	IT Infrastructure Library
LOG	Logistics Metadata
MODAF	Ministry of Defense Architecture Framework
MM	Metamodel
P	Property
PRM	Probabilistic Relation Model

List of Abbreviations

PROC	Processing Metadata
SBB	Solution Building Block
SEM	Semantic Metadata
SEMMA	Sample, Explore, Modify, Model, and Assess
SME	Situational Method Engineering
TOGAF	The Open Group Architecture Framework
U	Observable Unit
UML	Unified Modelling Language
V	Variable