TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Software Engineering betrieblicher Informationssysteme

Empowering End-users to Collaboratively Analyze Evolving Complex Linked Data

Thomas Georg Reschenhofer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Martin Bichler
Prüfer der Dissertation:		
	1.	Univ.-Prof. Dr. Florian Matthes
	2.	Univ.-Prof. Dr. Ulrich Frank, Universität Duisburg-Essen

Die Dissertation wurde am 03.11.2016 bei der Technischen Universität München

eingereicht und durch die Fakultät für Informatik am 21.02.2017 angenommen.

Zusammenfassung

Heutige Unternehmen haben ihre Informationssysteme laufend an sich ändernde Rahmenbedingungen anzupassen. Diese Veränderungen werden getrieben von neuen Geschäftsanforderungen, von technologischen Innovationen, oder von neuen gesetzlichen Regulierungen. Die Befähigung der Anwender, sich selbst ihre Informationssysteme anzupassen, kann dabei zu einer erhöhten Effizienz führen. Das umfasst insbesondere Anpassungen an den domänenspezifischen Datenstrukturen der verwendeten Informationssysteme. Aus sich laufend ändernden und immer komplexer werdenden Datenstrukturen ergeben sich allerdings neue Herausforderungen bei der Analyse der Daten und der Generierung von Wissen daraus.

In dieser Arbeit adressieren wir diese Herausforderungen und entwickelten einen Ansatz für die anwender-basierte Analyse komplexer Datenstrukturen in einer sich ändernden und kollaborativen Umgebung. Dabei werden Geschäftsanwender dazu befähigt, analytische Abstraktionen zu definieren und diese visuell in domänenspezifischen Dashboards zu analysieren. Das entwickelte Informationssystem sorgt zum einen dafür, dass benutzerdefinierten Analyseartefakte stets konsistent bleiben, insbesondere bei Änderungen des Datenmodells. Andererseits erlaubt es seinen Anwendern, die Analyseartefakte und deren Abhängigkeiten zueinander zu untersuchen und diesbezüglich Transparenz zu schaffen.

Im ersten Schritt analysieren wir den aktuellen Stand der Technik bei der Fachanwender-orientierten Datenanalyse (EUA) und bestimmen Erfolgsfaktoren sowie Schwächen aktueller Werkzeuge. Basierend auf diesen Ergebnissen sowie auf Basis relevanter Literatur leiten wir Anforderungen für eine kollaborative EUA Lösung ab. Diese dienen dabei als Grundlage für den konzeptionellen Entwurf eines innovativen Werkzeugs, welches Fachanwendern ermöglicht, kollaborativ komplexe Datenstrukturen zu analysieren. Durch eine prototypische Implementierung können wir dessen Benutzerschnittstelle demonstrieren und des Weiteren eine Evaluation in Kooperation mit unterschiedlichen Industriepartnern durchführen.

Um unterschiedliche Aspekte der entwickelten Konzepte und des Prototyps zu validieren, haben wir unterschiedliche Evaluationsstrategien angewandt. In einem Online-Experiment mit Studenten, Forschern und Praktikern bewerteten wir die Bedienbarkeit des Prototyps, während wir durch Interviews mit Industriepartnern Feedback über dessen Nutzen für verschiedene Anwendungsfälle sammelten. Des Weiteren waren wir im Rahmen einer umfassenden Fallstudie in der Lage, die Anwendbarkeit des Prototyps in der Praxis für einen bestimmten EUA Anwendungsfall zu untersuchen.

Der Beitrag dieser Arbeit ist vielfältig: Erstens können die abgeleiteten Anforderungen als Wegweiser für die Entwicklung ähnlicher Softwarewerkzeuge dienen. Zweitens können die diskutierten Designentscheidungen als Indikatoren dienen, wie diese Anforderungen umgesetzt werden können. Und drittens repräsentieren die Ergebnisse der Evaluation des Prototyps zum einen interessante Erkenntnisse aus dessen Anwendung in der Praxis, und zum anderen Verbesserungsvorschläge und damit zukünftige potentielle Forschungsrichtungen.

Abstract

Today's enterprises have to align and adapt their information systems continuously to an evolving environment which is driven by changing business requirements, technological innovations, and new legal regulations. Empowering end-users to adapt information systems by themselves can improve their autonomy and thus the efficiency of their daily business operation. This particularly includes the end-user-driven adaption of an information system's data model capturing domain-specific data structures. However, the evolution of increasingly complex data structures imposes significant challenges to knowledge generation through analyzing those data structures.

In the thesis at hand, we addressed this issue and developed an approach empowering end-users to analyze complex data structures in an evolving and collaborative environment. Thereby, business users are enabled to define analytical abstractions based on an evolving data model, and to visually analyze them in domain-specific and customizable dashboards. On the one hand, the presented information system ensures the consistency of end-user-created analysis artifacts in the light of data model evolution. On the other hand, it also provides means to enable the exploration of the analysis artifacts and their dependencies through end-users in order to create awareness in a collaborative environment.

We analyze the state-of-the-art in End-User Analytics (EUA) and determine success factors of respective tools as well as shortcomings they are currently suffering from. Based on those findings as well as related literature, we derive generic requirements for a collaborative EUA solution. Those requirements serve as foundation for the conceptual design of an innovative tool empowering end-user to collaboratively analyze evolving complex data structures. By prototypically implementing this solution, we demonstrate its user interface design and enable its evaluation in cooperation with different industry partners.

We applied different evaluations strategies in order to validate different aspects of the developed concepts and the corresponding prototype. By conducting an online experiment with students, researchers, and practitioners, we assessed the prototype's usability, while through interviews we gathered feedback on its utility. Furthermore, we conducted a comprehensive case study to evaluate the prototype's practicability. Thereby, three companies applied it for a concrete EUA use-case and reported on their experiences.

The contribution of this thesis is manifold: First, the derived requirements can serve as drivers for the development of related software tools empowering end-users to analyze evolving complex linked data in a collaborative way. Second, the discussions of design decision regarding the solution's concepts and user interface provide indications of how those requirements can be addressed. And third, the findings from the prototype's evaluation represent helpful lessons learned of applying such a solution on the one hand, and suggestions for improvement and thus future research objectives on the other hand.

Acknowledgment

This thesis emerged from my work as a research associate at the Chair for Software Engineering for Business Information Systems (sebis) at the Technical University of Munich (TUM). Over that period of time I enjoyed a fruitful cooperation and pleasant time with many people including my advisors, colleagues, students, family, and friends.

First and foremost, I would like to thank my doctoral father and supervisor Prof. Dr. Florian Matthes for providing me the opportunity to work on this extremely interesting research topic under the best possible conditions. Since the very first day you made me feel confident in working towards my thesis with a well-balanced blend of freedom and supervision. I further want to express my sincere gratitude to Prof. Dr. Ulrich Frank for being the second supervisor of my thesis and for our discussions about the subject.

The sebis chair provided an excellent environment for my research. The thesis would not have been possible without the support of my colleagues. I would like to thank Bernhard Waltl for his contribution to my research through multiple co-authorships and extensive discussions, which partially continued even at home. Furthermore, my thanks go to Manoj Mahabaleshwar, Dr. Ivan Monahov, Dr. Sven-Volker Rehm, Dr. Alexander Schneider, and Klym Shumaiev for contributing to my research through discussing ideas and co-authoring publications, and to Adrian Hernandez-Mendez and Felix Michel for their cooperation in developing the SocioCortex platform. Thanks to Pouya Aleatrati Khosroshahi, Dr. Matheus Hauder, Dominik Huth, Martin Kleehaus, Ömer Uludag, and Marin Zec for the constructive cooperation in teaching and researching.

I would also like to thank the students who contributed to my research with their theses, guided research, or as student assistant. Thanks to Patrick Bürgin, Daniel Elsner, Sirma Gjorgievska, Florian Katenbrink, Matti Maier, Alexander Meissner, Oleksii Moroz, Tobias Schrade, Peter Velten, and Valérianne Walter. I really enjoyed working with you.

Finally, and most importantly, I express my sincere gratitude to my partner Nicole Kugler for her unconditional love, support, and patience during this challenging time. Thank you for reminding me regularly to take a step back from my work in order to refuel new energy. I am also utmost grateful to my parents Anna and Georg Reschenhofer. Without your great support and encouragement during my whole life this work would not have been possible.

Garching b. München, 03.11.2016

Thomas Reschenhofer

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Motivation and Introduction

New business requirements, changing legal regulations, and technological innovations lead to continuously changing demands and requirements for information systems in today's enterprises (Teece et al., 1997; Pavlou and El Sawy, 2011; Ahlemann et al., 2012; Aleatrati Khosroshahi et al., 2015). Grubb and Takang (2003) even argue that most changes of information systems happen after their implementation. Addressing this issue, End-User Development (EUD) (Lieberman et al., 2006; Nardi, 1993) empowers end-users to adapt business applications by themselves in order to tailor it to the changing environment. This kind of end-user empowerment is a double-edged sword: On the one hand, it imposes significant challenges to knowledge workers, e.g., the necessity for additional training, or the additional burden through being responsible for those adaptions (Pinnington et al., 2007). On the other hand, EUD can improve the autonomy and potentially the efficiency of knowledge workers in their daily business, since they are not reliant on Information Technology (IT) experts to adapt business applications accordingly (Mørch et al., 2004; Vitharana, 2003).

For data-driven business applications, EUD is about adapting a domain-specific data model to changes affecting the corresponding business domain (McGinnes and Kapros, 2015; Curino et al., 2009). An example for a domain in which the data model can be subject to changes at run-time is Enterprise Architecture Management (EAM). EAM as a management discipline to holistically plan, develop, and control the fundamental organization (International Organization for Standardization, 2007)—also referred to as EA—includes the maintenance of an EA model which captures components and relationships of an EA and serves as decision support for different EA stakeholders (Ahlemann et al., 2012). Changing business requirements or other external factors can imply the need for adaptions of the organization-specific EA model. As a consequence, EA tools implementing such EA models have to provide EUD means in order to allow enterprise architects to tailor the implemented EA model to the changing environment (Matthes et al., 2008). This analogously applies to innovation processes where a conceptual data model cannot be defined beforehand since—naturally—it is not yet known at design-time (Davenport, 2013; Rehm et al., 2014).

McGinnes and Kapros (2015) describe the adaptability of an information system's data model and the decoupling of its implementation from domain-specific structures as *conceptual independence*. It can be achieved by meta-model-based information systems which define a generic meta-model instead of a domain-specific data model. In this sense, the meta-model represents a model of the data model, i.e., it specifies the means by which end-users of a meta-model-based information systems can define data models at run-time.

With this regard, (Matthes et al., 2011) developed a meta-model-based information system in the shape of the *Hybrid Wiki* approach. Thereby, wiki pages represent data entities which initially consist of unstructured textual content and which can be related to each other through embedded hyperlinks. Those wiki pages can be incrementally and collaboratively structured by assigning an entity type and by adding attributes and relations. At the same time, modeling experts can define a data model consisting of entity types and attribute definitions, which stipulate certain constraints to corresponding data entities and attributes. Therefore, data models emerge collaboratively through mutual interaction between end-users structuring the data entities and modeling experts defining the domain-specific structures and constraints. Since the Hybrid Wiki meta-model not only allows the definition of trivial attributes (e.g., number, string, boolean, date), but—after a minor extension, cf. Section 4.1.1—also the definition of complex nested objects (Kim et al., 1989), attributes with multiple values, and relations to other entities, we refer to the corresponding data structures as *complex linked data*.

DEFINITION: **Complex Linked Data**

In the context of this thesis, complex linked data refers to entities which potentially have arbitrarily nested attributes, attributes with multiple values, and multiple relations to other data entities.

Based on a domain-specific data model, users typically want to define domain-specific queries and tailored views (Burnett and Myers, 2014) in order to perform stakeholder-specific data analytics. In fact, the application of the Hybrid Wiki approach in different research projects and domains, e.g., EAM (Matthes and Neubert, 2011; Buckl et al., 2009, 2010) and New Product Development (NPD) (Rehm et al., 2014), also revealed the need for means which enable end-users to analyze the underlying data, i.e., to create queries and views based on the collaboratively defined data model (Schneider et al., 2015; Roth et al., 2013; Rehm et al., 2014)—often in an ad-hoc manner. Tamm et al. (2013) name this kind of self-directed analysis *End-User Analytics (EUA)*. However, while the adaptability of a meta-model-based information system's data model as well as their support for complex linked data structures is appealing and enables tool-support for cases with uncertain or evolving data structures, those system properties also lead to non-trivial EUA-related challenges. We will elaborate on those challenges in the following section.

1.1. Problem Description

The general research objective of the present thesis is to address the challenges which arise from the endeavor of applying EUA to collaborative Adaptive Information System (AIS) which implement evolving and complex linked data structures and support knowledge-intensive team work (Davenport, 2013). This objective is primarily motivated by the application of Hybrid Wikis in multiple research projects within the last five years.

For example, the application of Hybrid Wikis in EAM revealed that different EA stakeholders want to define and evaluate KPIs (Matthes et al., 2012a) and metrics (Schneider et al., 2015) to facilitate evidence-based decision support, e.g., to support standardization processes by quantifying the diversity of an organization's application landscape. Consequently, end-users have to be empowered to define such KPIs and metrics which potentially consist of a series of diverse operations, e.g., query, logical, or arithmetic operations (Monahov, 2014). For example, a metric for calculating the heterogeneity of used database systems in a given functional domain can be defined as the "diversity of databases used by business applications in a given functional domain" (Schneider et al., 2015). This metric is based on the entropy measure (EM) which is formally defined as (Schuetz et al., 2013):

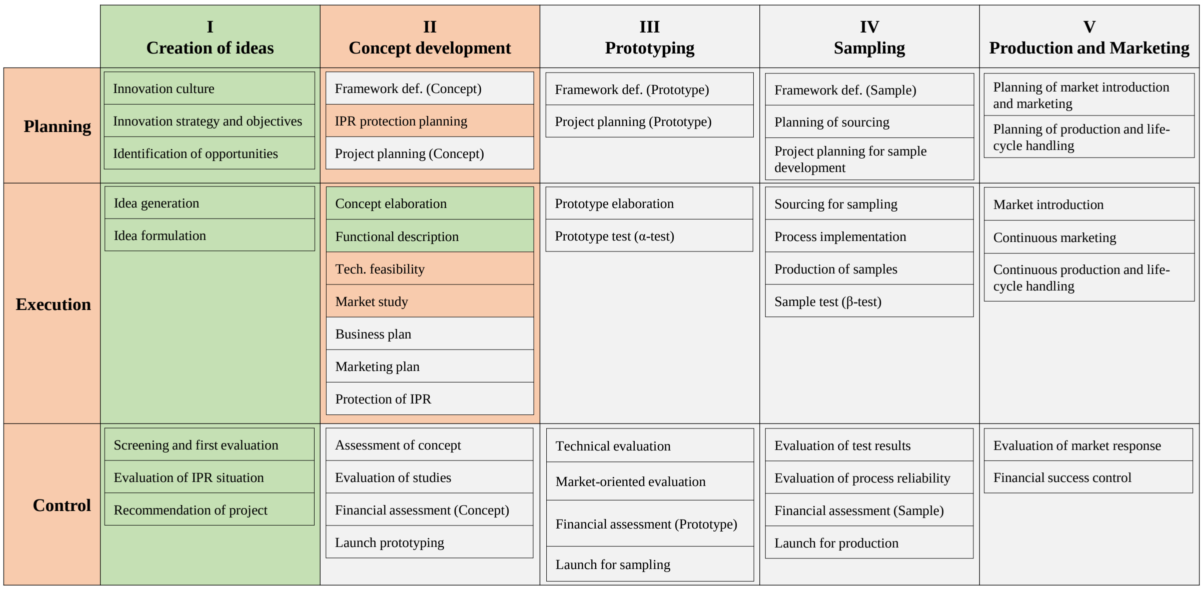
$$EM := -\sum_{i=1}^{n} p_i ln(p_i) \; with \; p_i := \frac{x_i}{\sum_{j=1}^{n} x_j}$$

$$x_i := absolute\ number\ of\ elements\ assigned\ to\ characteristic\ i$$

Although, we do not need to understand this formula in detail, it demonstrates that EA metrics in particular and queries in general can be complex, e.g., they might contain nested aggregations (sum) and sophisticated mathematical operations (like the natural logarithm). Hauder et al. (2013) confirm that the definition of metrics is still an open issue for current tools in EAM.

In another context, we applied the Hybrid Wiki tool as a collaborative information system to systematically share and adaptively integrate knowledge between partners of innovation networks (Rehm et al., 2014). For coordinating a development project whose aim is the design of a new product, involved stakeholders had to be aware of the current state within this project, whereas the project state can be derived from specific business rules. Addressing this issue, the so-called *SmartNet Navigator* (Matheis, 2013) as shown in Figure 1.1 visualizes the phases of a development project as well as the current status, i.e., which phases are already completed (green), in progress (orange), or still open (gray). The SmartNet Navigator is data-driven, since the status of the individual phases are automatically derived by end-user-defined business rules which are based on an underlying domain-specific data model. The uncertain nature of innovation processes implies that the data model might evolve over time, which in turn means that the business rules determining the project status have to be adaptable as well as adaptive. The former property refers to the changeability at run-time, while the latter one refers to the automated adaption of a business rule on changes of the underlying data model.

The most obvious approach to End-User Analytics (EUA) would be to simply use spreadsheets as the most common class EUA tools (Tamm et al., 2013). In this way, the complex linked data structures have to be imported to a spreadsheet tool, which then provides proven means for quantitative EUA. The spreadsheet empowers end-users to define calculations and



	I Creation of ideas	II Concept development	III Prototyping	IV Sampling	V Production and Marketing
Planning	Innovation culture Innovation strategy and objectives Identification of opportunities	Framework def. (Concept) IPR protection planning Project planning (Concept)	Framework def. (Prototype) Project planning (Prototype)	Framework def. (Sample) Planning of sourcing Project planning for sample development	Planning of market introduction and marketing Planning of production and life-cycle handling
Execution	Idea generation Idea formulation	Concept elaboration Functional description Tech. feasibility Market study Business plan Marketing plan Protection of IPR	Prototype elaboration Prototype test (α-test)	Sourcing for sampling Process implementation Production of samples Sample test (β-test)	Market introduction Continuous marketing Continuous production and life-cycle handling
Control	Screening and first evaluation Evaluation of IPR situation Recommendation of project	Assessment of concept Evaluation of studies Financial assessment (Concept) Launch prototyping	Technical evaluation Market-oriented evaluation Financial assessment (Prototype) Launch for sampling	Evaluation of test results Evaluation of process reliability Financial assessment (Sample) Launch for production	Evaluation of market response Financial success control

Figure 1.1.: The SmartNet Navigator Matheis (2013) as an example for a data-driven view

quantitative visualizations and to implement KPIs and metrics. However, this leads to several issues (Reschenhofer and Matthes, 2015a): Firstly, while the rigid spreadsheet grid is an easy-to-understand metaphor for end-users to manage and analyze data, it hinders the representation of complex linked data as defined in the previous section. Secondly, repeating certain EUA tasks on a regular basis requires to redo the data import multiple times. If the data model evolved and changed since the last import, the spreadsheets importing the newly structured data have to be manually adapted too. Thirdly, spreadsheets as Desktop tools for personal decision making (Senn, 2004) lack collaborative features (Nardi and Miller, 1990; Ginige et al., 2010a) which would enable end-users to perform EUA collaboratively. In Section 2.2 we elaborate on the role of spreadsheets in the context of this thesis in more detail.

On a related note, the end-user-friendly analysis of model-based data structures was subject to research by Roth et al. (2013) which developed an approach to automatically configure visualizations based on structural model matching. Thereby, a visualization's information demand is defined by an abstract view model capturing generic entities, attributes, and relations. A mapping algorithm can identify parts of a domain-specific data model which match to the abstract view model of a visualization and provide suggestions for binding data entities to visualizations. While this enables a user-friendly way of configuring model-based visualizations, end-users are not empowered to perform quantitative analysis, e.g., define complex metrics or calculations based on the underlying data.

To sum up, the application of Hybrid Wikis as a representative of meta-model-based information systems revealed that there is the need for performing quantitative EUA on the system's evolving complex linked data structures, whereas both aforementioned alternatives—namely spreadsheets and structured model matching—are not suitable for use cases like the definition and tool-supported evaluation of KPIs and metrics, or the creation of data-driven views.

There are multiple properties of collaborative meta-model-based information systems imposing significant challenges to the endeavor of applying EUA to this kind of software: Firstly, the evolution of the data model has multiple consequences. On the one hand, end-users have to be empowered to define queries and calculations at run-time and in an ad-hoc manner. Since the data model is continuously evolving, end-users have to be familiarized with the underlying data model continuously too. Furthermore, already existing and stored queries of the data model have to be adaptive, i.e., the have to be kept consistent regarding references to changing data model elements. Secondly, querying complex linked data models implies that the query language has to provide a respective type system and a sufficient expressiveness which includes an extendable set of operators required for the definition of complex metrics and calculations. Thirdly, support for team work and collaborativeness implies the need for access rights in order to control which users are allowed to interact with which system artifacts in which way. Collaboration support also includes means to share system artifacts with co-workers. Moreover, to coordinate collaborating end-users in performing EUA, a respective system should create awareness regarding which co-workers should be contacted for specific purposes.

Current tools do not address all those challenges. On the one hand, there are traditional EUA tools (e.g., spreadsheets) and Self-Service BI applications enabling end-users to analyze and visualize data in a self-directed way. However, those tools typically lack collaboration features or support for evolving complex linked data. On the other hand, collaborative meta-model-based information systems support evolving complex linked data structures, but typically lack capabilities empowering end-users to analyze and visualize it (cf. Chapter 2 for a more detailed discussion about related research and tools). As a consequence, the general research objective of this thesis is to close this gap.

1.2. Research Questions

We formalize the main contributions of this thesis by deriving research questions from the problem statement. Those particularly address the challenges as described in the previous section. The first research question is about a state-of-the-art analysis of how current EUA tools are used in industry, and from which shortcomings spreadsheets are suffering when applying them for knowledge-intensive team work:

> ***RQ 1:*** What is the state-of-the-art of tool support in EUA, and what are shortcomings of those tools—particularly spreadsheets—in the context of knowledge-intensive team work?

The state-of-the-art analysis is done by multiple case studies in different companies whose purpose is the identification of common usage patterns of spreadsheets (Reschenhofer and Matthes, 2015b) and recurring shortcomings of today's spreadsheet applications (Reschenhofer and Matthes, 2015a) (cf. Sections 2.2.2, 2.2.3, and 2.2.4). Furthermore, by analyzing more than 20,000 spreadsheets from two publicly available corpora, we were able to investigate semantic patterns of spreadsheets (Reschenhofer et al., 2016c) as well as the complexity of their structure (Reschenhofer et al., 2016d). Utilizing the knowledge of current patterns and shortcomings of spreadsheets, our intent is to improve the current situation and to identify system principles which have to be implemented by novel solutions to collaborative EUA based on evolving complex linked data. Therefore, we define the second research question as follows:

> ***RQ 2:*** What are requirements for empowering end-users to collaboratively analyze evolving complex linked data structures?

To answer this question, we first derived a conceptual framework for collaborative EUA based on a meta-model-based information system from related work on end-user-driven data visualizations (Chi and Riedl, 1998), visual information analysis (Isenberg et al., 2008), end-user-driven model analysis (Roth et al., 2013), and general EUA (Tamm et al., 2013) (cf. Section 3.1). Based on this framework as well as on already existing requirements and design principles for collaborative information systems (de Hertogh et al., 2011) we can define a set of requirements which guide the system design of a respective tool. This leads to the next research question:

> ***RQ 3:*** How can a system design for a tool empowering end-users to define data models, analytical abstractions, and visualizations look like?

The system design includes the definition of a conceptual meta-model for the collaborative management and analysis of complex linked data. In the present thesis, we use the Hybrid Wiki meta-model as defined by Matthes et al. (2011) as a foundation and extend it by concepts required for performing EUA on Hybrid Wiki data models (Reschenhofer and Matthes, 2016a) (cf. Chapter 4) Although we extend the Hybrid Wiki meta-model, the proposed concepts for collaborative EUA are applicable to meta-model-based information systems in general, as long as they share certain properties with the Hybrid Wiki tool. A specific set of properties is about the query language which is used by end-users to define queries and calculations. This language is subject to the next research question:

> ***RQ 4:*** What are features and properties a language for defining queries and calculations on evolving complex linked data structures must have?

Requirements like the support for complex linked data and adaptiveness also affect the expression language by which end-users define queries and thus refer to the user-defined and potentially evolving data model. To meet these requirements, such an expression language has to have certain properties which will be discussed in Section 4.2. UI-related aspects, e.g., how to define queries and visualizations, are subject to the next research question:

> ***RQ 5:*** How can end-users be supported in defining complex queries, calculations, and visualizations on evolving complex linked data structures?

When defining queries, end-users have to be aware not only of the query language, but also have to be familiar with the underlying data model (Valencia-García et al., 2011). This is particularly challenging in the light of frequent data model changes which implies the need for continuous familiarization of the end-user with the data model. Furthermore, the complex linked nature of the data model imposes specific requirements to a UI for formulating queries, e.g., when defining metrics or data-driven visualizations. Respective UI concepts will be described in Section 5.2.3. Since we aim for collaborative EUA, multiple users are interacting with the systems implying the need to create awareness of who is doing what, and in particular, which co-workers interact with system artifacts which are relevant for a specific end-user. Therefore, the next research question is addressing collaboration in general, and the awareness aspect in particular:

> ***RQ 6:*** How can end-users be supported in performing a meta-analysis on collaboratively defined data models, analytical abstractions, and visualizations?

By making this meta-information explorable for end-users, the system can support them in performing usage analysis, stakeholder identification, or impact analysis (Reschenhofer et al., 2016b). Section 4.4 elaborates on concepts and UI-techniques empowering end-users to explore existing data, analysis, and visualization artifacts and their relations to the system's users. As part of the design science research methodology as defined by Hevner et al. (2004), we evaluated the proposed concepts in multiple iterations and with different evaluation methods, e.g., case studies and experiments (cf. Chapter 6). The last research question addresses the findings of this evaluation, particularly those identified in the prototype's practical application in an industrial environment:

> ***RQ 7:*** What is the experience of users of the proposed solution? What are further challenges of Web 2.0-based EUA?

In the following Section, we elaborate on how we intend to address those research questions from a research methodology perspective.

1.3. Research Design

This thesis applies the design science research methodology as defined by Hevner et al. (2004). Figure 1.2 shows how we adopted the framework to this thesis' context. The purpose of design science is the creation and evaluation of a new IT artifact based on business needs on the one hand, and an existing knowledge base on the other hand. In this sense, it provides a framework for our objective of systematically developing a solution for collaborative EUA on evolving complex linked data.

The environment as one driving force of the framework represents the problem space and context and refers to people, organizations, and technologies which drive the design of an IT artifact. Related to this, Sections 1.1 and 1.2 already introduced the problem statement and corresponding research questions respectively. Chapter 3 will further elaborate on the environment by describing a conceptual framework of collaborative EUA based on complex linked data which also includes roles of involved users. The derived requirements represent the business needs as defined by the design science research method and thus ensure relevance of the designed IT artifact.

The other driving force of design science is the knowledge base which ensures rigor of the IT artifact's design. It captures related work which is summarized in Chapter 2 of this thesis. On the one hand, we describe related work which serves as foundation for the design of this thesis' artifact. This includes research about EUA in general and spreadsheets in particular. Furthermore, we elaborate on meta-model-based information systems and describe Hybrid Wikis

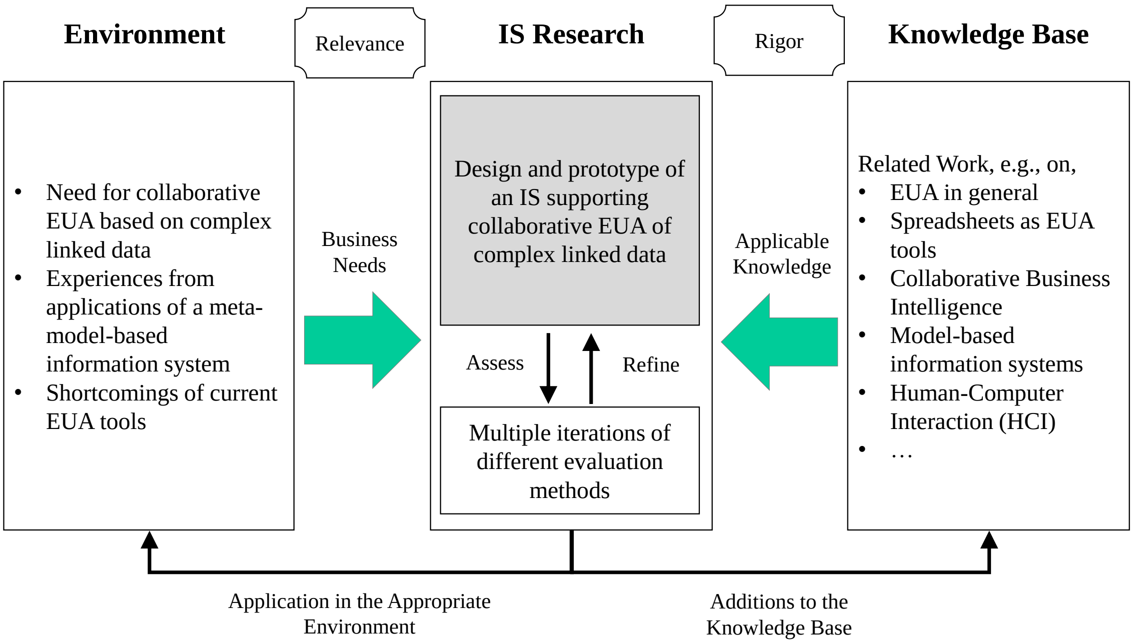

Environment
Relevance
IS Research
Rigor
Knowledge Base
• Need for collaborative EUA based on complex linked data
• Experiences from applications of a meta-model-based information system
• Shortcomings of current EUA tools
Business Needs
Design and prototype of an IS supporting collaborative EUA of complex linked data
Assess
Refine
Multiple iterations of different evaluation methods
Applicable Knowledge
Related Work, e.g., on,
• EUA in general
• Spreadsheets as EUA tools
• Collaborative Business Intelligence
• Model-based information systems
• Human-Computer Interaction (HCI)
• …
Application in the Appropriate Environment
Additions to the Knowledge Base

Figure 1.2.: Information systems research framework (Hevner et al., 2004) adapted to the present thesis' contribution

as one of their representatives in detail. On the other hand, we summarize related approaches to collaborative EUA and differentiate them from the approach as pursued in this thesis.

The heart of the design science research framework—both figuratively and conceptually—is the design of an IT artifact. This thesis' core IT artifact is the design and prototypical implementation of an information system which supports collaborative EUA based on complex linked data (cf. Chapters 4 and 5). According to Hevner et al. (2004), the IT artifact should be designed in multiple iterations including two different phases: A "Develop/Build" phase in which concrete design and development activities are carried out, and a "Justify/Evaluate" phase in which intermediary manifestations are assessed against the business needs in general, and concrete requirements in particular. Chapter 6 of this thesis elaborates on how our IT artifact was evaluated in different phases of the design process and with different evaluation methods, e.g., experiments and case studies.

As recommended by Hevner et al. (2004), we follow the seven guidelines which they propose in order to perform effective design-science research:

1. Design as an Artifact In this thesis, we develop an approach for collaborative EUA based on complex linked data which represents the main IT artifact in the sense of design science. As described in Chapter 5, we also implemented a respective prototype and outline technical implementation highlights, particularly its UI design. Therefore, the prototype is the *instantiation* of the designed IT artifact and furthermore serves as a proof of concept.

2. Problem Relevance In Section 1.1, we outlined two domains—EAM and NPD—in which EUA

is hindered by challenges induced by, e.g., complex linked data, data model evolution, and collaboration. As shown in Chapter 2, current EUA approaches and tools do not address those challenges adequately, or—more specifically—do not meet the requirements as described in Chapter 3. On the other hand, related research (cf. Section 2.3) shows that there is a need for a collaborative EUA approach for evolving complex-linked data.

3. Design Evaluation As described in Chapter 6, we validated our approach in multiple iterations. In the first iteration, we evaluated primarily the expressiveness and utility of the designed query language by applying the prototype for the definition and automated calculation of metrics in EAM (cf. Section 4.2.3). In a second iteration, we demonstrated the prototype to three different organizations in different domains, and interviewed them in order to identify concrete and practical use cases and concerns which are addressable by our approach (cf. Section 6.2). As a third evaluation iteration, we conducted experiments for assessing the usability of the editor for defining complex queries. Finally, in the fourth and final iteration, we gave the prototype to three enterprises which used it for ad-hoc EA analysis based on their organization-specific EA data (cf. Section 6.3).

4. Research Contributions The conceptual framework and requirements for an approach to collaborative EUA for evolving complex linked data are outlined in Chapter 3. Based on this foundation, we design generic concepts and generate generic knowledge which serves as the main research contribution of this thesis. A prototypical implementation serves as a proof of concept, while the evaluation iterations showed the utility of the prototype and practicability of the approach.

5. Research Rigor Chapter 2 reports on the foundations and related research of our approach, e.g., end-user-driven data visualizations (Chi and Riedl, 1998), visual information analysis (Isenberg et al., 2008), end-user-driven model analysis (Roth et al., 2013), and general EUA (Tamm et al., 2013). Based on this, we derive a conceptual framework for collaborative EUA and subsequently derived concrete requirements for a respective approach. Furthermore, we conduct a state-of-the-art analysis by observing spreadsheets as de facto standard tools for EUA in corporate environments. Finally, we discuss how we performed the different iterations of the evaluation, and provide a detailed description on the respective results and findings.

6. Design as a Search Process The IT artifact was iteratively developed and evaluated. In each evaluation phase, we also identified open issues and deficiencies of the prototype which were the input for the redesign of the artifact. For example, in the first iteration we identified deficiencies regarding the expressiveness of the query language for defining EA metrics. As a consequence, we utilized those findings by implementing additional operations and by extending the language's type system accordingly (cf. Sections 4.2.3). In the following iterations, we proceeded analogously, whereas the artifact represents the result of a continuous design and redesign process.

7. Communication of Research We published preliminary and partial results of different phases of the design process to conferences with a focus on Information System (IS) and Software Engineering (SE) and discussed our work with professionally like-minded researchers in related fields. For example, we published findings of the state-of-the-art analysis of today's EUA

tools (Reschenhofer and Matthes, 2015b,a; Reschenhofer et al., 2016c,d), and UI-related aspects of our approach (Reschenhofer et al., 2016b; Reschenhofer and Matthes, 2016b), and general lessons learned from applying meta-model-based information systems (Reschenhofer et al., 2016a).

1.4. Contributions of this Thesis

Figure 1.3 summarizes the contributions of this thesis and relates them to the research questions raised in Section 1.2, to the chapter of this thesis in which they are discussed and described, to concrete research artifacts, and to (peer-reviewed) publications which we created in the context of this thesis.

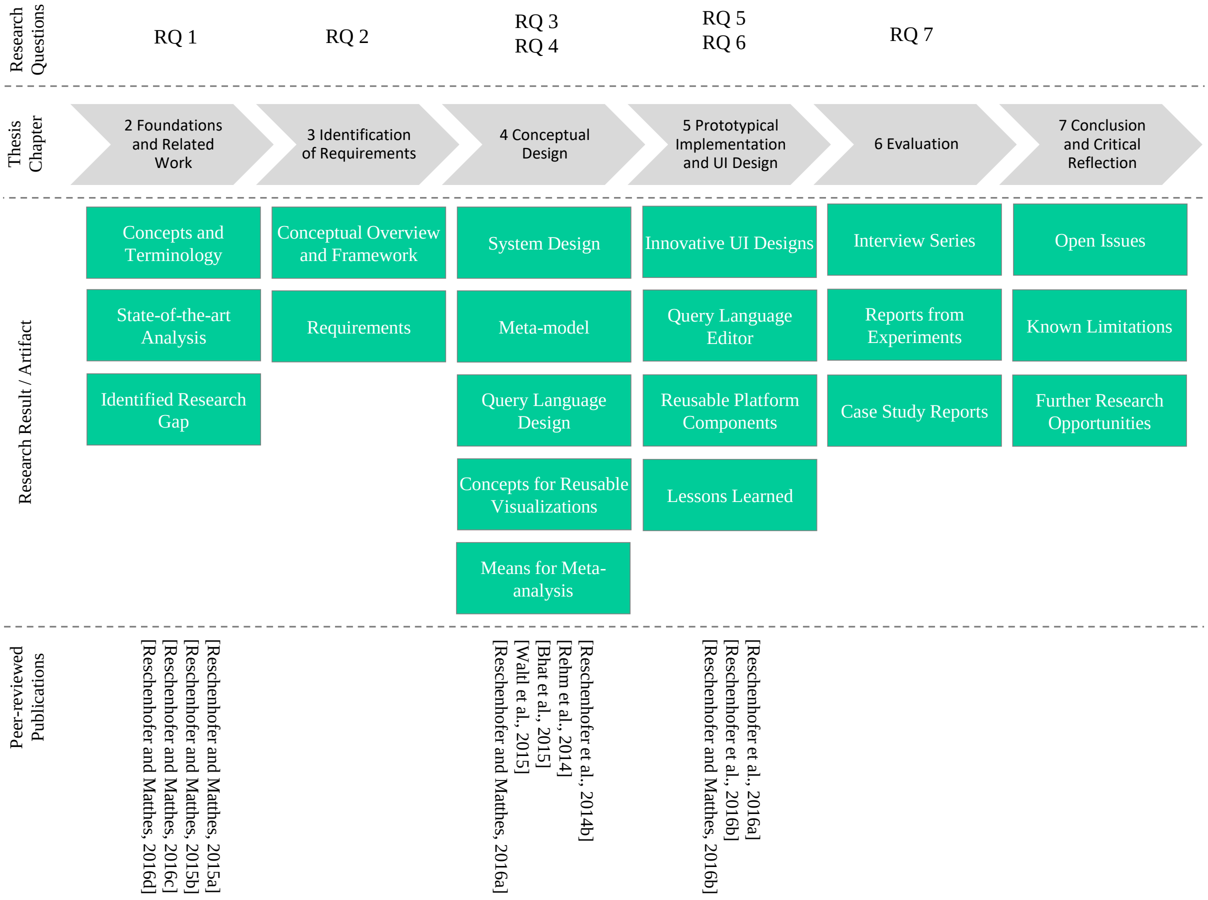

Research Questions
RQ 1
RQ 2
RQ 3
RQ 4
RQ 5
RQ 6
RQ 7
Thesis Chapter
2 Foundations and Related Work
3 Identification of Requirements
4 Conceptual Design
5 Prototypical Implementation and UI Design
6 Evaluation
7 Conclusion and Critical Reflection
Research Result / Artifact
Concepts and Terminology
State-of-the-art Analysis
Identified Research Gap
Conceptual Overview and Framework
Requirements
System Design
Meta-model
Query Language Design
Concepts for Reusable Visualizations
Means for Meta-analysis
Innovative UI Designs
Query Language Editor
Reusable Platform Components
Lessons Learned
Interview Series
Reports from Experiments
Case Study Reports
Open Issues
Known Limitations
Further Research Opportunities
Peer-reviewed Publications
[Reschenhofer and Matthes, 2015a]
[Reschenhofer and Matthes, 2015b]
[Reschenhofer and Matthes, 2016c]
[Reschenhofer and Matthes, 2016d]
[Reschenhofer et al., 2014b]
[Rehm et al., 2014]
[Bhat et al., 2015]
[Waltl et al., 2015]
[Reschenhofer and Matthes, 2016a]
[Reschenhofer et al., 2016a]
[Reschenhofer et al., 2016b]
[Reschenhofer and Matthes, 2016b]

Figure 1.3.: The main contributions of this thesis

The first contribution of this thesis is a comprehensive introduction and description of concepts related to EUA and meta-model-based information system as well as open issues of collaborative EUA on complex linked data. Thereby, we first describe EUA and meta-model-based information systems individually, and subsequently elaborate on how we intend to bring them together.

Furthermore, we conduct a state-of-the-art analysis of current EUA tools, and elaborate on how they are currently applied as information systems for the support of knowledge-intensive team work, and from which shortcomings they suffer in this context. This leads to the identification of a research gap which was already briefly introduced in Section 1.1. Based on this as well as on related research, we define a conceptual framework for collaborative EUA based on meta-model-based information systems. This framework forms the foundation for the identification of requirements (cf. Chapter 3).

The second contribution is the conceptual design and prototypical implementation of a tool which supports the collaborative EUA of evolving complex linked data structures. The conceptual design phase includes the derivation of a system design from the requirements as identified Chapter 3 as well as the development of conceptual meta-model capturing all concepts required for EUA based on meta-model-based information management. Moreover, we present the design of a query language which empowers end-users to define metrics and configure visualizations at run-time and elaborate on essential properties of this language in the context of evolving complex linked data structures. To address awareness issues in a collaborative environment, we developed concepts to enable end-user-driven meta-analysis of data, analysis, and visualization artifacts as well as their relations to the users of the system (cf. Section 4.4).

The third contribution arises from the evaluation of our approach and prototype in experiments and case studies as described in Chapter 6. The thesis reports on specific findings from applying the prototype for the definition and evaluation of complexity metrics in EAM, demonstrating it to practitioners in the domains of Enterprise Architecture Management, Financial Risk Management, and Data Quality Management, conducting experiments with students, researchers, and practitioners, and utilizing the tool for ad-hoc EA analysis in a corporate environment. Those findings particularly include open issues and shortcomings of the current prototype. On the top of this, the thesis reports on lessons learned from the implementation of our approach.

1.5. Outline of this Thesis

The structure of this thesis is primed by the problem statement and the research methodology as described in Sections 1.1 and 1.3 respectively. As illustrated in Figure 1.3, the thesis is divided into the following chapters:

Chapter 1: Motivation and Introduction motivates the thesis and describes the concrete problem statement which is addressed. Furthermore, this chapter reports on derived research questions and summarizes the main contributions which are implied by them.

Chapter 2: Foundations and Related Work introduces to related research which forms the foundations for this thesis' approach. It also contains a state-of-the-art analysis of how spreadsheets as de facto standard EUA tools are currently applied for supporting knowledge-intensive team work. Additionally, this chapter gives an overview over related research about similar approach, and differentiates them from ours.

Chapter 3: Identification of Requirements presents a conceptual framework of how we perceive collaborative EUA based on related research in the field of end-user-driven data visualizations (Chi and Riedl, 1998), visual information analysis (Isenberg et al., 2008), end-user-driven model analysis (Roth et al., 2013), and general EUA (Tamm et al., 2013). Based on this, the chapter describes how we derived a set of requirements which were drivers for the conceptual design of our solution.

Chapter 4: Conceptual Design elaborates on different aspects of the design of a system for collaborative EUA on evolving complex linked data structures. It covers the conceptual architecture and conceptual meta-model as well as the design of a formal language empowering end-users to define complex queries, e.g., metrics in EAM.

Chapter 5: Prototypical Implementation and User Interface Design presents selected highlights of the prototype's implementation. This particularly refers to UI-related design decisions, e.g., the design of a query editor and the design of an end-user explorable meta-analysis surface. Furthermore, the thesis also describes the concrete system architecture which extends the architecture of an already existing Hybrid Wiki prototype (Matthes et al., 2011).

Chapter 6: Evaluation reports on how we conducted multiple evaluations during the course of the design and prototypical implementation of our approach. The chapter also highlights respective key findings and how they drove the redesign of the prototype.

Chapter 7: Conclusion summarizes the main contributions of the thesis, and comprehensively and critically reflects on the research process and results regarding potential threats. Furthermore, the last chapter outlines possible future research activities which are enabled by this thesis.

CHAPTER 2

Foundations and Related Work

This chapter provides the foundation for the derivation of requirements in Chapter 3 and represents an overview over the related knowledge base in the sense of Hevner et al. (2004).

This thesis' main contribution is the development of an approach empowering end-users to collaboratively analyze evolving complex linked data. As shown in Figure 2.1, our approach is developed by bridging the gap between two complimentary fields of research, namely meta-model-based information systems on the one hand, and tool-support for End-User Analytics (EUA) on the other hand. Meta-model-based information systems address the support for adaptable and evolving data structures, while EUA is about tools empowering end-users to analyze data in a self-directed way. We introduce those two research fields in Sections 2.1 and 2.2 respectively. Section 2.3 provides an overview over related research approaches which are also considered as a combination of meta-model-based information systems and EUA approaches, although with different focuses.

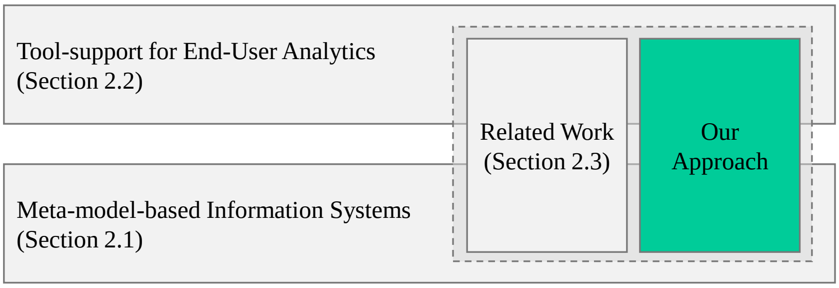

Tool-support for End-User Analytics
(Section 2.2)
Meta-model-based Information Systems
(Section 2.1)
Related Work
(Section 2.3)
Our
Approach

Figure 2.1.: An illustration of topics representing foundations and related work and their relation to the thesis' approach.

2.1. Adaptive Information System

Adaptive Information Systems (AIS) address the problem of changing requirements due to various factors, e.g., new legal regulations, or technological innovations. They provide means which allow end-users to quickly tailor them to the changing demands relevant to a particular domain. This is often referred to as End-User Development (EUD) or Domain Expert Configuration (DEC) (Mørch et al., 2004; Vitharana, 2003). Thereby, domain experts adopt a consulting role and configure the information system to the (potentially evolving) domain-specific environment (Følstad, 2007). This not only means a reduced effort and thus reduced costs when adapting the software at run-time, but also improves the efficiency due to the better fit with the domain-specific setting (Spahn et al., 2008a; Wulf and Jarke, 2004). Utilizing user creativity and user acquired habits for the continuous redesign are further motivations for EUD and DEC (Costabile et al., 2003). While not being able to adapt information systems at run-time to changing environments might lead to a decrease of the quality of support (van Oosterhout et al., 2006), EUD can impose significant challenges to a system's user as well (Pinnington et al., 2007). For example, EUD requires additional training for end-users, and an increased awareness of the implications of adaptions to the software.

2.1.1. Meta-model-based Information Systems

One elementary aspect of information systems which usually is changed very frequently is its data model (McGinnes and Kapros, 2015). Addressing this issue, meta-model-based information systems enable end-users to adapt (domain-specific) data models at run-time and to tailor them to changing environments. A meta-model describes a "model of a data model" and thus captures the means by which end-users can define and manipulate data models. In this sense, the domain-specific structures of a data model can be implemented and adapted at run-time and are decoupled of the information system's implementation. This system property is called conceptual independence (McGinnes and Kapros, 2015). Figure 2.2 illustrates how model-based information systems compare to meta-model-based information systems in terms of the data model's adaptability.

In general, there are two co-existing user roles for managing data models in meta-model-based information systems (Spahn et al., 2008a; Wulf and Rohde, 1995): *Model designers* (or *Modeling experts*) define and adapt the data model, and *data owners* maintain the data and perform changes to it. As illustrated in Figure 2.2, data is defined by data owners and consists of entities defined through their attributes and relations, while the data model imposes a given structure and constraints to the data. In meta-model-based information systems, the data model can be defined at run-time by model designers based on the system's meta-model. Therefore, in this kind of system, data and model co-evolve by a traditional top-down approach, by an agile bottom-up approach, or by a combination of them (Reschenhofer et al., 2016a).

In the top-down (*model-first*) approach, the co-evolution of the data and its model is triggered by model designers. This means that one or multiple model designers define the data model based on initial requirements, and tailor it if those requirements change. As a consequence, data owners have to adapt the data for which they are responsible for in order to keep it consistent

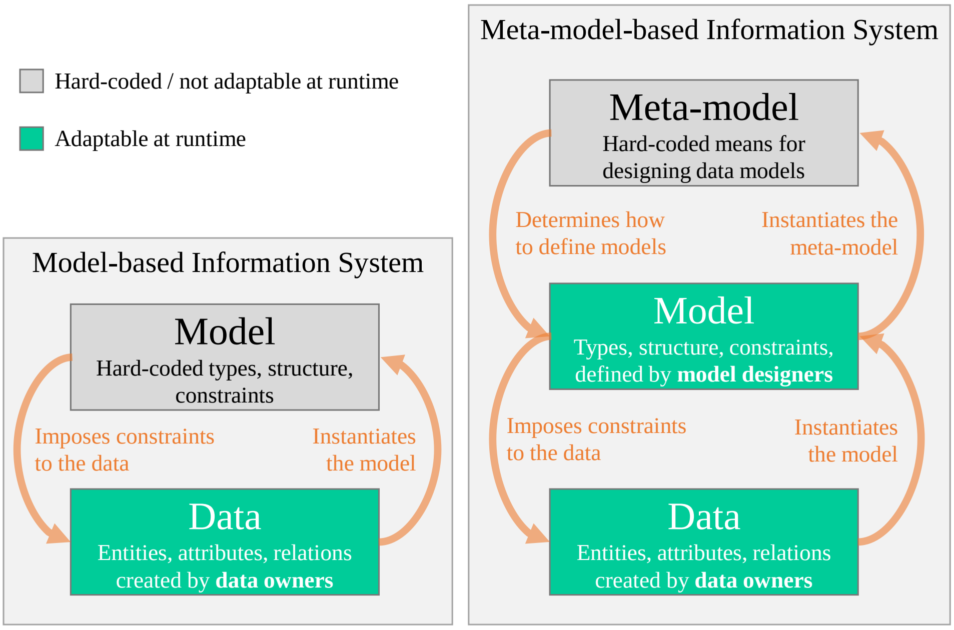

Hard-coded / not adaptable at runtime
Adaptable at runtime
Model-based Information System
Model
Hard-coded types, structure, constraints
Imposes constraints to the data
Instantiates the model
Data
Entities, attributes, relations created by data owners
Meta-model-based Information System
Meta-model
Hard-coded means for designing data models
Determines how to define models
Instantiates the meta-model
Model
Types, structure, constraints, defined by model designers
Imposes constraints to the data
Instantiates the model
Data
Entities, attributes, relations created by data owners

Figure 2.2.: model-based information systems vs. meta-model-based information systems

with the changing data model. In this way, the top-down approach structures the problem domain and enforces data owners to ensure consistency of the data and its model.

On the other hand, in the bottom-up (*data-first*) approach, the data is captured and structured before an actual data model is defined. Based on this and supported by the data modeler, a data model and data model adaptions can be derived semi-automatically from existing data structures. This allows data owners to contribute with their domain-specific knowledge to the design of the data model and reduces the number of hand-shakes required in updating the data model.

In cases where the data model is not known from the start, a hybrid approach might be the most appropriate (Reschenhofer et al., 2016a). This means that in early stages, the bottom-up approach enables model designers to harness collective intelligence of domain experts to design the model. If the data model evolves and matures, a switch to a top-down approach helps model designers to control the evolution of the data model and to reach a desired target model. Hybrid Wikis as defined by Matthes et al. (2011) support such a mixed approach.

2.1.2. Hybrid Wikis

The concept of Hybrid Wikis was introduced by Matthes et al. (2011) as a knowledge management means based on the common Web 2.0 concept of wikis. Originally, the term *Hybrid* referred to the fact that it only integrates a subset of semantic wiki features (Krötzsch et al., 2006). However, with the modeling approach of model-first and data-first in mind, *hybrid* can also express a mixed approach to data modeling. Hybrid Wikis support both approaches simultaneously, and model designers can switch between them seamlessly.

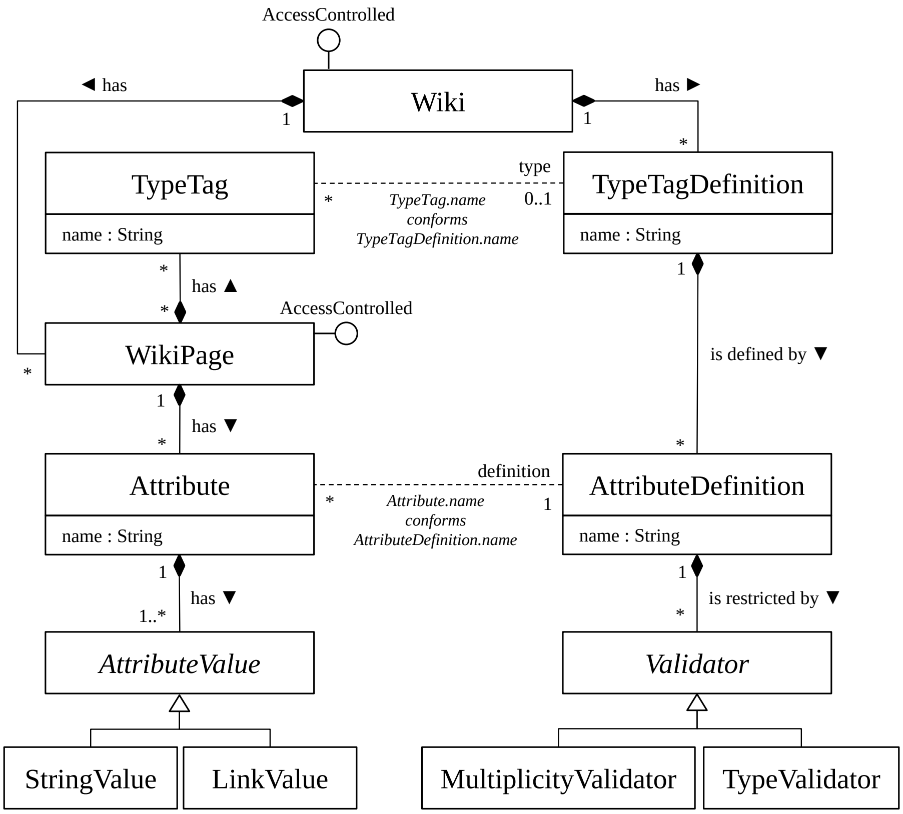


Figure 2.3.: The original Hybrid Wiki meta-model as defined by Matthes et al. (2011). The implementation of the *AccessControlled* interface is illustrated in Figure 2.5.

Figure 2.3 shows the original conceptual Hybrid Wiki meta-model by Matthes et al. (2011). Based on traditional wiki systems, it captures the concepts of *Wiki*s and *Wikipages* which are connected through a composition association. One of the characterizing features of Hybrid Wikis is the possibility to attach *Attributes* to *Wikipages*, which in turn can have potentially multiple *AttributeValues*—either simple *StringValues* or *LinkValues* representing relations to other *Wikipages*. In this sense, those concepts represent the data of the Hybrid Wiki system. The Hybrid Wiki system's data model is composed by *TypeTagDefinitions* which are defined through *AttributeDefinitions*. *AttributeDefinition*s define potentially multiple validators which impose certain constraints to corresponding *Attribute*s. For example, *TypeValidator*s can force corresponding *Attribute*s to be of a particular type, while *MultiplicyValidators* enforce a particular cardinality (e.g., there has to be at least one *AttributeValue* for the corresponding *Attribute*). The data is connected to its model via so-called *TypeTags*. In this way, data owners can assign *TypeTags* to a given *WikiPage*. As a consequence, the *WikiPage* has to comply to the structure as defined by the *TypeTagDefinition* whose name matches with the *TypeTag* within the same *Wiki*. This means that all of the *WikiPage*'s *Attribute*s which have a corresponding *AttributeDefinition* (again, *Attribute*s and *AttributeDefinitions* are matched by their names) have to comply to the respective *Validators* as defined by the model designer. It is worth noting that within the Hybrid Wiki meta-model, data and model concepts are defined on the same meta level. This means that from a conceptual point of view, *WikiPages* and *TypeTagDefinitions* are connected through intra-level associations instead of inter-level "instance of" relationships (Geisler et al., 1998). Consequently, there can be *WikiPages* which are not assigned to any *TypeTagDefinition*.

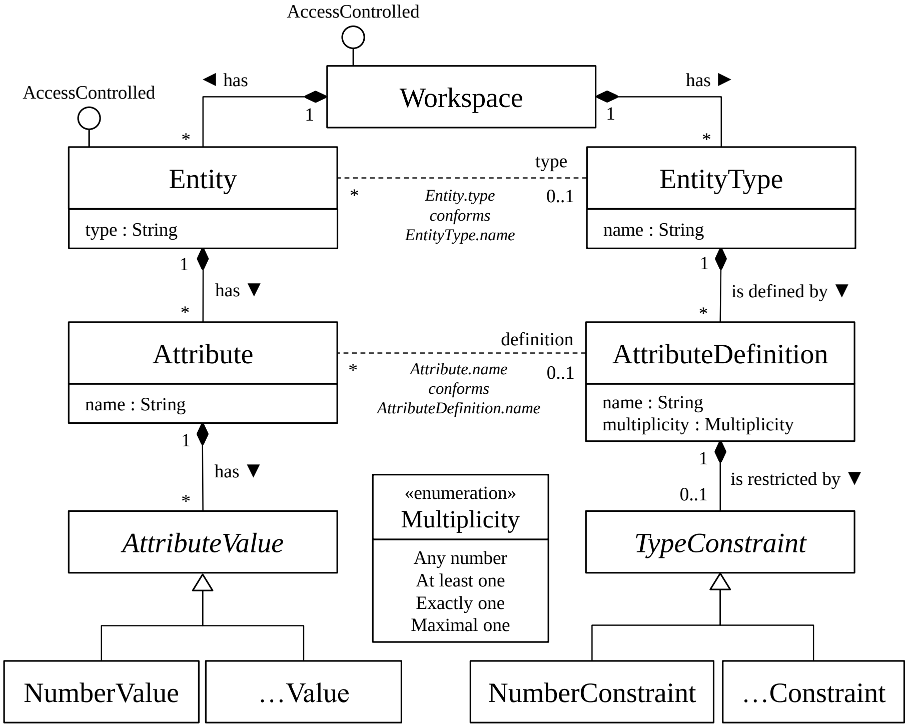


Figure 2.4.: The updated Hybrid Wiki meta-model by Reschenhofer et al. (2016a). The implementation of the *AccessControlled* interface is illustrated in Figure 2.5.

After more than five years of applying the Hybrid Wiki system in different domains (e.g., Enterprise Architecture Management and New Product Development), the feedback of users led to the need for changes of the Hybrid Wiki meta-model (Reschenhofer et al., 2016a). The updated Hybrid Wiki meta-model is depicted in Figure 2.4. First of all, Hybrid Wikis were applied not only as knowledge management tool, but also as user-driven and model-based data repository. As a consequence, users of the Hybrid Wiki system were referring to information objects as *Entities* instead of *WikiPages*. This more general designation of information objects also detaches them from their specific representation as wiki pages. At the same time, *Wikis* were renamed to *Workspaces*, and *TypeTagDefinitions* were renamed to *EntityTypes*. Renaming concepts seems to be a negligible adaption of the meta-model. However, collaborative information management requires to communicate with co-workers using terms and concepts as provided by the information system. Therefore, the terminology of the meta-model and the naming of concepts is of a significant relevance.

Another change to the Hybrid Wiki meta-model was the removal of the *TypeTag* concept as an association class between *WikiPages*/*Entities* and *TypeTagDefinitions*/*EntityTypes*. Furthermore, the Hybrid Wiki system's application revealed that the possibility of assigning multiple *TypeTags* to one *WikiPage* was perceived as unintuitive on the one hand, and caused distraction in case of conflicts regarding constraints of *Attributes* on the other hand (e.g., if there are two or more *AttributeDefinitions* with the same name but with divergent *TypeValidators*). In the updated meta-model, the association between *Entities* and *EntityTypes* was modeled as a many-to-one relationship. However, data owners can still create *Entities* which are not assigned to any *EntityType* and thus do not underlie any restrictions defined by model designers. Analogously, data owners can also add *Attributes* to *Entities* which are not (yet) defined as corresponding *AttributeDefinitions*. Those *Attributes* are also called *Free Attributes*.

Based on the feedback gained from the Hybrid Wiki system's application, the set of attribute types was extended. In total, the updated Hybrid Wiki meta-model supports the following attribute types, i.e., for each type *T* there is a respective *TValue* and *TConstraint* class:

String An arbitrary string which can be further restricted by model designers by either providing a regular expression or a predefined set of enumeration values.

Link A reference to another entity. Model designers can specify *LinkConstraints* in a way that only references to *Entities* of a specific *EntityTypes* are allowed.

Number Both integers and decimals.

Boolean Either *true* or *false*.

Date A date consisting of a day, month, and year component.

Since supporting attributes with multiple values as well as multiple relations to other data entities, the Hybrid Wiki system only lacks support for arbitrarily nested attributes in order to fit the definition of complex linked data structures from Chapter 1. Therefore, by extending the Hybrid Wiki meta-model accordingly (cf. Section 4.1.1), it qualifies as conceptual and technical foundation for this thesis' approach and prototype.

On the top of data management and structuring concepts, the Hybrid Wiki system also provides the following relevant basic collaboration and content management features (Neubert, 2012):

Access Control To control which users are allowed to see and/or edit entities, the Hybrid Wiki system implements access control lists. Thereby, administrators of an entity specify a set of users or groups which are allowed to read (*readers*), edit (*writers*), or set access rights (*administrators*) of this entity. Those access rights can also be set on the level of workspaces specifying a default behavior for the entities of a workspace.

Figure 2.5 illustrates the authorization model. The *AccessControlled* interface represents objects which are controlled by access control lists. In the Hybrid Wiki meta-model in Figure 2.4, the *Entity* and *Workspace* classes implement this interface. *Users* and *Groups* are subsumed under the abstract class *Principal* by using the composite pattern (Gamma et al., 1994), which can be assigned to an access-controlled object either as readers, writers, or administrators.

Version History The Hybrid Wiki system automatically tracks the evolution of entities within their life-cycle which is referred to as *version history* (Büchner, 2007). Thereby, for each

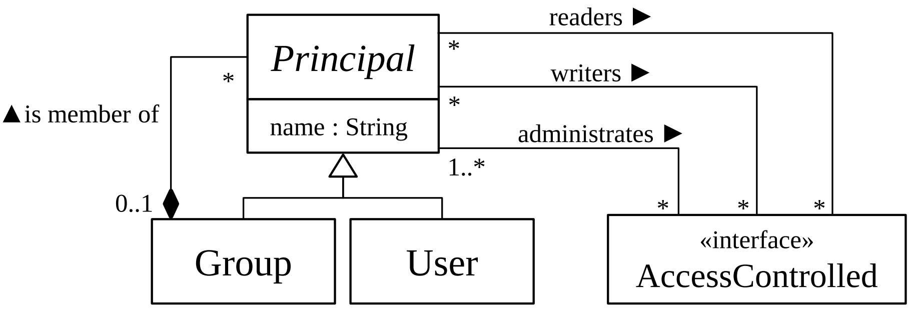


Figure 2.5.: The authorization model of Hybrid Wikis. Both the *Entity* and the *Workspace* in Figure 2.4 implement the *AccessControlled* interface.

entity the system maintains a chronologically ordered list of change events which capturing contextual meta-information, e.g., the time when the event occurred, or the user who triggered the event.

File Attachments Users can attach an arbitrary number of files to entities. Those files are also captured by the system's access control and version history mechanisms.

Full Text Search with Facets Since originally designed as a knowledge management tool, the Hybrid Wiki system empowers users to find entities by full text search capabilities. Furthermore, the search results can be refined and filtered by using facets addressing the structure properties of the entities in the result set.

Watching The Hybrid Wiki system allows users to explicitly *watch* entities of interest, i.e., users are notified if specific events are triggered for this entity, e.g., if it is changed.

In this sense, the Hybrid Wiki system does not only provide useful properties regarding the iterative definition of data models, but also a variety of collaboration features. This further underlines the Hybrid Wiki system's applicability as conceptual and technical foundation for a collaborative EUA tool as intended by this thesis.

2.2. Tool-support for End-User Analytics

The digital transformation of today's enterprises leads to an increasing importance of knowledge workers which are able to perform complex information management and analysis tasks autonomously and collaboratively (Davenport, 2013). On this note, Tamm et al. (2013) define End-User Analytics (EUA) as a sub-domain of End-User Development (EUD) empowering end-users to make evidence-based decisions by self-directed data analysis. They differentiate between two types of business analytics users: *Analytics Professionals* use powerful means and tools (e.g., data mining, simulation, prescriptive modeling) to perform sophisticated data science tasks (Davenport and Patil, 2012), while *Analytics End-Users* use mostly predefined reports, dashboards, and—most notably—spreadsheets. In the thesis at hand, we particularly focus on the latter user role which is empowered to create value from data by herself.

Spreadsheets provide versatility in the sense that they empower end-users to structure, analyze and visualize data by themselves (Pemberton and Robson, 2000). However, spreadsheets were designed with a *"focus on solving problems aiding in personal decision making, and increasing personal productivity"*(Senn, 2004). Therefore, traditional spreadsheets are EUA tools for individuals. Ironically, well-founded decision making should include multiple stakeholders with different expertises at different stages of the decision making process (Berthold et al., 2010). In this sense, collaborative EUA involves multiple end-users in defining the data model, analytical abstractions, and visualizations. This leads to new challenges (Kaufmann and Chamoni, 2014), particularly since a collaborative environment no longer matches the initial purpose of spreadsheets as personal productivity tools.

Nevertheless, spreadsheets are still the most-prominent and most-used software tool in general, and the most important EUA tool in particular. More than 90% of all desktops worldwide have spreadsheets installed (Bradley and McDaid, 2009), while a huge majority of enterprises in the United States and Europe use spreadsheets for different business analytics tasks (Panko, 2006). Due to their overwhelming quantity and importance, but at the same time their invisibility to an organization's IT departments, Panko and Port (2012) famously called spreadsheets the "*dark matter*" of corporate IT.

In this thesis, we aim for a novel EUA approach to empower end-users to analyze evolving complex linked data. As a consequence, we want to understand the success factors and shortcomings of current EUA tools. For this reason, we performed a state-of-the-art analysis of spreadsheets as the de facto standard among s EUA tool. In Section 2.2.1, we outline the characterizing features of spreadsheets which also represent the success factors of this kind of EUA tools. Subsequently, Sections 2.2.2, 2.2.3, and 2.2.4 summarize our findings from analyzing spreadsheets and their usage in different domains as well as shortcomings occurring in those contexts respectively.

2.2.1. Characterizing Features of Spreadsheets

According to Hermans (2012), the spreadsheet paradigm is the most successful programming paradigm. Even in 2005, Scaffidi et al. (2005) estimated that the number of end-user programmers in general and spreadsheet users in particular is much higher than the number of professional programmers.

The first spreadsheet software system—VisiCalc —was released in 1979. It already supported interaction features which still characterize today's spreadsheet software. IBM and Microsoft followed by building and releasing IBM Lotus 1-2-3 and Microsoft Excel respectively. There are even more spreadsheet tools and approaches which were developed in the last decades (Abraham et al., 2008). Most of them share a common set of features which characterize spreadsheets and represent the success factors of the spreadsheet paradigm:

Flexible Data Management The study by Pemberton and Robson (2000) shows that spreadsheets are heavily used for flexible data management, e.g., to sort and organize data, or to apply advanced database functions. By using the spreadsheet grid as an intuitive UI, end-users are able to maintain and structure huge data sets primarily consisting of textual and numerical data (Chambers and Scaffidi, 2010). While the flexibility regarding the data structures is a characterizing feature of spreadsheets, the lack of an explicit data model makes it difficult for users to understand a spreadsheet's design which in turn can lead to errors (Hermans, 2012). Section 2.3.1 outlines approaches for the explicit definition of spreadsheet models.

Analytical Functions The most obvious characteristic of spreadsheets is the empowerment of end-users to define calculations by themselves (Hermans, 2012). Thereby, the grid based data reference model turned out to be a very intuitive and user-friendly way of defining formulas based on the spreadsheet's data (Abraham et al., 2008). Users mostly use simple formulas and operations (Chambers and Scaffidi, 2010), e.g., simple arithmetic calculations. However, a considerable proportion of spreadsheet users also applies more sophisticated analytical functions, e.g., regression and correlation analysis and advanced statistical analysis (Pemberton and Robson, 2000).

Visual Analysis through Charts According to the study by Pemberton and Robson (2000), end-users utilize spreadsheets to create charts as graphical representations of their data and respectively derived analytical abstraction. Again, they mostly use basic charts types, e.g., bar charts, or line charts (Reschenhofer et al., 2016c). Chambers and Scaffidi (2010) argue that in certain cases there is also a need for customizations of existing charts or the creation of new "exotic" chart types.

Immediate Visual Feedback As a cross-cutting UI aspect, immediate visual feedback on interactions of the user with the spreadsheet is an essential feature of the spreadsheet paradigm (Abraham et al., 2008). Originally described by the term *"liveness"* (Tanimoto, 1990), visual feedback refers to the spreadsheet's automated recalculation and redrawing of formulas and visualizations respectively on changes of the artifact itself or one of its inputs. In this way, users can immediately check if the performed action leads to the intended outcome. Consequently, this feature forms the foundation for related research on improving the testability of spreadsheets, e.g., the "What You See Is What You Test" (WYSIWYT) methodology (Rothermel et al., 2000; Burnett et al., 2001). On a negative note, immediate visual feedback can also hinder effective problem solving, and it certainly does not guarantee the absence of errors (Svendsen, 1991; Wilcox et al., 1997).

Interoperability Interoperability with related systems, e.g., databases, word processors, or statistical software (Pemberton and Robson, 2000), is one of the deciding success factors of spreadsheets. Pemberton and Robson (2000) even claim that the position of Microsoft Excel as the market leader in the context of spreadsheet software is mainly due to its integration with other Microsoft products. In fact, in many cases companies do not even evaluate alternative spreadsheet solutions before turning to Microsoft Excel (Robson and Pemberton, 1996).

In Chapters 3 and 4 we will revive those features and elaborate on how they can be integrated into a collaborative and meta-model-based information system supporting evolving complex linked data.

2.2.2. Semantic Structures in Today's Spreadsheets

As a next step of the state-of-the-art analysis of spreadsheets, we outline our findings of analyzing semantic structures which typically occur in today's spreadsheets (Reschenhofer et al., 2016c). For this purpose, we analyzed 200 random spreadsheets of the Enron corpus (Hermans and Murphy-Hill, 2015) which consists of more than 15,000 spreadsheets extracted from emails of the Enron corporation (Klimt and Yang, 2004). Based on this analysis as well as mostly on the work done by Hermans et al. (2010), Erwig and Engels (2005), and Cunha (2011), we derived an integrated semantic meta-model of spreadsheets as shown in Figure 2.6.

This semantic meta-model captures the following structural patterns (Reschenhofer et al., 2016c):

Entity List This is the most common and most discussed semantic spreadsheet pattern. It represents multiple data objects organized in a table. An entity list is defined through its (potentially grouped) attributes which either represent input fields or derived attributes defined by formulas. This structure of classes conforms to the composite pattern (Gamma et al., 1994). While most entity lists (80 %) have a simple structure, some of them are more complex due to groupings, aggregations, or constraints.

Time Series Time series are an extension of entity lists and add a specific temporal timestamp attribute to the data objects which also implies an intrinsic order of data objects. Furthermore, the time series also marks a specific set of numerical attributes as its time series variables, based on which one can perform common time series analytics (Brockwell and Davis, 2009).

Singleton While entity lists represent an array of data objects, the singleton pattern refers to a single structured data item. Again, through the adoption of the composite pattern (Gamma et al., 1994), singleton attributes can be grouped (Knight et al., 2000).

Matrix Although only identified twice in 200 spreadsheets, we defined matrix as a spreadsheet structure having two fixed axes and respective values in the cells within the area spanned by those axes.

Chart We also interpret charts as a separate semantic component, although they are different from a syntactical point of view since they are not defined through a specific arrangement of data on the spreadsheet grid. We differentiate between different chart types, e.g., bar charts, line charts, or scatter charts.

Our analysis shows that entity lists are by far the most common semantic pattern in today's spreadsheets. Including time series as an extension of this pattern, they account for 75 % of all identified patterns. This means that current analytics end-users (Tamm et al., 2013) are mostly working with lists and tables of structured data objects. They define analytical abstractions usually either as derived attributes—calculations based on attributes of the same data object—or as aggregations, e.g., calculating the sum or average of mostly numerical values. This knowledge serves as input for the design of a collaborative EUA approach as described in Chapter 4.

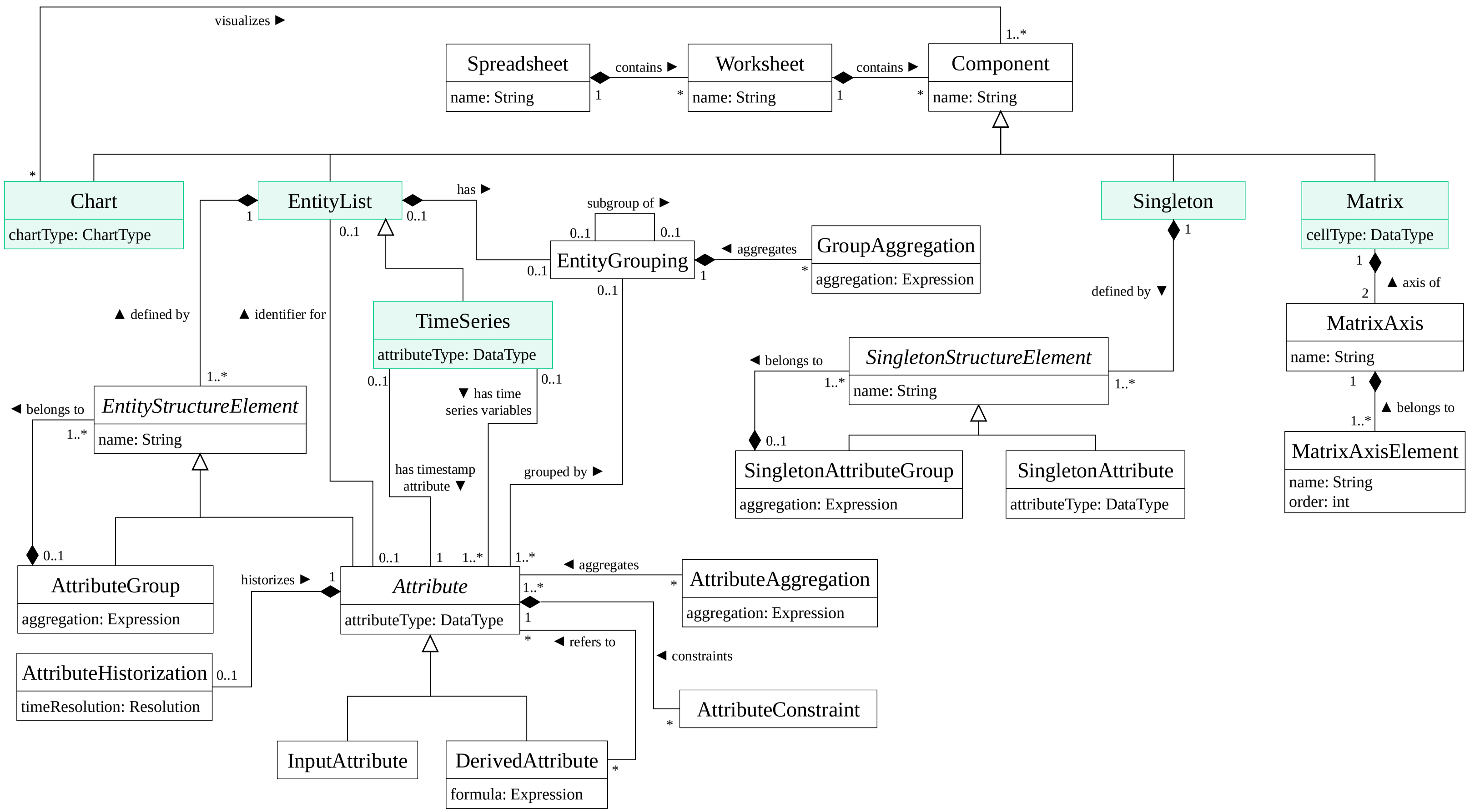


Figure 2.6.: An integrated semantic meta-model of spreadsheets (Reschenhofer et al., 2016c). The green classes represent identified semantic patterns.

2.2.3. Usage Patterns in Today's Spreadsheets

In the previous section, we examined semantic patterns capturing typical data structures users are working with. In this section, we study usage patterns describing how users actually utilize spreadsheets in practice. To this end, we conducted multiple case studies in two German companies (Reschenhofer and Matthes, 2015b). Firstly, we derived seven dimensions that enable the differentiation of spreadsheet usages based on findings in related work (Ronen et al., 1989; Simmhan et al., 2005; Hall, 1996; Lawson et al., 2009). The identified categories for distinguishing spreadsheet usages are as follows (Reschenhofer and Matthes, 2015b):

Design Context Captures the usage and temporal scope of the spreadsheet, i.e., if the spreadsheet is used only by its designer or also by other users, and if the spreadsheet is intended to be used as an intentional business application (Grossman et al., 2007) or just for solving one single problem instance (Ronen et al., 1989).

Data Origin Captures the provenance of the data (Simmhan et al., 2005) and the general type of the data source, i.e., if the input data is based on human-made assessments, or based on the output of other software systems.

Data Input Method Deals with the way how data is entered into the spreadsheet, e.g., manual input by users, tool-supported import from other systems.

Computational Complexity Differentiates between three levels of computational complexity (Hall, 1996), namely low (only elementary operations), medium (advanced statistical operations), and high (customized operations, e.g., through macros). Regarding the complexity of spreadsheets, we also did our own study (Reschenhofer et al., 2016d) in order to identify drivers to spreadsheet complexity, which could help to categorize spreadsheets based on a certain set of metrics.

Output View Captures the type of the spreadsheet's output, i.e., if the primary output of the sheet is one or multiple (unformatted) data tables, or visual representations consisting of charts and visually processed tables.

Data Consumer Analogous to the *Data Origin* dimension, the framework also deals with the consumer of the spreadsheet's output, which can be either another software system, or a human being.

Data Historization is a cross-cutting dimension and captures how the spreadsheet deals with historization (Lawson et al., 2009), e.g., if historized data is accumulated within one file, if spreadsheet files are version-controlled, or if no data historization is provided.

Together with respective categories, the dimensions represent the classification framework as a so-called *Morphological Box* (Zwicky, 1969) which is illustrated in Figure 2.7. By applying the classification framework to nine industrial use cases in two German companies, we identified three clusters consisting of spreadsheets with similar characteristics. Those clusters form the following generic spreadsheet usage patterns:

Reporting Sheets This usage pattern refers to spreadsheets which usually integrate data from

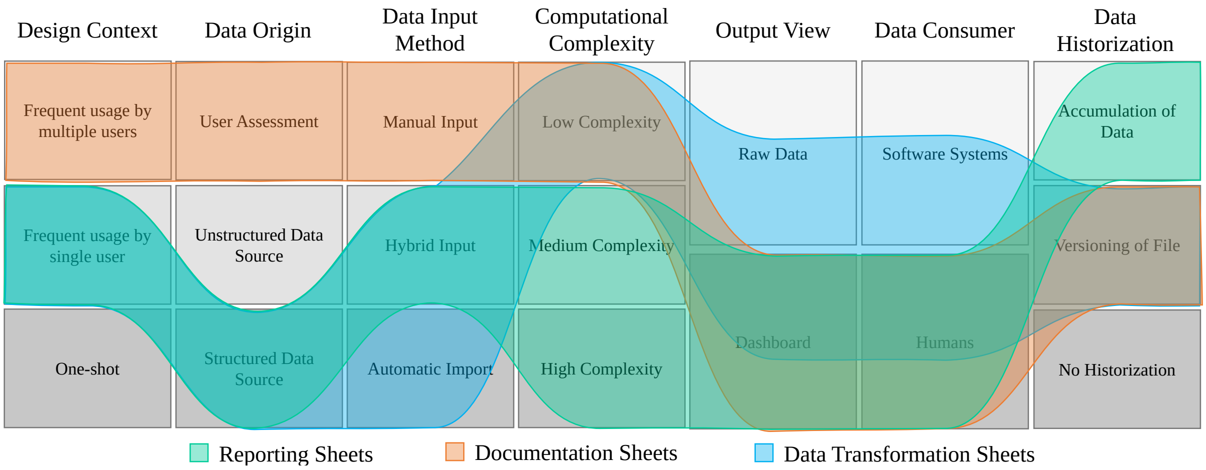


Figure 2.7.: The dimensions and respective categories of the framework for classifying the spreadsheet usages as well as identified usage patterns (Reschenhofer and Matthes, 2015b)

different sources, apply rather complex analytical operations, and generate a graphical representation which is consumable by a specific group of human stakeholders.

Documentation Sheets Those spreadsheets serve as end-user-friendly data management tools, in which usually data sets are maintained and viewed by one user group, and consumed by another one. If at all, they only contain very basic analytical operations and visualizations.

Data Transformation Sheets Spreadsheets of this kind serve as data manipulation tools whose purpose, e.g., is to restructure the output of a certain software system in order to make it importable by another tool.

It should be noted that there are further usage patterns which were not identified in the context of this work, e.g., data form sheets (Chambers and Scaffidi, 2010). Nevertheless, the developed classification framework helps to categorize spreadsheets regarding how they are actually applied in specific use cases. On the one hand, this categorization supports users to understand the nature of the use case and problem for which they apply the spreadsheet, and to compare it with similar spreadsheet applications. On the other hand, researchers and tool vendors can address specific usage patterns and thus pattern-specific issues. The novel EUA approach as presented in this thesis particularly addresses *Reporting Sheet* use cases.

2.2.4. Shortcomings of Spreadsheets from an IS Perspective

As the final part of the state-of-the-art analysis spreadsheets, we did research on problems and shortcomings when applying them as information systems to support EUA tasks.

In fact, there is already lots of scientific work on how to classify errors, detect them, and debug spreadsheets (Powell et al., 2008). With respect to classification, Ronen et al. (1989), Panko and Halverson Jr. (1996), and Rajalingham et al. (2000) define taxonomies and models to categorize errors, e.g., into quantitative and qualitative errors, while the former refers to errors which lead to wrong numbers, and the latter defines a design issue which might cause other quantitative errors. Regarding the identification and elimination of errors, there are multiple approaches to enable user-friendly and integrated spreadsheet testing (Fisher et al., 2002; Panko, 1999; Rothermel et al., 2000) and debugging (Abraham and Erwig, 2005; Ayalew and Mittermeir, 2003; Kankuzi and Sajaniemi, 2013). Those technical errors can have a considerable impact on an organization's business operation and thus lead to significant financial losses (Caulkins et al., 2007; Powell et al., 2009). In most cases users are not even aware of those errors and problems (Panko, 2015).

In addition to those technical errors, there are also shortcomings from an IS perspective. With IS perspective, we refer to the holistic observation of spreadsheets in their business environment including technology, people, and business processes. This means that spreadsheets do not have to be erroneous as such, but might not support specific business processes adequately due to technical debts. For example, spreadsheets were originally designed for "solving problems, aiding in personal decision making, and increasing personal productivity" (Senn, 2004). However, in practice, spreadsheets are often used as "intentional" applications (Grossman et al., 2007), i.e., spreadsheets which are purposefully developed and deployed to multiple users instead of using for personal purposes only. Consequently, the lack of multi-user support (Ginige et al., 2010a) leads inevitably to shortcomings with respect to the spreadsheet's support for the knowledge-intensive team work, although it might contain no errors.

To get an overview over this kind of shortcomings from an IS perspective, we observed current spreadsheet applications in two companies as part of a study including nine cases already described in Section 2.2.3 (Reschenhofer and Matthes, 2015a). In total, we identified 20 shortcomings which we classified into the following six categories:

Readability and Understandability Readability and understandability issues were already discussed in related research (Hermans et al., 2012; Cunha et al., 2012d) as a main driver to the risk of errors. As revealed by the case studies, understandability is an issue on different levels of abstraction. For example, interviewees mentioned a lack of formatability and commentability of formulas as a concrete shortcoming since this hampers reading and understanding concrete calculations. On a more general level, the lack of transparency of a spreadsheet's dependencies—constituted by its formulas and cell references—hinders to comprehend the overall design of a spreadsheet (Hermans et al., 2011b) and was also named as a related shortcoming.

Extendability The case studies revealed that interviewees found it difficult to extend the spreadsheet software by additional data integration means, analytical operations, and customized chart types. Although most spreadsheet software provide means to extend the computational expressiveness through macros, the interviewees consider their imperative paradigm as not suitable for defining functional operations which can be reused later on by end-users.

Manageability This category refers to shortcomings about designing and maintaining spreadsheets, particularly when using them over a longer period of time (Grossman et al., 2007). One concrete shortcoming observed in the context of the cases studies was the lack of modularity. This is related to the reuse of spreadsheets as already discussed by Chambers and Scaffidi (2010). Furthermore, a lack of support for Application Lifecycle Management (ALM) including designing, testing, and debugging spreadsheets was also mentioned as a shortcoming by interviewees. This, however, is heavily discussed in related spreadsheet research as described in the introducing paragraph of this section.

Collaboration and Multi-user Support Due to their original purpose of serving as a tool for personal problem solving and decision making (Senn, 2004), traditional spreadsheet software naturally suffers from shortcomings regarding multi-user and collaboration support. This includes the lack of means to efficiently design spreadsheets in a collaborative way, to set access rights to individual cells, or to define user-specific views.

Data Data-related shortcomings as revealed in the context of the case studies include the lack of support for complex data types as well as database-like spreadsheet queries. The former issue was already discussed in Chapter 1, while the latter one was also subject to related research (Cunha et al., 2013, 2014b).

Processes Process-related shortcomings of spreadsheets mainly refer to a lack of support for process integration, e.g., to trigger a specific action based on a specific spreadsheet event.

Table 2.1 summarizes the identified shortcomings and indicates in how many of the nine cases the shortcomings was mentioned by the respective interviewee.

The EUA approach based on Hybrid Wikis as described in this thesis intends to address most of those shortcomings, particularly the lack of transparency, extendability, multi-user support, and support for complex data. Therefore, the insights gathered from the studies about spreadsheet shortcomings from an IS perspective are an essential contribution to the definition of requirements and the design of the approach in Chapters 3 and 4 respectively.

2.3. Related Work

The previous Sections 2.1 and 2.2 summarize the foundations for our approach, namely meta-model-based information systems and AIS in general as well as EUA and spreadsheets as the most prominent representative of EUA tools in particular. Before combining both domains and deriving requirements for a novel EUA approach to analyze complex linked data, we provide an overview over related approaches. In fact, we identified three different research areas which are related to our work: Research on spreadsheets (cf. Section 2.3.1), collaborative BI (cf. Section 2.3.3), and end-user-driven and model-based analytics and visualizations (cf. Section 2.3.2).

2.3.1. Related Work based on Spreadsheets

Spreadsheets have already been studied for more than 25 years (Panko, 2006). Spreadsheet research is mostly about errors and how to avoid, identify, and fix them (Powell et al., 2008).

In the context of this thesis we propose a novel EUA approach for analyzing complex linked data. Since spreadsheets already provide proven means for EUA, the most obvious approach would be to build our solution on the foundations of spreadsheet software. Therefore, we will provide

Shortcoming		# of Cases
Readability and Understandability		
01	Formatability and Commentability of Formulas	3
02	Named Cell Addressing	3
03	Transparency of Information Flow	2
Extendability		
04	Integration of Custom Data Sources	3
05	Extendability of Computational Expressiveness	5
06	Custom Visualizations	6
Manageability		
07	Managed Evolution of Spreadsheets	3
08	Separation of Data, Schema, and Logic	5
09	Modularity / Reusability	2
10	Application Lifecycle Management	5
Collaboration and Multi-user Support		
11	Collaborative Spreadsheet Design	5
12	Element-based Access Control	7
13	Element-based Historization	8
14	User-specific Views	2
Data		
15	Complex Data Types	6
16	Scalability	6
17	Spreadsheet Queries	1
18	Custom Spreadsheet Meta-data	1
Processes		
19	Support for Automation	1
20	Reasoning of Derived Actions	2

Table 2.1.: A consolidated list of identified shortcomings as well as an indication of in how many of the nine observed cases it was mentioned by a respective interviewee (Reschenhofer and Matthes, 2015a).

an overview over related research in the field of spreadsheets in general, how related work on spreadsheets addresses the issues outlined in Section 1.1, and how our approach differentiates from existing ones. More specifically, related research in the field of spreadsheets is about model-based spreadsheet engineering as well as spreadsheets in multi-user environments.

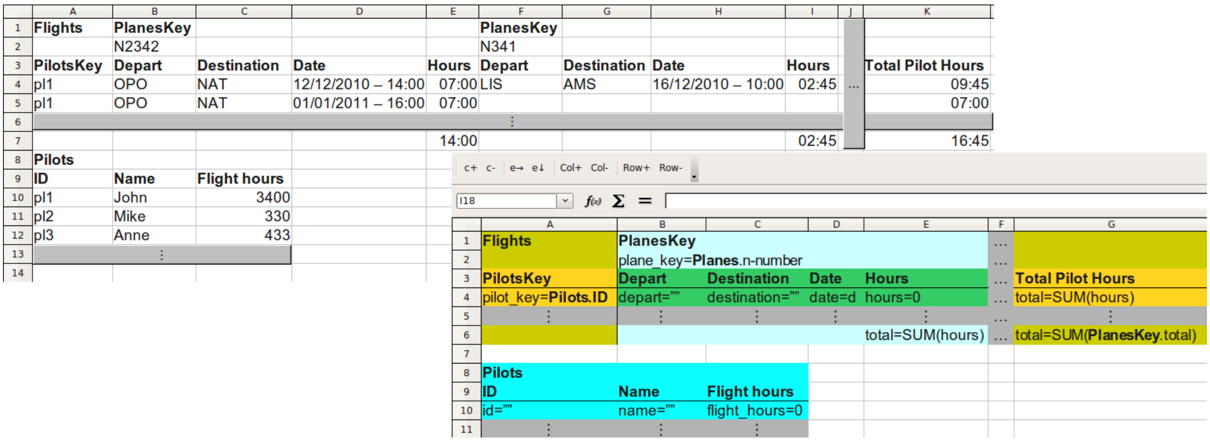
2.3.1.1. Model-based Spreadsheet Engineering

Addressing the avoidance of spreadsheet errors, model-based spreadsheet engineering (Cunha, 2011) aims for integrating the concept of structured data models with spreadsheets. Thereby, spreadsheet modeling takes into account specific spreadsheet aspects, e.g., column arrangements, and guides users in editing the spreadsheet data in a correct way (Cunha et al., 2014d). Beckwith et al. (2011) have shown in their study that model-based spreadsheets can improve the effectiveness of end-users and reduce the risk of errors significantly. This is particularly positive in the light of legal and compliance regulations which address the risks and impacts of spreadsheets and their errors when applied for supporting critical business processes (Panko and Ordway, 2005; Leon et al., 2010).

Erwig and Engels (2005) developed one of the first approaches to spreadsheet modeling, namely *ClassSheets*. ClassSheets represent spreadsheet templates which capture the structure and relationships of entities as well as descriptions of how attributes are derived from each other (cf. derived attributes as defined in Section 2.2.2). In this context, spreadsheet modelers use the Visual Template Specification Language (ViTSL) to define ClassSheets as a structure of vertically and horizontally composable and recurring cell blocks (Abraham et al., 2005). In addition to ensuring the compliance of spreadsheet instances to ClassSheet models and the resulting reduction of the risk of errors, the ClassSheet approach is explicitly encouraging the reuse of components and thus directly addresses shortcoming 09 of Table 2.1.

Cunha et al. (2012c) extend the ClassSheet approach and developed the *MDSheet* framework which embeds ClassSheet models in a spreadsheet system in order to provide a coherent environment, i.e., end-users use the spreadsheet interface to define spreadsheet models as well as corresponding spreadsheet instances (Cunha et al., 2011a). Figure 2.8 illustrates an example with a spreadsheet instance and a corresponding ClassSheet model embedded in a spreadsheet system. In this example, the model defines a horizontally recurring group of cells containing fields of different types. Furthermore, it also defines aggregations on the right end of the recurring groups. Based on this template, the spreadsheet software restricts the user in editing the data, e.g., by making sure that only data of the given type is entered, or that data sets or only added according to the (horizontally) expandable group.

MDSheet also enables the model-driven evolution of spreadsheets. With respect to this, Cunha et al. (2011b) propose a transformation system which map adaptions of spreadsheet models to adaptions of the corresponding spreadsheet instances. For this purpose, they define semantic and layout rules addressing both model and visual changes. Later on, they extend their approach to support bi-directional changes (Cunha et al., 2012b), i.e., changes to spreadsheet instances can be propagated to the corresponding spreadsheet model through model inference (Cunha et al., 2010, 2014a). This enables an end-user-friendly refactoring of spreadsheets and their models



	A	B	C	D	E	F	G	H	I	J	K
1	**Flights**	**PlanesKey**				**PlanesKey**					
2		N2342				N341					
3	**PilotsKey**	**Depart**	**Destination**	**Date**	**Hours**	**Depart**	**Destination**	**Date**	**Hours**		**Total Pilot Hours**
4	pl1	OPO	NAT	12/12/2010 – 14:00	07:00	LIS	AMS	16/12/2010 – 10:00	02:45	...	09:45
5	pl1	OPO	NAT	01/01/2011 – 16:00	07:00						07:00
6					⋮						
7					14:00				02:45		16:45
8	**Pilots**										
9	**ID**	**Name**	**Flight hours**								
10	pl1	John	3400								
11	pl2	Mike	330								
12	pl3	Anne	433								
13		⋮									
14											

	A	B	C	D	E	F	G
1	**Flights**	**PlanesKey**				...	
2		plane_key=**Planes**.n-number				...	
3	**PilotsKey**	**Depart**	**Destination**	**Date**	**Hours**	...	**Total Pilot Hours**
4	pilot_key=**Pilots.ID**	depart=""	destination=""	date=d	hours=0	...	total=SUM(hours)
5	⋮	⋮	⋮	⋮	⋮	...	⋮
6					total=SUM(hours)	...	total=SUM(**PlanesKey**.total)
7							
8	**Pilots**						
9	**ID**	**Name**	**Flight hours**				
10	id=""	name=""	flight_hours=0				
11	⋮	⋮	⋮				

Figure 2.8.: An example of the application of *MDSheet* showing a spreadsheet instance of a fictive airline company as well as the corresponding ClassSheet model (Cunha et al., 2012a).

(Cunha et al., 2014c). In this sense, MDSheet supports both the data-first and model-first approach to data modeling as described in Section 2.1.1.

On the top of spreadsheet models, end-users are also enabled to define model-based queries, either by a textual query interface (Cunha et al., 2013) or a visual query language (Cunha et al., 2014b). This addresses another shortcoming which was identified by our study as described in Section 2.2.4, namely shortcoming 17 in Table 2.1.

While model-based spreadsheet engineering holds lots of promises (Cunha, 2011; Beckwith et al., 2011), there are still shortcomings regarding the support for complex linked data as defined in Chapter 1. Due to the limitation that spreadsheet cells basically only support strings, numbers, and boolean values (Sestoft, 2011), multiple values as well as nested objects within a spreadsheet cell are not supported. Furthermore, modeling relations and performing calculations based on them—usually by applying the *VLOOKUP* function—is one driver to spreadsheet complexity and a major cause for errors in spreadsheets (Reschenhofer et al., 2016d; Hodnigg and Mittermeir, 2008). Consequently, although ClassSheets and the MDSheet approach provide useful means to define spreadsheet models, they are still not able to define complex linked data structures qualifying to our definition.

2.3.1.2. Spreadsheets in Multi-user Environments

Already in 1990, Nardi and Miller (1990) discussed the support of spreadsheets for the cooperative development of applications from an ethnographic perspective. They observed that a majority of spreadsheets in corporate environments are developed through collaboration of users with different levels of technical skills and domain knowledge. In this way, tech-savvy developers contribute to the spreadsheet design by implementing sophisticated code (e.g., complex formulas or macros), and providing and teaching those code artifacts to less experienced users. Bringing users with diverse technical skill sets and domain knowledge together fosters the development of spreadsheet applications which are highly tailored to the respective domain setting and thus can significantly improve the efficiency of their end-users (Spahn et al., 2008a; Wulf and Rohde, 1995).

On the top of bringing technical skills and domain knowledge together, Panko and Halverson Jr. (1994, 1997, 2001) argue that the collaborative development of spreadsheets reduces the number of errors significantly. Their studies show that groups of three collaborating users make 78% fewer errors than single users. Collaborative spreadsheet development not only refers to group development where multiple users are designing the spreadsheet simultaneously, but also to scenarios where users are building program modules of spreadsheets independently and perform a code inspection of each other's modules (Panko and Halverson Jr., 2001).

While Nardi and Miller (1990) argue that generally a majority of corporate spreadsheets are developed collaboratively, Ginige et al. (2010a) highlight that today's spreadsheet software do not provide technical capabilities and features to support collaboration adequately. For example, collaborating users share spreadsheet via email which leads to difficulties in synchronizing changes and keeping them consistent. Furthermore, the lack of access rights and a revision history are named as further issues when using spreadsheets in multi-user environments. This observation matches with our own study on spreadsheet shortcomings as described in Section 2.2.4.

Ginige et al. (2010a) also derive spreadsheet-specific design patterns based on a more general set of design patterns for cooperative interaction defined by Martin and Sommerville (2004). Those patterns include *Artifact as an Audit Trail* and *Multiple Representations of Information* based on which they derive UI-related requirements for collaborative spreadsheet solutions covering the following aspects (Ginige et al., 2010b):

Collaboration Environment Awareness A collaborative spreadsheet tool should create awareness of which users are interacting with which parts of the spreadsheet. This applies to synchronous as well as to asynchronous collaboration.

Overall Collaboration Process Analysis The spreadsheet tool should support individual end-users in exploring the personal record of their own activities in order to foster transparency regarding the current phase within a certain business process.

Operations on the Artifact The spreadsheet should be collaboratively definable and adaptable by multiple users, including the collaborative maintenance of the spreadsheet's data, collaborative design of its structure, and collaborative definition of its logic (e.g., formulas).

Addressing the second of those points, the general problem of making the design and structure of a spreadsheet understandable to end-users and particularly co-workers which are not its designer was already subject to related research in the past, e.g., by Ronen et al. (1989), Shiozawa et al. (1999), Clermont (2003), and Kankuzi and Ayalew (2008). Most recently, Hermans et al. (2011b) studied problems of transferring spreadsheets from one user to another, or of understanding and checking spreadsheets (e.g., as part of an audit) by users other than its designer. They conclude that the most important information need of users in those scenarios is about the structure of formula dependencies. For this reason, they propose data flow diagrams to make those semantic dependencies transparent to end-users.

In their work, Hermans et al. (2011b) developed three graph-based views: The first one is

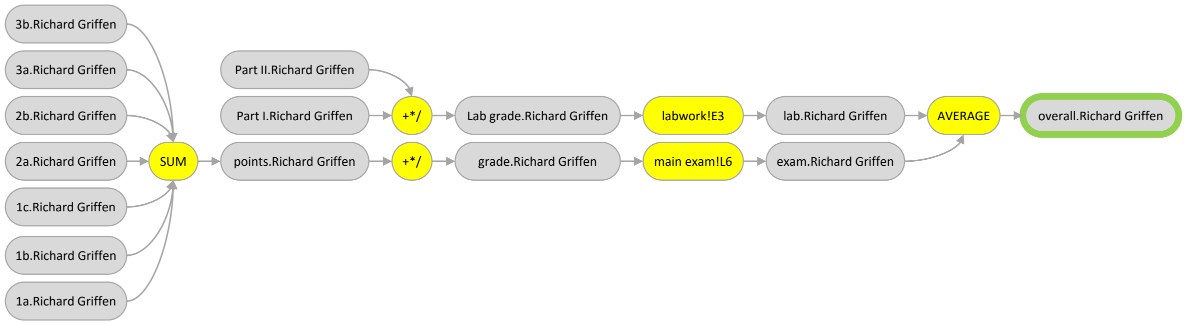


Figure 2.9.: An example of the *Formula View* visualizing the data flow between cells based on formulas for the calculation of an exam grade (Hermans et al., 2011b).

the *Global View* which illustrates worksheets (spreadsheet tabs) and data flows between them, whereas the thickness of edges indicates the number of cross-worksheet references. Analogously, the second view—the *Worksheet View*—visualizes data blocks of a single workspace as nodes, and dependencies between them as edges. The third view is the *Formula View* which visualizes the data flow on cell and formula-level, i.e., in this view the nodes are represented by single cells. Figure 2.9 shows an example for a *Formula View* based on a simple scenario for the calculation of an exam grade. In a follow-up publication, they also describe the implementation of the tool *Breviz* which automatically generates those views from a given spreadsheet application (Hermans et al., 2011a).

Hermans et al. (2011b) conclude that the different data flow views help users to understand the spreadsheet's design and to familiarize them with the concrete spreadsheet application. Those data flow views still miss the relations to the users which are interacting with different parts of the spreadsheet, and thus with the different nodes of the data flow views. However, this is one of the requirements for collaborative spreadsheet applications as defined by Ginige et al. (2010a) and as described above. For example, when clicking on a node in the data flow graph in Figure 2.9 in a multi-user setting, an end-user might want to check who provided the values for certain cells, or who defined certain formulas in the calculation chain.

In general, we can summarize that although collaborative spreadsheet development was already discussed by Nardi and Miller (1990), current spreadsheet tools still lack technical capabilities to support collaboration adequately, e.g., fine-grained access rights, and revision history (Ginige et al., 2010a). This matches with our observations as described in Section 2.2.4. On the other hand, researchers already developed approaches to solve related and partial problems, e.g., a tool to make a spreadsheet's design transparent and to facilitate the familiarization of users with this spreadsheet (Hermans et al., 2011b). However, those ideas are still not integrated to a holistic collaborative spreadsheet approach. By taking also the shortcomings regarding the support for complex linked data as discussed in the previous Section into account, we conclude that current spreadsheet software as well as related research approaches do not represent a viable option for serving as a tool supporting the collaborative EUA of evolving complex linked data structures.

2.3.2. Related Work on End-user-driven and Model-based Analytics and Visualizations

In the previous section, we observed how current spreadsheet approaches providing sophisticated EUA features relate to our endeavor of developing an approach for collaborative EUA based on evolving complex linked data. In the following section, our argumentation will start from the opposite side: Based on collaborative meta-model-based information systems which already support evolving complex linked data structures, we studied related work on end-user-driven analytics and visualizations based on those systems.

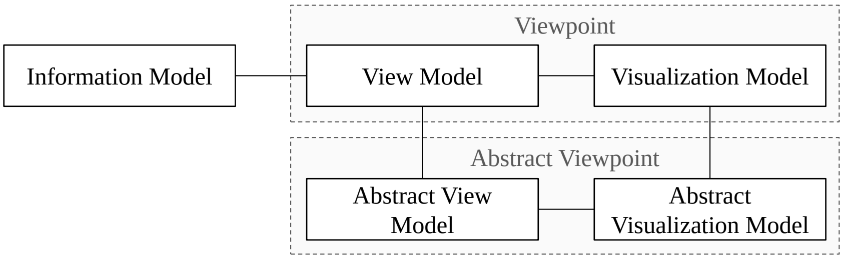


Figure 2.10.: An excerpt of the conceptual model for generating visualizations based on information models (Schaub et al., 2012; Hauder et al., 2012).

On that note, Schaub et al. (2012) developed a conceptual framework for the end-user-driven definition of interactive visualizations. They argue that the effort for end-users to configure visualizations should be minimal, i.e., in the best case the configuration should be done automatically. Thereby, the conceptual framework as depicted Figure 2.10 differentiates between the concepts *Information Model*, *Viewpoint*, and *Abstract Viewpoint*. The *Information Model* refers to the actual data model. On the other side, the *Abstract Viewpoint* represents generic and configurable visualization templates, which define their generic information demand as *Abstract View Model*. To instantiate the *Abstract Viewpoint* and to create a concrete *Viewpoint*, the end-user has to map the *Abstract View Model* to a matching part of the *Information Model*, which in turn is referred to as the *View Model*. This *View Model* represents the concrete information demand of the instantiated visualization. Hauder et al. (2012) apply this framework also for cross-organizational process visualizations, while Roth et al. (2013) propose a structural model matching algorithm enabling tools to automatically match a given *Abstract View Model* with a respective *Information Model*.

While the conceptual framework by Schaub et al. (2012) provides a user-friendly way of configuring visualizations based on arbitrary information models, it lacks means to define model-based rules and calculations, e.g., metrics quantifying specific aspects of the information model. Consequently, while the abstract view model matching framework enables end-users to create and configure qualitative visualizations, the definition of quantitative visualizations require means to enable end-users to quantify specific aspects of the information model. The EA metrics as defined by Schneider et al. (2015) and as introduced in Chapter 1 are an example for such quantifications which could not be visualized adequately by the means provided by the abstract view model matching framework (Schaub et al., 2012).

With respect to quantitative and model-based visualizations, Frank et al. (2009) develop a Domain-Specific Modeling Language (DSML) named *ScoreML* for the design of a Performance Management Information System (PMIS). In this system, users can create KPIs by defining them with an expressive quantification based on a formal calculation rule, and associate them with objects of an underlying information model. In their work, Frank et al. (2009) focus on modeling KPIs and their relationships, i.e., they do not elaborate on the formal expressions for defining calculation rules, but rather discuss how different KPIs are related to each other, and which meta-information of KPIs is relevant for certain stakeholders. For this purpose, they present the meta-model for *ScoreML*, which defines multiple reflective relationships for indicators as well as additional concepts representing an indicator's meta-information, e.g., threshold values or data sources for its computation. Furthermore, the empowerment of end-users to define visualizations based on those KPIs is also not a subject of their work.

In fact, the endeavor of the thesis at hand to enable EUA based on evolving complex linked data can be seen as a combination of the approaches of Schaub et al. (2012) and Frank et al. (2009): On the one hand, end-users should be enabled to define calculation rules representing metrics, KPIs and other analytical abstractions. The meta-information of those analytical abstractions—as exemplarily captured by the *ScoreML* meta-model (Frank et al., 2009)—is particularly relevant in a collaborative environment, since it allows users to perform a meta-analysis of existing analytical abstractions and thus to familiarize themselves with existing structures. On the other hand, in the context of this thesis we also aim for empowering end-users to define visualizations based on analytical abstractions. For this purpose, we will build on the foundations of Schaub et al. (2012), and extend the abstract concept of *Abstract View Models* in order to support the visualization of analytical abstractions.

2.3.3. Related Work on Self-Service BI and Collaborative BI

In the previous sections we discussed related approaches to EUA for complex linked data which (at least partially) serve as foundation for the development of our own approach. Conversely, in this section we will discuss related parallel approaches in the field of end-user oriented (Self-Service) and collaborative Business Intelligence (BI) (Reschenhofer and Matthes, 2016a).

Sell et al. (2005) argue that analytical tools usually suffer from a lack of flexibility regarding their exploratory capabilities, lack of support for defining business logic, and missing capabilities to define stakeholder-specific representations of information. Those observations match the findings of our study on spreadsheet shortcomings as described in Section 2.2.4. To tackle those issues, Sell et al. (2005) propose a web-based architecture for analytical information systems based on user-definable ontologies (cf. Figure 2.11). In general, ontologies represent an "explicit specification of a conceptualization" (Gruber, 1993). They describe the semantics of information structures and make their contents explicit (Wache et al., 2001). The layered architecture defined by Sell et al. (2005) is built on generic BI and specific domain ontologies which can be analyzed by user-definable functional modules. On the top of the abstract architecture and by using the capabilities of the functional modules through respective web services, different clients can represent and process the system's data in a stakeholder-specific way. While the web-based and service-based nature of this architecture already enables support for multiple and different users

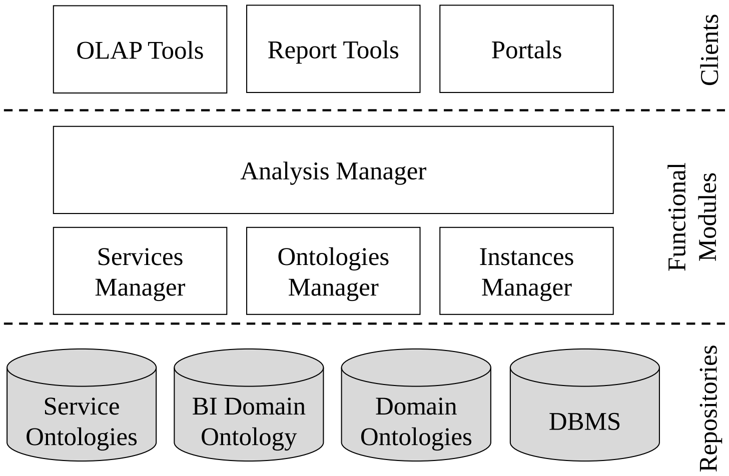


Figure 2.11.: The illustration of the analytical information system architecture modules as defined by Sell et al. (2005).

and clients, Sell et al. (2005) do not discuss collaborative analysis and consequences arising from it. This seems to be an issue in general, as Kaufmann and Chamoni (2014) found out that collaboration-related challenges are hardly addressed in scientific literature in the context of EUA.

Similarly, Spahn et al. (2008b) argue that traditional BI systems are too complex to be adapted by end-users, and that those end-users are not able to tailor the BI systems to changing domain-specific needs. In addition to the work by Sell et al. (2005), they propose a layered architecture consisting of ontologies, query transformations, and stakeholder-specific applications (cf. Figure 2.12). In the context of the work by Spahn et al. (2008b), ontologies define a holistic and domain-specific data structure which potentially integrates data from multiple sources. Based on this, end-users can define analytical abstractions, which in turn can be consumed by potentially multiple clients or applications. However, similar to Sell et al. (2005), Spahn et al. (2008b) discuss neither how the definition of ontologies and queries can be done collaboratively, nor which consequences would arise from this kind of collaboration.

In 2012, Mertens and Krahn (2012) conducted a study on shortcomings of analytical information systems, which enable IT experts to analyze huge data sets. They derive requirements to make analytical information systems usable by business users with a limited technical expertise, and develop a layered conceptual architecture addressing those requirements. As illustrated in Figure 2.13 and similar to the architecture by Spahn et al. (2008b) (cf. 2.12), the bottom layer represents heterogeneous data sources and services. In the back-end layer of the architecture, those data sources are integrated and structured through a semantic meta-data component, which in turn represents the basis for end-user-driven data analysis and data annotation. The front-end layer provides respective UI components to enable end-users to consume the analysis services provided by the analytical information system. But again, collaboration and induced challenges are not subject of the work by Mertens and Krahn (2012).

Studying the works by Sell et al. (2005), Spahn et al. (2008b), and Mertens and Krahn (2012)

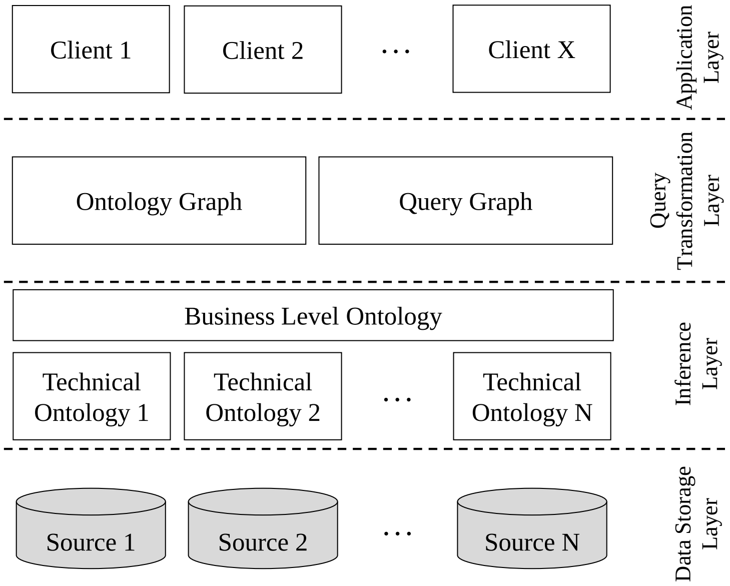


Figure 2.12.: The illustration of the system architecture layers as defined by Spahn et al. (2008b).

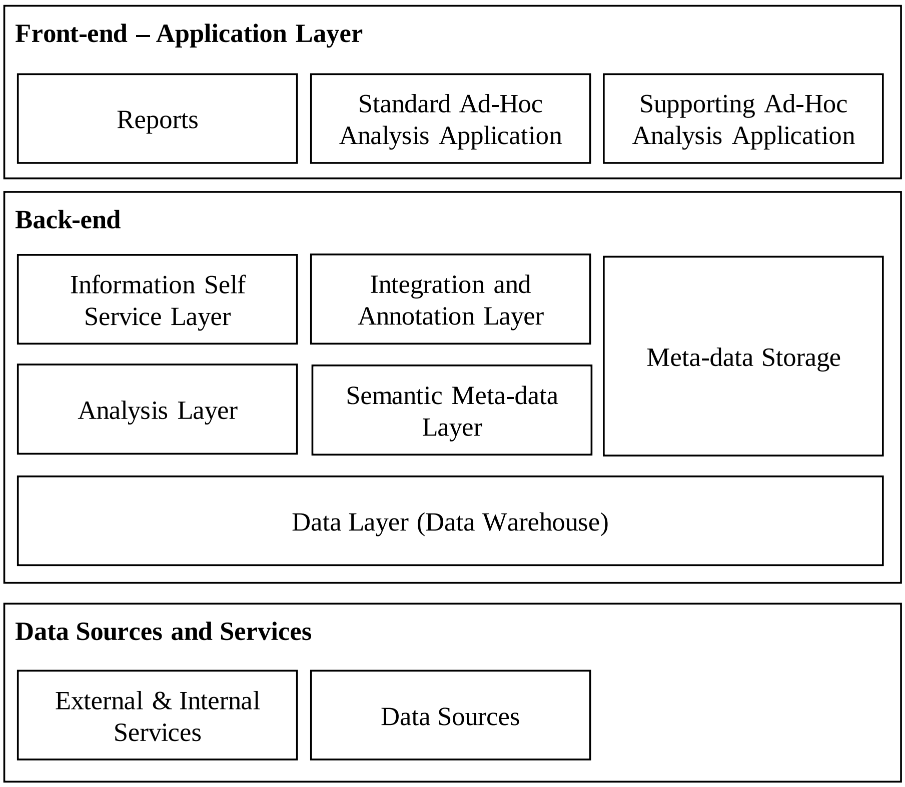

Front-end – Application Layer
Reports
Standard Ad-Hoc Analysis Application
Supporting Ad-Hoc Analysis Application
Back-end
Information Self Service Layer
Integration and Annotation Layer
Meta-data Storage
Analysis Layer
Semantic Meta-data Layer
Data Layer (Data Warehouse)
Data Sources and Services
External & Internal Services
Data Sources

Figure 2.13.: The illustration of the architecture of an analytical information system with a semantic meta-data layer as defined by Mertens and Krahn (2012).

confirms the observations of Kaufmann and Chamoni (2014)—namely that collaboration-related challenges are hardly discussed in scientific literature, particularly with respect to conceptual architectures of BI systems. Nevertheless, these works still provide valuable input for the derivation of requirements and conceptual design of a tool for collaborative EUA on evolving complex linked data. All of those architectures share the same abstract conceptual layers: A layer to empower end-users to define high-level and holistic data structures, one to allow them to define analytical abstractions based on those data structures (e.g., queries), and one to enable users to define user-specific data representations and applications. We will elaborate on those layers in detail in Chapters 3 and 4.

One of the first approaches towards collaborative BI was developed by Dayal et al. (2008) which derive a set of respective requirements. For example, one of the derived requirements is the need for means to query, report, and analyze both historical and real-time data. Again, they propose a conceptual architecture including layers for data integration, data and process modeling, and analytics, based on which different dashboards visualize the data and analytical abstractions of the underlying BI platform. However, the focus of the work by Dayal et al. (2008) is the design of a virtual three-dimensional room to allow real-time collaboration in the context of BI. Just like in the aforementioned works by Sell et al. (2005), Spahn et al. (2008b), and Mertens and Krahn (2012), collaboration-induced challenges are hardly discussed.

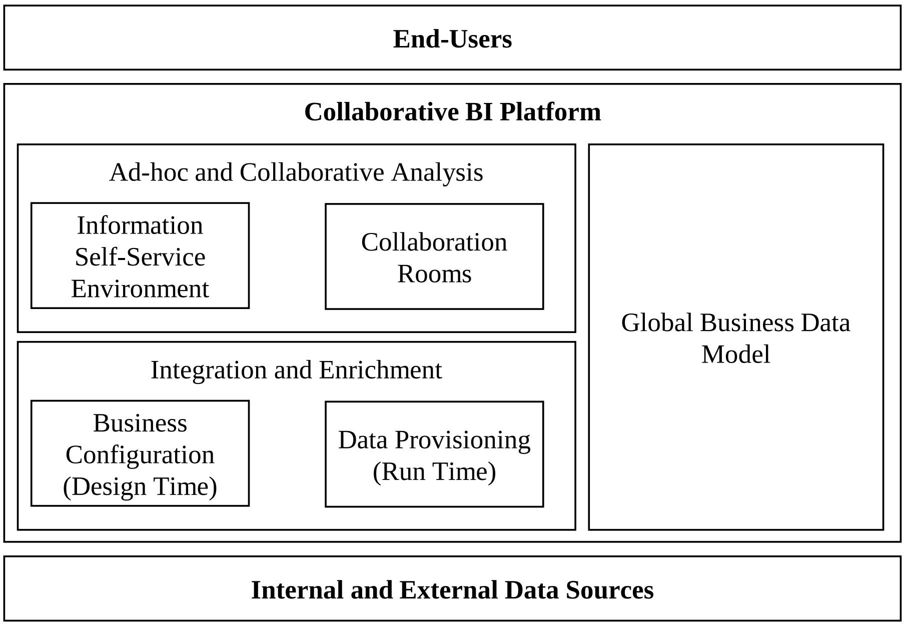

End-Users
Collaborative BI Platform
Ad-hoc and Collaborative Analysis
Information Self-Service Environment
Collaboration Rooms
Integration and Enrichment
Business Configuration (Design Time)
Data Provisioning (Run Time)
Global Business Data Model
Internal and External Data Sources

Figure 2.14.: The illustration of the architecture for a collaborative BI platform as proposed by Berthold et al. (2010).

Berthold et al. (2010) envision a platform for end-user-oriented BI in a collaborative environment. The goal of this platform is to empower end-users to configure, structure, and analyze data in a self-directed way and without technical skills. It implements a life cycle for BI consisting of iterative phases for business configuration (data and process modeling), information self-service, collaborative analysis, and collaborative decision making. The presented architecture consists of a data integration and enrichment layer as well as an analysis layer including components enabling information self-service and collaboration (cf. 2.14). However, in their paper, Berthold et al. (2010) only provide a high-level architecture of a collaborative BI platform, and elaborate neither on concrete concepts to enable collaboration on all layers of their architecture, nor on challenges arising from it.

On a related note, Rizzi (2012) interprets collaborative BI as the cooperation of multiple companies as part of a joint endeavor to extend and improve their decision-making processes. For this purpose, those companies have to connect their individual and autonomous BI systems which can lead to novel insights. However, the thesis at hand addresses intra-organizational collaborative EUA rather than inter-organizational decision making.

As a state-of-the-art analysis, Walter (2015) observed Self-Service BI tools which are typically applied in practice for empowering end-users to perform basic data analytics tasks. Based on Gartner's *Magic Quadrant for Business Intelligence and Analytics Platforms* (Gartner Inc., 2015), she selected six tools (*Tableau Desktop*[1], Qlik Sense[2], Microsoft Excel[3], SAS Visual Analytics[4], SAP Lumira[5], and IBM Watson Analytics[6]). These tools were analyzed along seven different dimensions representing the steps of a typical (Self-Service) BI pipeline, namely the generation, integration, storage, access, assessment, sharing, and usage of data. One conclusion of this state-of-the-art study is that current Self-Service BI tools suffer from a lack of support for evolving complex linked data. Particularly the evolution of the structure of the data to be analyzed leads to problems since the data structuring and analysis facilities of current Self-Service BI tools are isolated from each other and thus cannot react adequately on changes of another part of the BI pipeline.

To summarize related work on Self-Service and collaborative BI: Particularly the works by Sell et al. (2005), Spahn et al. (2008b), Mertens and Krahn (2012), and Berthold et al. (2010) serve as a conceptual foundation for the requirements and conceptual design for a tool empowering end-users to collaboratively analyze evolving complex linked data as described in the following Chapters 3 and 4. In this sense, we particularly address shortcomings of today's spreadsheets (cf. Section 2.2.4) and other self-service BI tools.

2.4. Summary of Foundations and Related Work

In the previous sections, we introduced the foundations for the development of an approach empowering end-users to collaboratively analyze evolving complex linked data.

In Section 2.1, we discussed adaptive and meta-model-based information systems in general, and elaborates on the Hybrid Wiki approach as one concrete representative of this class of information

[1] `http://www.tableau.com/desktop`, last accessed on: 04.10.2016

[2] `http://www.qlik.com/us/products/qlik-sense/desktop`, last accessed on: 04.10.2016

[3] `https://products.office.com/en/excel`, last accessed on: 04.10.2016

[4] `http://www.sas.com/en_us/software/business-intelligence.html`, last accessed on: 04.10.2016

[5] `http://saplumira.com`, last accessed on: 04.10.2016

[6] `https://www.ibm.com/analytics/watson-analytics`, last accessed on: 04.10.2016

systems in particular. Hybrid Wiki concepts will serve as a conceptual (and technical) foundation for the development of our own approach. As described later on in Chapter 4, we will extend those concepts by means for the end-user-driven analysis of the Hybrid Wiki system's data structures.

For this purpose, we studied EUA and its tool-support, and summarize the results in Section 2.2. Thereby, we particularly observed the success factors and characterizing features of EUA software (cf. Section 2.2.1) with the objective of adopting them to our approach. Furthermore, we conducted a state-of-the-art analysis of spreadsheets in order to reveal how they are applied in practice, which data structures they usually contain, and from which shortcomings they are suffering when applied as information systems. The findings of this analysis represent drivers for the conceptual and technical design of our own EUA approach.

While Sections 2.1 and 2.2 summarize research topics serving as foundation for the development of our solution, Section 2.3 discusses related approaches which also aim for bridging the gap between adaptive information systems and EUA. Section 2.3.1 discusses research about approaches to model-based and collaborative spreadsheets. In this sense, spreadsheets as the de facto standard for EUA tools are extended by data modeling and collaboration capabilities in order to move them towards support for complex linked data structures in a collaborative environment. In contrast, Section 2.3.2 elaborates on the opposite approach, i.e., moving collaborative meta-model-based information systems towards support for EUA. Finally, Section 2.3.3 provides an overview over approaches which are comparable to the approach of this thesis. Thereby, we outline the highlights and conceptual architectures of collaborative BI platforms and briefly describe the differences to our own approach.

CHAPTER 3

Identification of Requirements

In this chapter, we elaborate on how we envision the conceptual framework for empowering end-users to collaboratively analyze evolving complex linked data. Based on this conceptual framework and findings of related work, we systematically derive and describe requirements of five different categories for respective EUA tool support.

In order to enable a plausible derivation of conceptual requirements for EUA based on evolving complex linked data, we define a framework capturing basic concepts and roles which are involved in this context. This conceptual framework is based on related work in the field of end-user-driven data visualizations, visual information analysis, end-user-driven model analysis, and End-User Analytics (EUA).

3.1. A Conceptual Framework for Collaborative EUA

EUA tools, e.g., spreadsheets, are used to structure, analyze, and visualize data in a self-directed and ad-hoc manner (Pemberton and Robson, 2000). In this context, Isenberg et al. (2008) describe an information analysis process consisting of eight steps, e.g., scanning through available data, and understanding and selecting proper visualizations. To formalize this kind of information analysis and visualization process, Chi and Riedl (1998) developed the *Information Visualization Data State Reference Model* as illustrated in Figure 3.1 (in the remainder of the thesis we refer to this model as "data state model"). The data state model describes four states through which initially raw data is (iteratively) transformed into interactive views. Those four states describe the following artifacts:

Data/Values represent the raw data and are the starting point of an information flow.

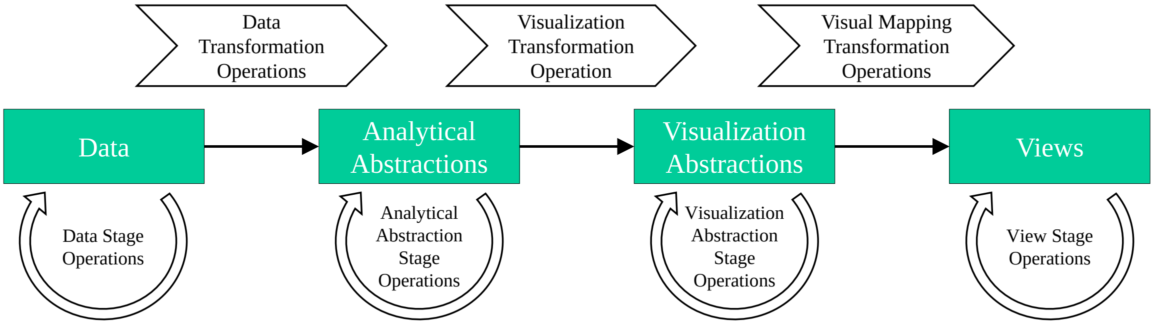

Data Transformation Operations
Visualization Transformation Operation
Visual Mapping Transformation Operations
Data
Analytical Abstractions
Visualization Abstractions
Views
Data Stage Operations
Analytical Abstraction Stage Operations
Visualization Abstraction Stage Operations
View Stage Operations

Figure 3.1.: The *Information Visualization Data State Reference Model* as defined by Chi and Riedl (1998).

Analytical Abstractions describe intermediary states of analytical data transformations and meta-information about the data. For example, in spreadsheets, results of formulas can be interpreted as analytical abstractions.

Visualization Abstractions are abstract representations of visualizations. From a model-based perspective, they compare to *Abstract Viewpoints* as described in Section 2.3.2 and as defined by Schaub et al. (2012) and Hauder et al. (2012).

Views represent the final picture as shown to users and the end point of an information flow.

Between the artifacts of the stages there are three kinds of cross-stage transformations (Chi and Riedl, 1998). *Data Transformation Operations* transform data into analytical abstractions. *Visualization Transformation Operations* bind the analytical abstractions to abstract visualization models. And *Visual Mapping Transformation Operations* render the abstract visualization models to graphical views. In addition to those cross-stage transformations, the data state model also defines operations between two artifacts within the same stage, e.g., *Analytical Stage Operations* as transformation from one analytical abstraction to another one.

As shown by Chi (2000), the data state model is applicable to a diversity of information visualization techniques. In the present thesis, we use it together with the conceptual model for model-based visualizations (cf. Section 2.3.2) as a foundation for a conceptual framework for our tool enabling collaborative model-based EUA on evolving complex linked data.

Our understanding of collaborative model-based EUA is captured by the conceptual framework in Figure 3.2. It is defined by the *Data Models*, *Analytical Abstractions*, *View Templates*, and *Views* layers. Those layers consist of interrelated conceptual artifacts which are inspired by related work (cf. Section 2.3.3) and described in the following.

In a model-based environment, data is structured through respective data models. In the meta-model-based Hybrid Wiki approach (cf. Section 2.1.2), data models are defined by entity types, attributes, and relations. Chuah and Roth (1996) argue that adding derived attributes is also a basic interaction in the context of information analysis. There are multiple roles of users involved in the definition and management of data models: On the one hand, *Data Modeling Experts* as trained experts understand the meta-model and modeling facilities offered by the

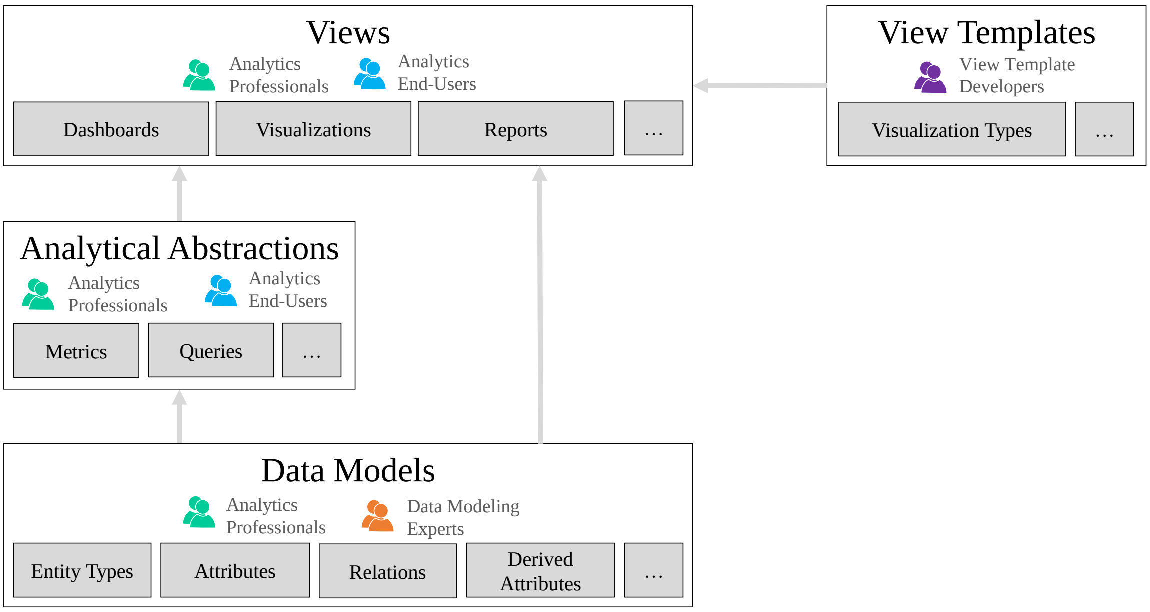


Figure 3.2.: Conceptual framework for collaborative EUA on complex linked data, inspired by Chi and Riedl (1998), and Schaub et al. (2012), and related work summarized in Section 2.3.3.

respective tool and are able to define and adapt complex domain-specific data structures and constraints (Roth, 2014). In this sense, they represent data custodians and stewards (Khatri and Brown, 2010; Roth, 2014) as well as technical domain experts (Rehm et al., 2014). On the other hand, *Analytics Professionals* are involved in the definition of derived attributes which describe attributes whose values are calculated automatically based on a given calculation rule. Therefore, derived attributes represent analytical abstractions which are embedded into data models. With regard to the data state model, changes to the data model qualify as *Data Stage Operations*.

Based on the data model, both *Analytics Professionals* and *Analytics End-Users* (Tamm et al., 2013) define analytical abstractions. Those analytical abstractions describe metrics, queries, or simple calculations, to which both the underlying data model and other analytical abstractions can serve as input. Therefore, and with regard to the data state model in Figure 3.1, our conceptual framework allows for both *Data Transformation Operations* and *Analytical Abstraction Stage Operations*.

On the top level of the framework depicted in Figure 3.2, *Analytics Professionals* as well as *Analytics End-Users* can define different kinds of views, e.g., dashboards, visualizations, or reports. To define views, users have to select from a set of view templates which compare to *Abstract Viewpoints* as described in Section 2.3.2. View templates are implemented by *View Template Developers* which are professional programmers with a respective skill set and technology knowledge, e.g., about visualization libraries. Therefore, *View Template Developers* represent Software engineers and designers as described by Roth et al. (2013). They are also responsible for the implementation of *Visual Mapping Transformation Operations* as well as operations for both the visualization abstraction and view stage as described by Chi and Riedl (1998).

View templates have to define a *data input interface* specifying which kind of data they are able to process and to present, e.g., in the form of an abstract view model (Schaub et al., 2012). When selecting a view template during the creation of a view, data input interfaces determine which data or analytical abstractions can serve as input for the view, or how they have to be transformed in order to serve as input. In this sense, views are created by instantiating view templates and bind them to analytical abstractions which conform to the view template's data input interface. This kind of conceptual framework compares to the Enterprise Mashup (EM) environment as defined by Pahlke et al. (2010): In this environment, software vendors can offer different kinds of information services which can serve as input for end-user-definable EM applications. In terms of the data state model by Chi and Riedl (1998), the binding of analytical abstractions to view templates represents *Visualization Transformation Operations*.

In summary, collaborative model-based EUA involves different user-definable artifacts on different layers, e.g., data model elements, analytical abstractions, or views. Related work suggests that there are different types of user roles primarily interacting with artifacts of specific layers, e.g., *Data Modeling Experts* define the data model (Roth, 2014; Khatri and Brown, 2010; Rehm et al., 2014), *View Template Developers* technically implement abstract views, and *Analytics Professionals* as well as *Analytics End-Users* manage analytical abstractions and concrete views. As a whole, the artifacts of all layers form an analysis model capturing the artifacts themselves as well as their dependencies to each other.

For example, we can apply the conceptual framework for collaborative EUA to the domain of EAM (Roth, 2014): An EA modeling expert would take the role of the *Data Modeling Expert* and define an organization-specific EA model. Based on this model, an enterprise architect as *Analytics Professional* can define EA metrics and respective visualizations. Other EA stakeholders and decision makers would take the role of *Analytics End-Users* and would be create new stakeholder-specific EA visualizations, while they could choose from a predefined set of specific EA view templates (Roth et al., 2014).

3.2. Identification of Requirements

The conceptual framework for collaborative and model-based EUA as depicted and described in the previous section defines which user roles are interacting with which artifacts, and how those artifacts are generally related to each other. Based on this understanding, we can now derive abstract requirements for tool-support for collaborative model-based EUA. In general, we apply general collaboration features to the artifacts of all layers of the conceptual framework. de Hertogh et al. (2011) define them as the basic Web 2.0 requirements, which we adopt to EUA as follows (Reschenhofer and Matthes, 2016a):

Reusability of Artifacts In a collaborative EUA tool, the artifacts of all layers should be reusable and combinable by oneself and by other users.

Adaptability of Artifacts End-users should be able to adapt the artifacts of each layer at any time, provided they have respective access rights.

Collaborative Content Creation and Modification A collaborative EUA tool facilitates the collaborative creation and management of data, analytical abstractions, and views.

No Predefined Structure of Content A collaborative EUA tool should not impose a predefined structure on its content, i.e., it should support knowledge workers to dynamically enrich content with additional, not yet defined structure.

Responsive and Personalized UI In this context, responsiveness of a UI refers to immediate feedback to users when performing certain actions, while a personalized UI takes into account the user's access rights, roles, preferences, etc.

Gathering of Collective Intelligence A collaborative EUA tool should involve knowledge workers with different domain knowledge in content creation and management.

Together with the findings from related work (cf. Section 2.3) and the identified shortcomings of current EUA software (cf. Section 2.2.4), those generic Web 2.0 requirements build a foundation for the derivation of requirements for collaborative EUA tool-support based on evolving complex linked data. We categorize the requirements based on the layers of our conceptual framework, namely data model, analytical abstractions, and views with view templates (cf. Sections 3.2.1, 3.2.2, and 3.2.3 respectively). Section 3.2.4 describes requirements related to the meta-analysis of artifacts and their (potentially cross-layer) relations.

3.2.1. Data Model Requirements

As motivated in Chapter 1, we aim for tool-support for analyzing evolving complex linked data. In the context of this thesis, complex linked data is defined as entities which potentially have arbitrarily nested attributes, attributes with multiple values, and multiple relations to other data entities. Examples for this kind of data structures are EA models (Buckl et al., 2010). The lack of support for complex linked data was identified as a major shortcoming of current EUA tools, particularly spreadsheets (cf. Section 2.2.4). A natural consequence for the data model layer of a supporting EUA tool is that it has to support the user-driven design of complex linked data models:

> Req 1: **Modeling of Complex Linked Data**
>
> A solution must enable end-users to model complex linked data. This means that the solution's meta-model has to define concepts for describing entities with nested and multi-valued attributes and relations.

Collaborative creation and gathering collective intelligence are both defined as basic Web 2.0 requirements (de Hertogh et al., 2011). In this sense, collaborative modeling of complex linked data should be performed by cooperating domain experts, data modeling experts, and analytics professionals. Those users provide diverse skills and expertises which can be utilized in a collaborative data modeling process. Again, as described in Section 2.2.4, current EUA tool-support does not provide adequate collaboration features.

Req 2: **Collaborative Creation of Data Model**

Multiple users with different roles and different kinds of expertise should be enabled to contribute to the collaborative design and creation of the data model.

In a collaborative environment, users contribute to the design of the data model at different stages of the design process (de Hertogh et al., 2011). For example, domain experts might identify changes of the domain environment which imply the need for adaptions of the data model, while analytics professionals can augment the data model with analytical abstractions, e.g., derived attributes (Chuah and Roth, 1996). Therefore, adaptability and flexibility are crucial properties of a data model in a collaborative environment (Mertens and Krahn, 2012), and a fundamental prerequisite for enabling evolving data structures.

Req 3: **Adaptability of Data Model**

Data models have to be adaptable by end-users at run-time. In this sense, end-users should be able to iteratively define the data model in an initial design phase, and subsequently refine and tailor it in accordance to potential environmental changes.

The first step in the information analysis process is to scan through the data in order to gain some understanding of the available data (Isenberg et al., 2008). In the context of model-based and collaborative EUA based on evolving complex linked data structures, this means that users should be enabled to explore the current state of the evolving data model at any time, but particularly during the definition of model-based analytical abstractions. The evolving nature of the data model implies that the exploration of the data model is a continuous task for end-users.

Req 4: **Explorability of Data Model**

A solution has to provide means and views enabling end-users to explore and browse through the data model at any time, but particularly during the formulation of queries.

Protecting certain information from unwanted access is an indispensable requirement for collaborative enterprise applications (Shen and Dewan, 1992). Addressing this, the purpose of access control rights is to specify which users or user roles (*subject*) are allowed to interact with a certain artifact (*object*) in a certain way (*right*). With respect to the data model layer, effective collaborative data modeling requires respective access control to direct and govern the continuous development of the data model. Furthermore, also the data itself has to be protected by an authorization mechanism to ensure that users only see information according to their individual access rights (Neubert, 2012). This is also true for queries, i.e., queries only retrieve data to which the executing user has at least read access to.

Req 5: **Access Rights for Data and Data Model**

A solution has to provide a mechanism enforcing access rights for each user which ensures a personalized view only showing information and model elements for which the user is authorized.

3.2.2. Analytical Abstraction Requirements

After deriving requirements on the data model layer, we continue with the identification of requirements related to analytical abstractions. Analytical abstractions represent any form of meta-data or derived data. The former refers to information about the data (e.g., who is the creator of a data object), while the latter one refers to information objects which are automatically computed based on calculation rules. If those calculation rules reference data model elements (e.g., *count of all objects of a certain type*), we refer to them as queries.

As described in Section 3.2.1, requirement Req 1 defines that a solution for EUA based on complex linked data has to support the creation and management of complex linked data. Consequently, the analytical abstractions layer of such a solution has to provide means to empower end-users to define queries supporting those complex linked data structures.

Req 6: **End-User-Driven Querying of Complex Linked Data**

A solution has to support the end-user-driven definition of queries of complex linked data as defined by requirement Req 1. This explicitly includes the provision of UI means to facilitate the query formulation by end-users.

In Section 1.1, we described different exemplary queries which we experienced in previous research projects, e.g., a EA complexity metrics using the Shannon entropy (Schuetz et al., 2013), or a computation rule for deriving the status of a project based on end-user-definable criteria (Matheis, 2013). Those examples show that—in addition to support for complex linked data—different domains imply different requirements to the expressiveness of a respective query language. In order to adapt the query language's expressiveness to domain-specific requirements, the query language has to be extendable by custom and potentially complex functions. Indeed, according to Sell et al. (2005), a lack of flexibility and extension is one of the most critical issues of traditional analytical tools.

Req 7: **Extensibility of the Query Language's Expressiveness**

The query language of a solution has to be extendable in order to enhance its expressiveness and to implement complex calculation rules from different domains, e.g., new complexity metrics in EAM.

Analogous to requirement Req 2, we adopt collaborative creation and gathering collective intelligence as two basic Web 2.0 requirements (de Hertogh et al., 2011) to the analytical abstractions layer. This means that multiple users with different roles should be able to contribute to the definition of analytical abstractions. For example, multiple EA stakeholders should be enabled to contribute to the definition of an EA metric in order to harness each individual's knowledge and expertise, e.g., by enabling the definition of reusable functional components which can be shared with others (Chambers and Scaffidi, 2010; Grossman and Burd, 2015). In this regard, current EUA tool-support suffers from shortcomings related to collaboration and modularity (cf. Section 2.2.4).

Req 8: **Collaborative Definition of Analytical Abstractions**

Multiple users with different roles and kinds of expertise should be enabled to contribute to the collaborative design and creation of analytical abstractions, e.g., by creating, sharing, and composing reusable functional components.

Environmental changes (e.g., new or changing business requirements) might cause the need for adaptions of existing analytical abstractions. For example, refining an enterprise's strategy for standardizing its application landscape (Schneider, 2015) might imply an adaption of an existing EA metric measuring database diversity (cf. Section 1.1) and to capture additional aspects of the EA model. As a consequence, the adaptability of the definitions of analytical abstractions is essential to tailor them to a changing environment.

Req 9: **Adaptability of Analytical Abstraction Definitions**

The definitions of analytical abstractions have to be adaptable at run-time.

In addition to adaptability, adaptiveness is another essential feature of analytical abstraction definitions of the desired solution. While adaptability describes the capability of being manually editable at run-time by end-users, the adaptiveness of analytical abstraction definitions refers to their automated adaption on changes of artifacts they are based on, e.g., data model elements or other analytical abstractions. In the light of requirement Req 3 which implies the need for adaptable data models, adaptiveness ensures the consistency of analytical abstractions which are affected by data model changes.

Req 10: **Adaptiveness of Analytical Abstraction Definitions**

Analytical abstraction definitions have to be adaptive, i.e., changes of the underlying data model elements should automatically trigger adaptions of affected analytical abstractions and thus ensure their consistency.

In line with requirement Req 4 which describes the explorability of the data model, end-users aiming for the definition of analytical abstractions should be enabled to explore already existing and reusable analytical abstractions. In order to helping users to discover and find an analytical abstraction, the desired solution has to manage respective meta-information (e.g., a textual description) or to derive it automatically (e.g., the last modification date).

Req 11: **Explorability of Analytical Abstractions**

A solution has to provide means and views enabling end-users to explore and browse through existing analytical abstractions at any time, but particularly during the formulation of queries.

With requirement Req 5 we argue for an authorization mechanism for the data model and the underlying data. Analogously, analytical abstractions representing derived data should also be subject to access control (Shen and Dewan, 1992). Consequently, the access control mechanism as described by requirement Req 5 has to capture also analytical abstractions, i.e., access rights should control which users are allowed to interact with which analytical abstractions in which way.

Req 12: **Access Rights for Analytical Abstractions**

A solution has to provide a mechanism enforcing access rights for each user which ensures that they can only read analytical abstractions or adapt their definitions if they are authorized accordingly.

3.2.3. View and View Template Requirements

In the following section, we summarize the requirements for our conceptual framework's view and view template layers (cf. Figure 3.2). The former describes concrete views visualizing concrete data and analytical abstractions, while the latter refers to abstract viewpoints (Schaub et al., 2012; Hauder et al., 2012) defined by an abstract data input interface through which the actual visualization is instantiated with data (cf. Section 3.1).

As illustrated in Figure 3.2, visualization templates are created by view template developers including not only web developers and web designers, but also domain experts which support developers and designers in implementing domain-specific visualizations (Roth et al., 2013). Therefore, multiple users with different roles and kinds of expertise contribute to the development of view templates.

REQ 13: **Collaborative Creation of View Templates**

Multiple users with different roles and kinds of expertise should be enabled to contribute to the collaborative design and creation of view templates.

Once view templates are created, end-users can instantiate them in order to define concrete views. However, similar to the data model and analytical abstractions, view templates can be affected by environmental changes (e.g., new business requirements or the change of corporate design guidelines) potentially causing adaptions of the implementation or design of view templates.

REQ 14: **Adaptability of View Templates**

View templates should be adaptable at run-time, which in turn should imply the automated adaption of instantiated views based on the changing templates.

Adapting an existing view template would imply the automated adaption of already instantiated views. However, this is not always an intended behavior, e.g., when refining a view template based on domain-specific requirements. In this cases, developers still would like to reuse (parts of) the already existing view template without adapting it. In this sense, this refers to the usability requirement as defined by de Hertogh et al. (2011).

REQ 15: **Reusability of View Templates**

A solution has to enable view template developers to reuse existing view templates or parts of them.

The consequence of the adaption of view templates is the automated update of all views which are based on this template. In order to avoid the adaption of view templates by unauthorized users, a solution has to provide an access control mechanism which takes care that only users with respective access rights are allowed to modify existing view templates.

REQ 16: **Access Rights for View Templates**

A solution has to provide a mechanism enforcing access rights for users which ensures that they can only modify view templates if they are authorized accordingly.

Views are created at run-time by analytics professionals or analytics end-users. Similar to requirement Req 14, those users should be enabled to adapt their views in case of corresponding environmental changes. In this way, a solution provides means which allow to keep the actual views in line with the domain-specific needs and requirements of the users. Again, this requirement is in line with the basic Web 2.0 requirements (de Hertogh et al., 2011).

Req 17: **Adaptability of Views**

Views should be adaptable at run-time in order to allow end-users to align their views to environmental changes.

Another basic Web 2.0 requirement is reusability (de Hertogh et al., 2011), which again can be adopted to views. Consequently, a solution has to provide a mechanism to reuse a view or a part of it and to tailor it to personal needs. Furthermore, an already configured view (or part of it) might be of interest for multiple stakeholders, wherefore they would like to include it in their personal views. This means that a solution has to provide means to reuse and share views and parts of them.

Req 18: **Reusability of Views**

A solution has to enable end-users to reuse existing views or parts of them.

As stated by Shen and Dewan (1992), protecting information from unwanted access is essential for collaborative applications. This is particularly true for views which are constituting visual representations of information. In this sense, the desired solution should provide access control for views and control which users are allowed to see or modify them.

Req 19: **Access Rights for Views**

A solution has to provide a mechanism enforcing access rights for users which ensures that they can only see or adapt views if they are authorized accordingly.

3.2.4. Meta-Analysis Requirements

Sections 3.2.1, 3.2.2, and 3.2.3 described requirements on individual levels of the conceptual framework in Figure 3.2. In contrast, the following section elaborates on the derivation of cross-layer requirements related to the holistic analysis model. As described in Section 3.1, the analysis model represents an abstract graph which nodes and edges are represented by the artifacts of the conceptual framework and their relations to each other. In this sense, exploring and analyzing the analysis model and its graph representation is referred to as *meta-analysis*.

One important objective in a collaboration environment is to create awareness of important aspects and activities within the system which might be relevant for a user (Martin and Sommerville, 2004). Indeed, identifying the right person to interact and coordinate with is one of the most challenging tasks in collaborative systems (Cataldo et al., 2006). To create awareness in a collaborative EUA tool, the analysis model serves as foundation for a holistic perspective on data model elements, analytical abstractions, views, and their dependencies. By providing an explorable view of the analysis model to end-users, they are enabled to identify artifacts of interest and to inspect their meta-information, e.g., the creator or owner of the artifact, or the last modification date. However, this presupposes that a collaborative EUA solution supports meta-attributes for artifacts of different layers. Those meta attributes can be maintained either manually through end-users, e.g., owner (Khatri and Brown, 2010), or automatically by the tool, e.g., revision history (Ginige et al., 2010a)).

Req 20: **Meta Attributes of Artifacts**

A solution has to support meta attributes for the artifacts of all system layers, i.e., data model elements, analytical abstractions, views, and view templates. This involves manually as well as automatically maintained meta attributes, and meta attributes of different types, e.g., dates, users, etc.

Meta attributes represent properties of the nodes of the analysis model's graph representation. The edges are defined by meta relations between them. Again, those meta relations can be specified manually by end-users of the system, e.g., "similar to" relationships between analytical abstractions representing metrics (Frank et al., 2009), or determined automatically by the system, e.g., dependencies between queries and model elements they are referring to.

Req 21: **Meta Relations between Artifacts**

A solution has to support meta relations between artifacts of all system layers, i.e., data model elements, analytical abstractions, views, and view templates. This involves manually as well as automatically maintained relations, and particularly relations across different layers.

By supporting both meta attributes and meta relations, a tool can automatically generate a graph representing the analysis model. In order to provide value for end-users, the tool has to provide a respective graph view making it explorable and analyzable by users. Thereby, users should be supported in exploring particularly the meta attributes and meta relations of artifacts. This might be helpful for the coordination with co-workers and for performing different kinds of meta-analysis, e.g., impact analysis or stakeholder analysis (Reschenhofer et al., 2016b).

Req 22: **Explorability of Analysis Model**

A solution should provide views and means for making the analysis model explorable and analyzable by end-users in order to support them in performing meta-analysis tasks.

Above mentioned requirements as well as the characterizing features of EUA software describe the key properties a tool supporting the collaborative EUA of evolving complex linked data has to fulfill. They are the main drivers of the conceptual and UI design of a respective tool.

In Chapter 7, we will recapitulate the contribution of this thesis in general, and how those requirements are addressed and met by the proposed conceptual design and prototypical implementation in particular.

CHAPTER 4

Conceptual Design

Based on the conceptual framework and requirements described in Chapter 3 as well as related work summarized in Chapter 2, the following sections describe the conceptual design of a software solution enabling end-users to collaboratively analyze evolving complex linked data. Thereby, we discuss and argue about design decisions on a technology-independent abstraction level.

4.1. Hybrid Wikis as Data Model Layer

The conceptual foundation for the data model layer of the desired collaborative EUA tool will be formed by the Hybrid Wiki system. As already described in Section 2.1.2, this system provides conceptual features which enable the iterative and collaborative design of data models as well as further collaboration capabilities, e.g., access rights and version history. In this sense, the Hybrid Wiki system already supports requirements Req 2, Req 3, and Req 5.

However, requirement Req 1 describing the need for supporting complex linked data (cf. definition in Chapter 1) is only partially fulfilled by the Hybrid Wiki system: Although it allows the definition of entity types with multi-valued attributes and relations, it does not yet support the definition of complex nested attributes. Consequently, we extend the Hybrid Wiki meta-model as described in the following section.

4.1.1. Extending Hybrid Wikis by Complex Attributes

In the context of this thesis, complex attributes refer to attributes whose values are complex or composite objects (Kim et al., 1989). As illustrated in Figure 4.2, a composite object is defined by a *CompositeType* and has an arbitrary number of attributes. The attributes of a composite

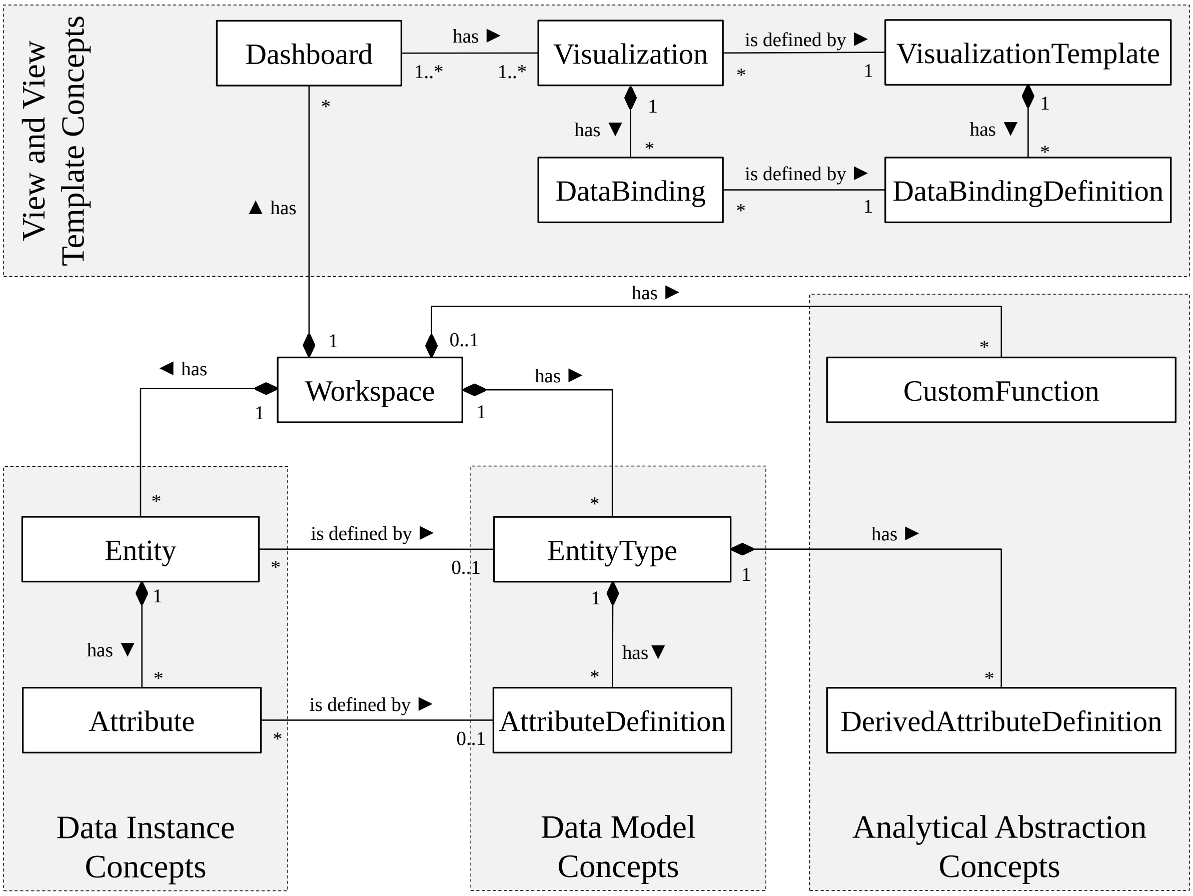


Figure 4.1.: An overview over the conceptual meta-model which is based on the Hybrid Wiki meta-model (Matthes et al., 2011; Reschenhofer et al., 2016a). The data model, analytical abstraction, and view concepts are described in detail in Sections 4.1, 4.2, and 4.3.

object can be either primitive (e.g., strings or numbers) or complex. The recursive definition of complex attributes enables arbitrarily nested objects as values of complex attributes.

In contrast to relations to other entities, a complex attribute and its value is part of the respective owner entity. A strict "part-of" relationship has multiple consequences: On the one hand, a composite object is exclusively part of only one entity, and cannot be contained by multiple entities. On the other hand, a complex attribute value's existence is strictly dependent on the existence of the parent entity, i.e., if an entity is deleted, all its attributes and thus all attribute values are deleted too. Therefore, complex objects are particularly useful to represent *physical part hierarchies* (Motschnig-Pitrik and Kaasboll, 1999).

In order to extend the Hybrid Wiki meta-model by the support for complex linked data, we extend the set of type constraints by a respective *ComplexConstraint* class (cf. Figure 4.3). Model designers can further restrict the structure of respective *ComplexValues* by defining a minimally required structure of the complex object. For example, when defining an *AttributeDefinition*

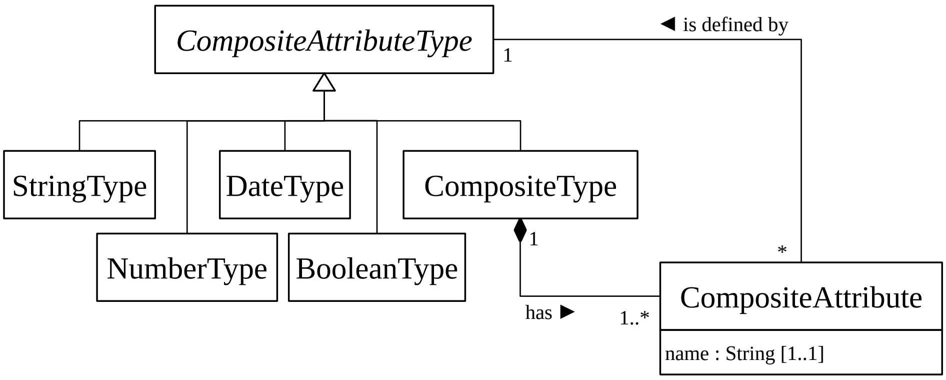


Figure 4.2.: The conceptual model for the complex type based on a variation of the composite pattern (Gamma et al., 1994). A complex type defined attributes which are either of a basic type (e.g., string, number) or again complex.

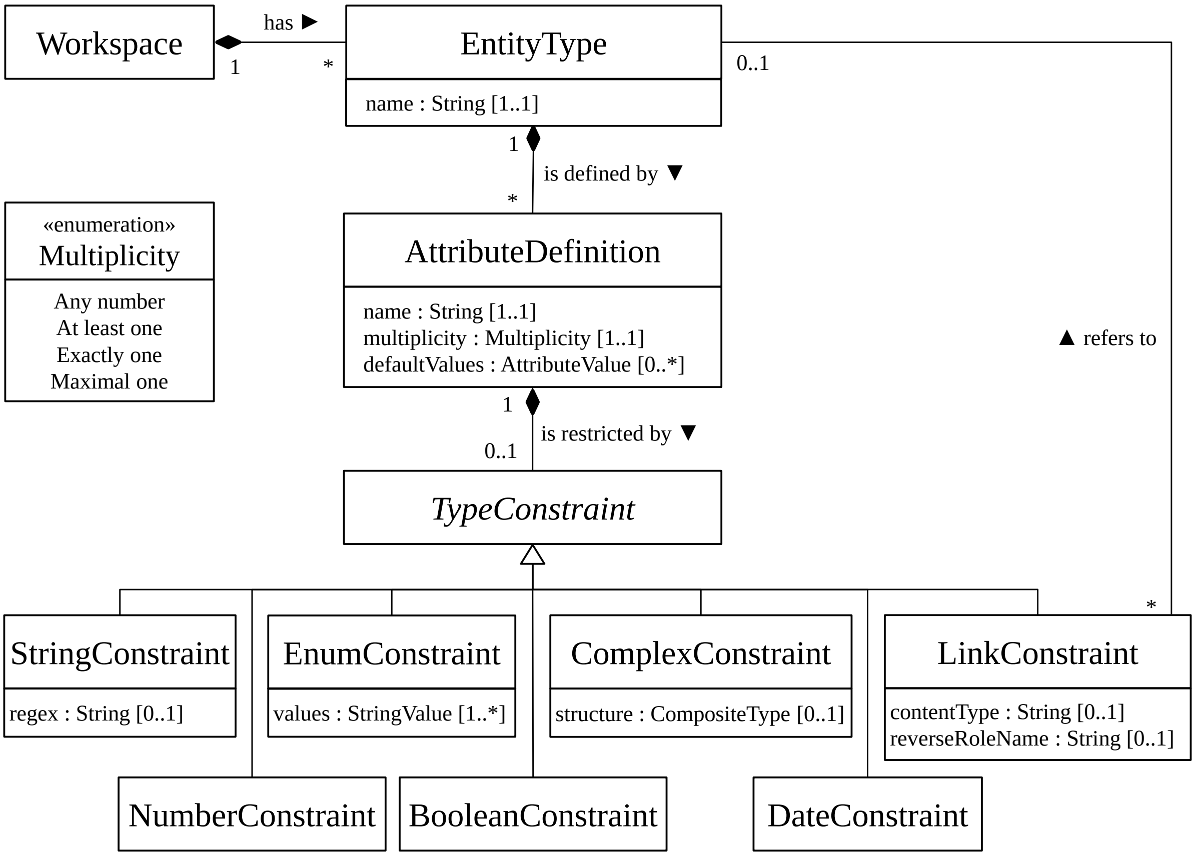


Figure 4.3.: A detailed excerpt of the extended Hybrid Wiki meta-model (cf. Figure 2.4) with a focus on the data model concepts. We extended the set of type constraints by the *ComplexConstraint*, which refer to composite objects (cf. Figure 4.2).

named "Address" as a complex attribute, the model designer can define a *ComplexConstraint* whose structure specification requires users to provide the nested string attributes "Street", "Postal Code", and "City" when instantiating a respective entity. By using the JavaScript Object Notation (JSON) and particularly, we can express a corresponding complex object as:

```
{
        "Address" : "Boltzmannstr. 3",
        "Postal Code" : "85748",
        "City" : "Garching",
        "Country" : "Germany"
}
```

Although this object also contains the attribute "Country", it stills complies to the minimal structure as defined by the model designer.

As stated by Kim et al. (1989), composite objects typically suffer from several shortcomings causing challenges when implementing them in object-oriented data models, e.g., versioning and access control for nested objects, softening the exclusiveness of the "part-of" relationship, or allowing the definition relations from or to inner objects. However, since the data models in the Hybrid Wiki system have to be designed at run-time through end-users, e.g., domain or modeling experts, it is essential to keep a balance between expressiveness and usability with respect to the definition of complex attributes. Therefore, we decided to implement a relatively simple model for composite objects which supports neither relations from or to inner objects nor an authentication or versioning mechanism for nested objects.

4.1.2. Assessment of the Meta-model's Expressiveness

In Section 2.2.2 we presented a semantic meta-model for spreadsheets. This meta-model captures semantic patterns which typically occur in real-world spreadsheets (Reschenhofer et al., 2016c). In order to assess the expressiveness of the extended Hybrid Wiki meta-model in Figure 4.3, we compare it to the semantic spreadsheet meta-model (cf. Figure 2.6) which represent semantic structures typically occurring in End-User Analytics (EUA).

In general, the *EntityType* of the Hybrid Wiki meta-model compares to the *EntityList* spreadsheet pattern. Analogously, *AttributeDefinitions* are equivalent to an *EntityList*'s *InputAttributes*. As described later on in Section 4.2.1, the Hybrid Wiki meta-model is extended by *DerivedAttributes* which have an eponymous counterpart in the semantic spreadsheet meta-model. *AttributeGroups* are implementable by complex attributes as long as inner attributes are only of primitive types. *AttributeAggregations* and *AttributeHistorizations* are not explicitly supported by the Hybrid Wiki meta-model. However, with custom functions as described in Section 4.2.1 as well as with the version history mechanism the Hybrid Wiki system provides means to define emulate the semantics of those two concepts. Furthermore, since *TimeSeries* are considered to be a specialization of *EntityLists*, they are also supported by the Hybrid Wiki system— at least under the aforementioned conditions.

EntityLists and *TimeSeries* account for 75 % of all semantic components in spreadsheets (Reschenhofer et al., 2016c). This number is even higher if we exclude charts as components which are not related to the data model. Even when taking into account the restrictions described in the previous paragraph, the Hybrid Wiki meta-model should be compatible with a majority of semantic structures in today's spreadsheets. Furthermore, the support for evolving complex linked data enables new EUA scenarios, e.g., EA analysis (Matthes et al., 2012a; Schneider et al., 2015).

Although further concepts could be implemented to support even more EUA use-cases, the application of the Hybrid Wiki system in different projects has shown that the right balance between usability and expressiveness of such a tool is essential for its practicability (Reschenhofer et al., 2016a). The relatively complex semantic spreadsheets meta-model of works because in spreadsheets data models are defined primarily in an implicit way, e.g., through the arrangement of data in the spreadsheet grid. However, while this basic spreadsheet property ensures that spreadsheet users do not have to care about the design of the data model, it is also one of the major causes for errors in spreadsheets (Reschenhofer et al., 2016c). Furthermore, if users have to understand a spreadsheet's design and semantics, the lack of an explicit data model can be problematic (Hermans, 2012). This is particularly the case in collaborative environments where multiple users design and analyze data cooperatively, since understanding the contribution of other users to the data model design is essential. By taking an explicit data modeling approach, we try to address the issues of an implicit one. For example, an explicitly defined data model enables the system to generate an accurate view of the data model and thus addressing requirement Req 4 without having to "guess" the model from the data. As a consequence of taking an explicit approach to end-user-driven data modeling, the complexity of a respective meta-model should be reduced in order to ensure the right balance between expressiveness and usability.

4.2. Analytical Abstractions for Hybrid Wikis

Based on the Hybrid Wiki meta-model which refers to the data model layer of our collaborative EUA framework (cf. Figure 3.2), we define concepts representing analytical abstractions and means empowering end-users to define the same. Thereby, we particularly address the requirements as derived in Section 3.2.2.

An important aspect in defining analytical abstractions is the query language which empowers end-users to define complex queries, metrics, and other analytical abstractions based on a user-definable and evolving complex linked data model at run-time. The requirements arising from this were already discussed in Section 3.2.2 and are addressed by the Model-based Expression Language (MxL). MxL enables end-users to access the underlying data through the corresponding data model and to transform this data through different kinds of operations, e.g., query, arithmetic, and logical operations (Reschenhofer, 2013). Its design and key features will be described in detail in Section 4.2.2. However, before elaborating on MxL's design, we first describe extensions to the meta-model by concepts enabling the definition of analytical abstraction.

4.2.1. Extending Hybrid Wikis by Analytical Abstractions

As shown in Figure 4.4, we added two main concepts to the Hybrid Wiki meta-model, namely *DerivedAttributeDefinitions* and *CustomFunctions* (Reschenhofer and Matthes, 2016a).

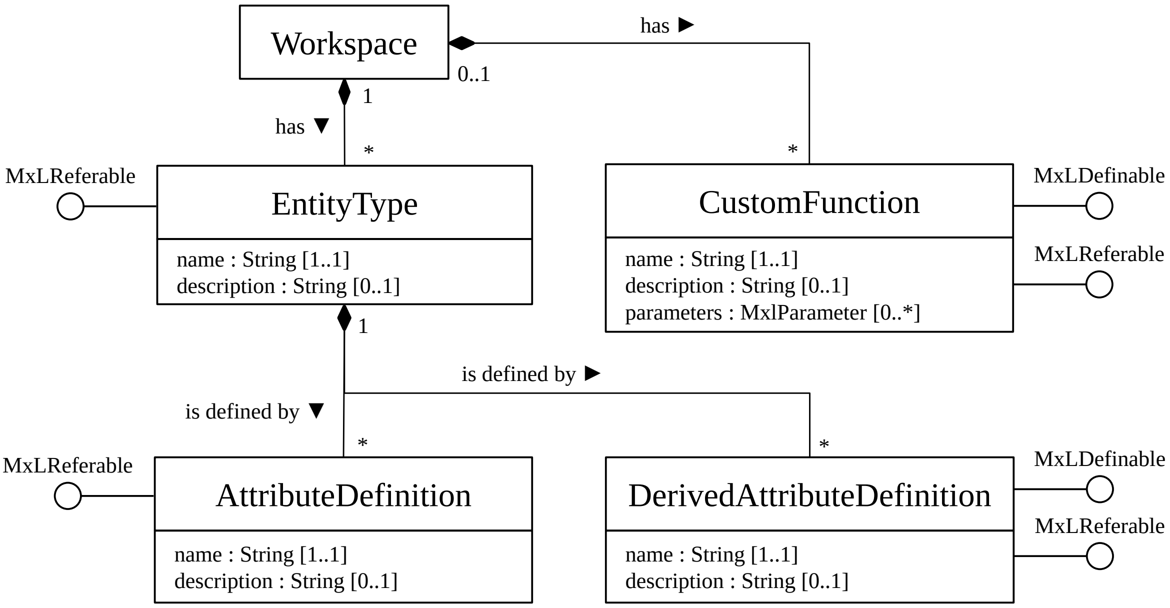


Figure 4.4.: A detailed excerpt of the meta-model (cf. Figure 2.4) with a focus on analytical abstraction concepts. The definitions of the interfaces *MxlDefinable* and *MxlReferable* are depicted in Figure 4.5, while the type *MxlParameter* describes function parameters as a pair of a name and a type.

A *DerivedAttributeDefinition* specifies a derived attribute whose value is automatically computed by the system instead of being manually maintained by the system's user. Chuah and Roth (1996) refer to derived attributes as an important information analysis and visualization concept. Derived attributes enable users to augment the data with analytical abstractions and to integrate their definition directly with the data model.

On the other hand, the *CustomFunction* concept addresses the reusability of computation rules and analytical abstractions and thus tackles requirement Req 8. End-users can define potentially parametrized *CustomFunctions* in order to store (partial) queries or calculations, which in turn can be reused by others, e.g., for the definition of *DerivedAttributeDefinitions* or other *CustomFunctions*. For example, this empower analytics end-users to outsource and encapsulate the definition of complex parts of an analytical abstraction definition to analytics professionals. In this sense, *CustomFunctions* foster the collaborative creation of analytical abstractions.

A *CustomFunction* can be either defined locally by assigning it to a *Workspace* or globally without relation to a *Workspace*. In the former case, *CustomFunctions* can access the *Workspace*'s specific data model and—assuming that *Workspaces* represent a domain and its model—implement domain-specific functionality. For example, a *Workspace* representing an EA model could contain a locally defined *CustomFunction* which represents a specific EA metric. In contrast, global *CustomFunctions* can be used across multiple workspaces and typically define a very generic functionality. Since it is not assigned to a particular *Workspace* and thus not part of a business domain, they neither access domain-specific data models nor implement domain-specific functionality. For example, an analytics professional could globally define a generic *CustomFunction* for the Shannon entropy (Shannon, 1948) in order to make it accessible in each *Workspace*.

The conceptual meta-model does not define *CustomFunctions* and *DerivedAttributeDefinitions* as *AccessControllables* (cf. Figure 2.5). Consequently, the access control rules for analytical abstractions are not defined explicitly, but derived from the respective *Workspace* they are assigned to. For example, editors of the *Workspace* are inherently allowed to adapt its custom functions. Furthermore, having the *reader* role for an analytical abstraction implies that a user is also allowed to execute is.

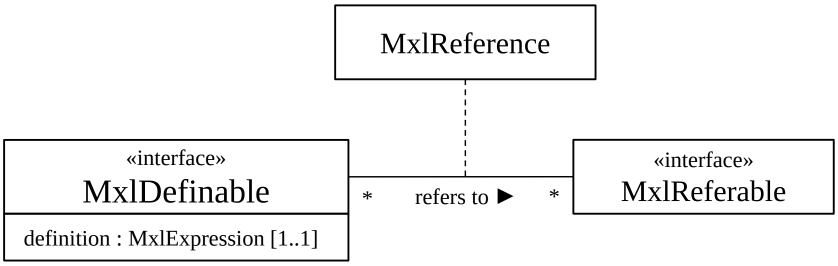


Figure 4.5.: This excerpt of the meta-model shows how the interfaces *MxlDefinable* and *MxlReferable* define semantic dependencies between analytical abstractions and data model elements.

Both *DerivedAttributeDefinitions* and *CustomFunctions* represent user-defined and model-based analytical abstractions. MxL empowers end-users to create them at run-time. Therefore, we define the concepts *DerivedAttributeDefinitions* and *CustomFunctions* as *MxlDefinables*, wherefore they implement a respective interface (cf. Figure 4.4). An *MxlDefinable* has an *MxlExpression* attribute which represents the actual query or calculation rule (cf. Section 4.2.2.3). By referring to data model elements (e.g., *EntityTypes* or *AttributeDefinitions*) or to analytical abstraction definitions (e.g., *DerivedAttributeDefinitions* or *CustomFunctions*), the user formulating the query defines semantic dependencies between *MxlDefinables* and so-called *MxlReferables* which represent all concepts which are referable and usable in MxL. The static type-safety of MxL enables the automated determination of those static dependencies through the static analysis of MxL expressions. Further details of MxL are described in the following section.

4.2.2. The Model-based Expression Language (MxL)

MxL[1] was originally developed as an untyped query and expression language to enable the definition of EAM KPIs and metrics at run-time (Monahov et al., 2013). Through incorporating

[1] Formerly known as TxL (Tricia Expression Language), whereas Tricia represents a concrete implementation of the Hybrid Wiki approach

the findings from applying an untyped version of MxL in different domains, we found out that the static analyzability of MxL expression enabled through the language's type-safety offers several benefits, e.g., the automated derivation of semantic dependencies, or automated refactoring of expressions on changes of dependent elements (Monahov, 2014; Reschenhofer et al., 2014a). Therefore, we redesigned MxL to be statically type-safe (Reschenhofer, 2013; Reschenhofer et al., 2014b). Based on the typed version of MxL, we extended and improved the language's type system and operational expressiveness in order to match the requirements identified in Section 3.2.2. In this section, we summarize the key features of MxL and describe the concepts which we extended as part of this thesis.

MxL is mainly inspired by the Object Constraint Language (OCL) which—originally developed by IBM—is part of the UML standard (Object Management Group, 2014b, 2015). OCL is an obvious choice for defining queries based on UML-based or UML-like data models (Störrle, 2013). However, in order to avoid restrictions with respect to query language-related design decisions (e.g., type system as discussed in Section 4.2.2.1), we decided to develop MxL as a variation of OCL instead of using the standardized OCL. Nevertheless, from a syntactical point of view, MxL is still very similar to OCL which fosters the familiarization of users with MxL if they already know OCL.

The most important properties of MxL are (Reschenhofer, 2013):

Functional Using the functional programming paradigm for query languages has a long tradition (Codd, 1972; Buneman and Frankel, 1979; Grust and Scholl, 1999; Wisnesky, 2014). As outlined by Buneman et al. (1995), functional programming languages are particularly suitable for defining complex queries on complex objects and collection types. Therefore, the functional programming paradigm addresses requirements Req 6 and Req 7 as described in Section 3.2.2.

Defining properties of functional programming and thus of MxL are the absence of side-effects as well as the support for higher-order functions (van Roy and Haridi, 2004). The former property means that expressions do not change a program state, while higher-order programming refers to the possibility of passing functions as arguments to other functions.

Object-oriented Since the Hybrid Wiki meta-model empowers end-users to define complex linked data, MxL is an object-oriented language in order to provide a convenient access to the data objects. As described in detail in Section 4.2.2.1, MxL also supports inheritance as one of the most important concepts of object-orientation (van Roy and Haridi, 2004).

Sequence-oriented Due to MxL's purpose as query language, it has to provide operations filter and transform sequences of data objects defined through the data model (Buneman et al., 1995), e.g., all *Entities* of a certain *EntityType*. Section 4.2.2.2 provides an overview over supported query operators.

Statically Type-safe In contrast to the initial version of MxL, the current one is statically type-safe (Pierce, 2002). This means that as part of the MxL interpretation process and after the syntax analysis through scanning and parsing steps, a type checker validates the static semantics of an MxL expression and thus ensures its semantic consistency at compile-time. In the context of MxL, compile-time refers to the point in time where an end-user defines an MxL expression (which can be at run-time of the implementing software tool), while the execution-time is the point in time where the expression is executed and queries actual data. In this sense, semantic analysis of an MxL query also implies that the MxL interpreter checks for the existence and consistency of all (user-definable) data model elements and analytical abstractions which are used and referred to within the expression.

Temporal Since addressing evolving data models, MxL was extended to be a temporal query language (Snodgrass, 1987), i.e., it is able to access the version history of the underlying data model (Bhat et al., 2015). This enables end-users to formulate queries which operate on a past state of the data model, and thus to analyze the evolution of the data.

In the following sections we elaborate on further important aspects and features of MxL. Basic language constructs are described in more detail in the master's thesis by Reschenhofer (2013), while MxL's temporal features are discussed by Bhat et al. (2015).

4.2.2.1. The MxL Type system

Figure 4.6 illustrates MxL's type system depicted as a hierarchy of *MxLTypes*. The *MxLType* itself is also part of the Meta Object Facility (MOF) diagram as a *M2* class (Object Management Group, 2014a). The most generic type and thus instance of the meta type *MxlType* is *Object*. This type can be found in most object-oriented programming languages and refers to the abstract type *Top* as defined by Pierce (2002). In a functional programming language, the type hierarchy is particularly relevant to determine which types of objects can be passed as parameters to typed functions. In our tool, a generic *Object* type allows the definition of generic functions which should operate on any type of object, e.g., the *toString* function. Furthermore, MxL supports the special value *null* representing "no value", or "empty value".

As subtypes of *Object*, MxL provides the (nullable) primitive types *String*, *Number*, *Boolean*, and *Date*. As known from most programming languages, *Strings* are character sequences encapsulated in quotation marks. *Number* represents both integers and decimals, while the only values of type *Boolean* are *true* and *false*. Instances of type *Date* are objects consisting of the attributes *day*, *month*, and *year*. This set of primitive types is derived from the primitive type constraints of the Hybrid Wiki meta-model in Figure 4.3, i.e., primitive *AttributeDefinitions* of the Hybrid Wiki system can be straightforwardly mapped to corresponding *MxlTypes*. MxL provides a set of basic operators for objects of primitive types, e.g., arithmetic and comparison operations for numbers, logical operations and conditionals for boolean values, or date comparison operators for dates. For example, the following is a valid MxL expression:

```
let x = 2 * 3 in
if x^2 > 5 and x < 10
 then 15.5
 else 0
```

This MxL expression also uses the *let-in* construct in order to define a constant x in a certain scope (which comes after the *in*).

In addition to primitive types, MxL supports a couple of generic types which define one or

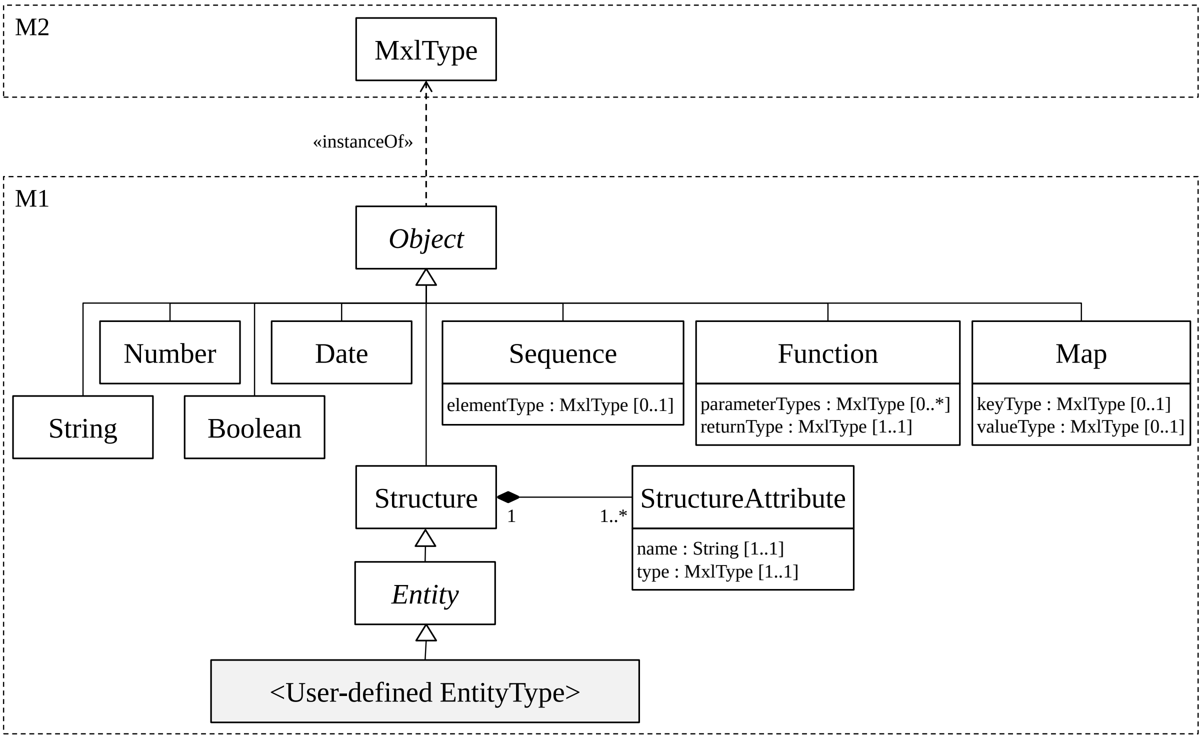


Figure 4.6.: A conceptual overview over MxL's type hierarchy. We use MOF (Object Management Group, 2014a) in order to be able to describe generic types.

more type parameters (Pierce, 2002). The first of them is the *Sequence* type which represents an ordered multi-set, i.e., each element has a dedicated index within the *Sequence*, and there can be duplicate elements. The optional type parameter *elementType* specifies the type of the elements of the *Sequence*. To define type parameters in MxL, we use the notation *GenericType*<*TypeParameter1, TypeParamater2,…, TypeParameterX*>. For example, a numerical collection can be specified by *Sequence*<*Number*>. If no type parameter is specified, the element type is implicitly defined as *Object*. If a *Sequence* contains elements of different types, the element type of the *Sequence* is the minimal common subtype of all its elements (cf. intersection types as defined by Pierce (2002)). For example, a collection containing a *String* and a *Number* is of type *Sequence*<*Object*>. MxL defines subtype relationships between *Sequence* types with different element types, i.e., a sequence type *Sequence*<T_1> is the subtype of a sequence type *Sequence*<T_2>, if and only if T_1 is a subtype of T_2. For example, *Sequence*<*Number*> is a subtype of *Sequence* (equivalent to *Sequence*<*Object*>).

For the construction of *Sequences*, we use a common notation with a comma-separated list of its elements and surrounding square brackets. For example, a *Sequence* with three *Strings* as its elements can be constructed as follows:

```
["One", "Two", "Three"]
```

Elements of a sequence can be accessed by their (zero-based) index, e.g.:

```
let x = ["One", "Two", "Three"] in
x[0] /* returns "One" */
```

The multi-valued Hybrid Wiki *Attributes* are mapped to *Sequences*, i.e., if the multiplicity of an *AttributeDefinition* is either "At least one" or "Any number", the corresponding *MxlType* will be a *Sequence* whose element type is still derived from the type constraint. For example, an *AttributeDefinition* with a *NumberConstraint* and with the multiplicity "Any number" is mapped to *Sequence<Number>*.

As a higher-order language, MxL supports the *Function* type and thus functions as first-class objects. A function takes a certain number of parameters and returns a value. In MxL, higher-order functions are heavily used as generic query operations (cf. Section 4.2.2.2). The types of the parameters as well as the return type can be specified as type parameters of the *Function* type, whereas the last type parameter represents the return type and the others the ordered parameter types. For example, *Function<Number,Number,Boolean>* refers to a function which accepts two numerical parameters and returns a boolean value. Adding a question mark to a parameter type defines it as an optional parameter. For example, a function of type *Function<Number,Number?,Boolean>* accepts one or two parameters. The most trivial function is the *identity function* which just returns its only parameter (Pierce, 2002).

MxL defines subtype relationships between different *Function* types. However, we have to consider *contravariance* for the parameter types and *covariance* for the return type of the *Function*. This means that a type *Function*<P_1,...,P_N,R> is subtype of *Function*<P_1',...,P_M',R'>, if

- N is equal to M, i.e., they have the same number of parameters,
- P_1',...,P_M' are subtypes of P_1,...,P_N respectively (*contravariance*), and
- R is subtype of R' (*covariance*).

As a consequence, the most general *Function* type is not *Function<Object,..,Object,Object>*, but *Function<Any,..,Any,Object>*, whereas *Any* represents a pseudo type as subtype of every other *MxLType*. It is the analogue to the *Bottom* type as defined by Pierce (2002) and the counterpart to *Object*, which is the supertype of all other *MxlTypes*.

For the purpose of constructing functions, MxL supports *lambda* expression (Pierce, 2002) using the notation known from C#, with a list of (type-annotated) parameters followed by an arrow (=>) and the function's implementation (Pierce, 2002):

```
(x:Number, y:Number) => x + y
```

The function defined in this example takes two numerical parameters and returns the sum of them. The parameter types are explicitly annotated, while the return type is inferred automatically based on the implementation part of the lambda expression. As described later on, in most cases parameter types can be automatically derived from the lambda expression's context through type interference, i.e., they do not have to be specified explicitly and can be omitted.

For the execution of functions, we use the notation known from most programming languages, namely *FunctionObject(Param*$_1$*,...,Param*$_N$*)*. The function object can either be a lambda expression, or a named function referred to via an identifier, e.g.:

```
let add = (x:Number, y:Number) => x + y in
add(1, 2) /* returns 3 */
```

The third generic type in MxL is *Map* which represents a typed dictionary and thus key-value pairs. The types of the keys and the values can be specified by the type parameters *valueType* and *keyType*. For example, the type *Map<String,Number>* defines a dictionary which maps a *String* to a *Number*. The main reason for introducing this type was the *groupby* function as described in Section 4.2.2.2: Grouping elements by a certain key requires a respective data structure allowing users to easily access and operate on the identified groups.

Again, MxL defines subtype relationships between different *Map* types. However, since a *Map* type *Map*$<K,V>$ can be considered as a function of type *Function*$<K,V>$, we again have to apply *covariance* and *contravariance*. As a consequence, *Map*$<K,V>$ is subtype of *Map*$<K',V'>$, if K' is subtype of K, and V is subtype of V'. *Maps* can be constructed by the notation *[Key*$_1$*: Value*$_1$*, ..., Key*$_N$*: Value*$_N$*]*, e.g.:

```
["One" : 1, "Two" : 2, "Three" : 3]
```

This example represents a dictionary of type *Map<String,Number>* with three dictionary items, which can be accessed by their respective key, e.g.:

```
let x = ["One" : 1, "Two" : 2, "Three" : 3] in
x["One"] /* returns 1 */
```

In addition to primitive and generic types, MxL implements the generic composite type *Structure* consisting of a specifiable set of attributes. Those attributes are defined as pairs of a name and a type by using the notation *Structure<name*$_1$*: type*$_1$*, ..., name*$_N$*: type*$_N$*>*. For example, *Structure<name: String, age: Number>* defines the type of a composite object which has a string attribute "name" and a numerical attribute "age". The *Structure* type is MxL's analogue to the *ComplexConstraint* as described in Section 4.1.1, i.e., *AttributeDefinitions* with *ComplexConstraints* are mapped to respective *Structures* in MxL.

The subtype relationship between different *Structure* types is defined as follows: *Structure<name*$_1$*: type*$_1$*, ..., name*$_N$*: type*$_N$*>* (referred to as A) is subtype of *Structure<name*$_1$*': type*$_1$*', ..., name*$_M$*': type*$_M$*'>* (referred to as B), if

- the set of attribute names of A is a superset of the set of names of B, and
- the type of each attribute of B is a supertype of the type of an eponymous attribute of A.

For example, *Structure<name:String,age:Number>* is a subtype of *Structure<name:String>*, but not of *Structure<name:String:birthdate:Date>*. In this sense, MxL is implementing a typed variant of *duck typing* as a form of polymorphism (Chugh et al., 2012). In this context, duck typing refers to the method of determining subtype relationships by an object's structure rather than by a strict "is-a" relation.

In MxL, composite objects can be constructed by using the JavaScript Object Notation (JSON), whereas their attributes are accessed by respective identifiers, e.g.:

```
let x = { name : "Thomas Reschenhofer", age : 28 } in
x.age /* returns 28 */
```

As a subtype of *Structure*, the type *Entity* is the abstract supertype for all user-defined *EntityTypes* as defined by the meta-model in Figure 4.3. Therefore, *EntityTypes* can be interpreted as specific *Structure* types, whereas the *EntityType*'s *AttributeDefinitions* are translated to respective *StructureAttributes*. As a consequence, a function defining a parameter of type *Structure<age:Number>* also accepts Hybrid Wiki *Entities* whose *EntityType* defines an *AttributeDefinition* named "age" and restricted by a *NumberConstraint*. Furthermore, objects of type *Entity* implicitly have built-in attributes, e.g., an ID and a name.

In summary, the MxL type system complies to the type constraints in Figure 4.3 on the one hand, and additionally provides types (e.g., *Map* and *Function*) enabling the implementation of generic and common query functions (e.g., *groupby*) on the other hand. Notable extensions compared to the MxL version as described by Reschenhofer (2013) are the implementation of the types *Map* and *Structure* as well as the corresponding *duck typing* approach.

4.2.2.2. Basic Functions in MxL

Basic functions are the default functions provided by MxL. They are summarized in the appendix in Section A. However, we will briefly describe some of them in more detail in the following section.

As consequence of its sequence-orientation, MxL provides a default set of basic functions which operate on sequences and represent common query operations. More specifically, MxL implements the Standard Query Operators (Heijlsberg and Torgersen, 2013) as defined by Microsoft and as also applied in their Language Integrated Queries (LINQ). Those operations include common query functions, quantifier functions, set functions, element functions, partitioning functions, and aggregation functions.

Most basic functions in general and sequence functions in particular are higher-order functions, i.e., they accept other functions as parameters. For example, the *select* function accepts a function as parameter which maps each element of the original sequence to a new value (cf. Table A.7). Therefore, the return type of the *select* function is determined by the return type of the given mapping function. For example, the following *MxL* expression shows the application of the select function to an object of type *Sequence<Number>* with a parameter of type *Function<Number,String>*. The return type of the select function is inferred automatically and thus is *Sequence<String>*:

```
let x = [1, 2, 3] in
x.select(toString) /* returns ["1", "2", "3"] */
```

Analogously, the *where* function accepts a predicate (a function returning a boolean value) as parameter and removes those elements from the original sequence which do not fulfill this predicate. Consequently, the return type of the where function is just the same as the type of the original sequence:

```
let x = [1, 2, 3] in
x.where(e => e > 2) /* returns [3] */
```

In this example, we do not need to annotate the type of the lambda's parameter since it can be automatically inferred from the context.

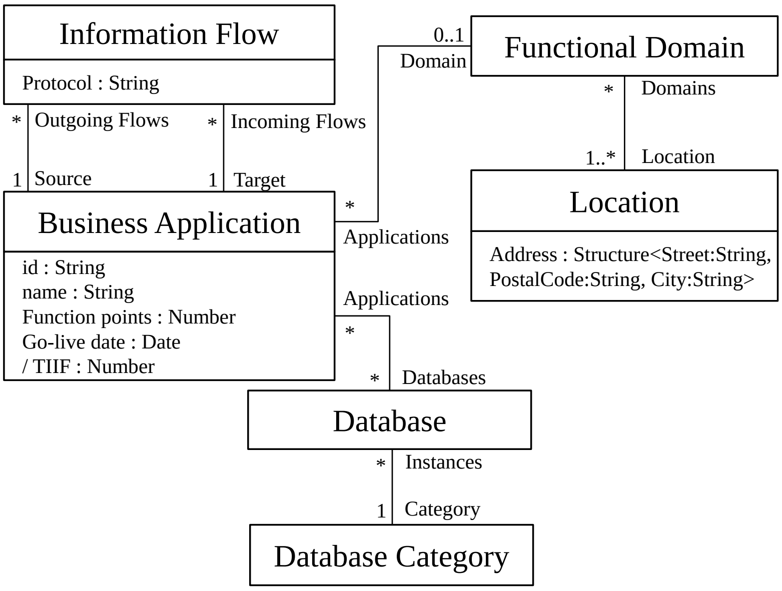


Figure 4.7.: An exemplary EA model capturing the application landscape and related concepts. This model is inspired by the work of Schneider et al. (2015). We use the MxL type notation in order to specify the *ComplexConstraint* of the *Location*'s *Address*.

Instead of applying the query functions to a statically constructed sequence of numbers, we can also use the *find* keyword of MxL in order to retrieve all entities of the specified *EntityType*. By considering that the underlying Hybrid Wiki data model defines EA model as illustrated in Figure 4.7, we can retrieve all business application as follows (the apostrophe is required since the name of the *EntityType* contains a special character):

```
find 'Business Application'
```

Based on this sequence of type *Sequence<'Business Application'>* we can now apply arbitrary sequence operations. For example, to determine the number of business applications whose function points exceeds a certain value, we can formulate the following query:

```
find 'Business Application'
  .where(ba => ba.'Function points' > 10)
```

Through type interference, the parameter type of the lambda expression is automatically determined and thus can be omitted by the user. Furthermore, the semantic analysis of an MxL expression allows users to apply an *implicit lambda* expression instead of explicitly formulating it (Reschenhofer, 2013). As a consequence, the system interprets partial MxL expressions as lambda expressions, and thus certain identifiers as attributes of the implicit lambda's parameter. The following query is semantically equivalent to the former one:

```
find 'Business Application'
  .where('Function points' > 10)
```

In order to define more complex queries, we can append further query operations, e.g., aggregations, and combine them with other operations and functions. For example, we can define a query which implements the database heterogeneity measure as defined by Schneider et al. (2015) and as already introduced in Section 1.1 as follows:

```
/* Get a list of databases used by business applications. This list might contain duplicates
   if a database is used by more than one application */
let list = find 'Business Application'.selectMany(Databases) in

/* Determine the total number of database usages */
let totalNumber = list.count() in

/* Map each distinct database to the number of applications by which it is used */
let groups = list.groupby().asSequence() in

/* if there are no database usages, the heterogeneity metric is defined to be 0 */
if totalNumber = 0
 then 0
 else
   /* this is an implementation of the Shannon Entropy */
   groups.sum(
     let fi = value.count() / totalNumber in
     if fi = 0 then 0 else fi * ln(1 / fi)
   )
```

The functions used in this example are described in detail in the appendix (Section A).

4.2.2.3. Defining Custom Functions and Derived Attributes

As shown by the previous example, MxL expressions can become rather complex. Refactoring and restructuring an expression can help to reduce the complexity of an expression, and increase the reusability of certain parts of an expression. Furthermore, this allows users to outsource the implementation of (partial) queries, and to share them as potentially parametrized *CustomFunctions*. In the previous example, we can define the Shannon entropy as a reusable *CustomFunction*. As illustrated in Figure 4.4, a *CustomFunction* has a name and an optional description. The name defines the identifier by which *CustomFunctions* can be referred to within an MxL expression, while the description provides additional textual information which might be particularly helpful for other users which intend to reuse the *CustomFunction*. Furthermore, a *CustomFunction* can define a list of parameters which in turn consist of a name and a parameter type. In general, we use the MxL type notation to specify parameters and their types. The actual implementation of the *CustomFunction* is captured by the *definition* attribute derived from the *MxlDefinable* interface. *CustomFunction* can be assigned to workspaces which implicitly links them to the respective data model. For example, assigning a *CustomFunction* to a workspace defining the EA model depicted in Figure 4.7 enables access to the data model elements, e.g., retrieving entities of a *Workspace*-specific *EntityType*.

Referring back to the previous example in Section 4.2.2.2, an *Analytics Professional* can define *entropy* as a *CustomFunction* with a respective and meaningful textual description. The only parameter is named "list" and is of type *Sequence*, whereas the element type does not have to be specified explicitly and thus is implicitly *Object*. This parameter is available as an identifier within the following MxL expression representing the definition of the *CustomFunction*:

```
let totalNumber = list.count() in
let groups = list.groupby().asSequence() in

if totalNumber = 0
 then 0
 else
   groups.sum(
     let fi = value.count() / totalNumber in
     if fi = 0 then 0 else fi * ln(1 / fi)
   )
```

By reusing this *CustomFunction*, we can reformulate the previous query for the database heterogeneity metric as:

```
/* Get a list of databases used by business applications. This list might contain duplicates
   if a database is used by more than one application */
let list = find 'Business Application'.selectMany(Databases) in

/* Apply the Shannon entropy */
entropy(list)
```

Similarly, we can define *DerivedAttributeDefinitions* for *EntityTypes*. In contrast to *AttributeDefinitions*, they do not describe the to-be structure of respective entities, but define attributes whose values are automatically computed based on a given MxL query. Therefore, the class *DerivedAttributeDefinitions* implements the *MxlDefinable* interface as illustrated in Figure 4.4. Derived attributes are defined once at *EntityType*-level, but executed for each of its *Entities*.

Referring back to the exemplary EA model in Figure 4.7, the type *Business Application* contains a derived attribute *TIIF* which is an acronym for "Transitive Incoming Information Flows". By interpreting the application landscape as a directed graph whose nodes are *Business Applications* and edges are *Information Flows*, *TIIF* is defined as the number of its descending nodes. With

MxL, we can define *TIIF* as *DerivedAttributeDefinition* for the *EntityType Business Application* as follows:

```
this.transitive(ba => ba.'Incoming Flows'.select(f => f.Source))
  .count()
```

In the definition of *DerivedAttributeDefinition*, end-users can use the *this* keyword in order to access the *Entity* for which the derived attribute's value is computed, which in this case is an *Entity* of type *Business Application*. *transitive* is defined as a function applicable to an object of any type *T*. Its parameter is a mapping function of type *Function<T,Sequence<T> >*. Initially applied for the source object, the map function is recursively invoked for the objects returned by it until no more distinct objects are retrieved. The result of the *transitive* function is a *Sequence* of all traversed objects. In the previous example, the mapping function navigates to incoming information flows and subsequently to their target application. By recursively repeating this navigation step through the *transitive* function and by counting the elements of the resulting sequence, the *DerivedAttributeDefinition* computes the desired metric.

MxL offers specific language features to shorten expressions and to make them easier to read. In addition to the already described *implicit lambda*, this also includes the *implicit this* as known from *Java* or *C#* (Reschenhofer, 2013): If an identifier cannot be resolved by *MxL*'s semantic analyzer, it is first interpreted as an attribute of an *implicit lambda*'s parameter. If this fails, the identifier is interpreted as an attribute of the object represented by the *this* keyword. In case this also fails, the semantic analysis will signal a respective validation error. Therefore, we can reformulate the previous definition of *TIIF* as follows:

```
transitive('Incoming Flows'.select(Source)).count()
```

4.2.3. Assessment of the Expressiveness of MxL

As shown by Pemberton and Robson (2000), sorting and database functions are the most frequently used operations in spreadsheets. By implementing the Standard Query Operators (Heijlsberg and Torgersen, 2013) as basic functions in MxL, we addressed this issue. However, the question remains if the type system and set of implemented functions is sufficient to define complex queries based on complex linked data.

To answer this question, we observed EA KPIs and metrics which are derived from related literature and from practice. There are two main reasons for choosing the EAM domain as setting for evaluating the expressiveness of MxL: On the one hand—as already introduced in Section 1.1—in EAM different stakeholders collaboratively define complex linked data models and corresponding metrics. On the other hand, the sebis chair has a long tradition in doing research about EAM in cooperation with industry partners, wherefore we have access to both data and metrics from industry.

In this context, Matthes et al. (2012a) developed a catalog consisting of 52 EAM KPIs gathered from literature and industry. Each KPI is described based on a predefined template capturing general and organization-specific structure elements (Matthes et al., 2012b). One of those general structure elements is the *Calculation* representing a textual description of how the KPI has to be calculated based on a given information model. For example, the information model for the EAM KPI *Application continuity plan availability* is depicted in Figure 4.8. The corresponding calculation is textually described as:

> *The number of critical applications where tested IT continuity plan is available divided by the total number of critical applications*

.

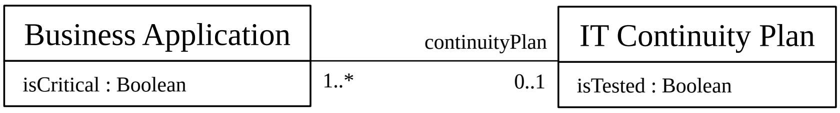


Figure 4.8.: The information model for the EAM KPI *Application continuity plan availability* (Matthes et al., 2012a).

In order to evaluate if MxL's expressiveness, we implemented all KPIs of the EAM KPI catalog (Monahov, 2014). For example, the *Application continuity plan availability* KPI was defined as follows Reschenhofer (2013):

```
/* Determine all critical applications */
find('Business Application').where(isCritical)

/* Calculate ratio of critical applications covered by a continuity plan */
  .ratio(coveringPlan <> null)
```

In the context of a related research project, Schneider et al. (2015) present EA complexity metrics which were derived from literature and industry. The metrics were categorized into *Heterogeneity-focused*, *Industry*, and *Topology-based* metrics. The database heterogeneity metric shown and explained in Section 4.2.2.2 is a representative of the first class of metrics, while *Industry* metrics are similar to the KPIs of the EAM KPI catalog. The topology-based metrics interpret the application landscape as a directed graph with *Business Applications* as nodes and *Information Flows* between them as edges. Based on the topology of this graph, the metrics identify four clusters, namely *core*, *control*, *shared*, and *periphery* applications (Lagerström et al., 2014).

As shown in Figure 4.9, the application of MxL for defining EAM KPIs and EA complexity metrics served as an evaluation of the expressiveness of the type system and basic functions. Based on this evaluation, we had to extend MxL continuously, e.g., by new *MxlTypes* and basic functions. For example, the types *Map* and *Structure* as well as the basic functions *transitive* and *ln* were added based on feedback gained from the implementation of specific KPIs and metrics.

The MxL type hierarchy as shown in Figure 4.6 as well as the basic functions as described in the appendix in Section A represent a state of MxL which provides an expressiveness which is sufficient to implement all identified EAM KPIs and EA complexity metrics. Therefore, we

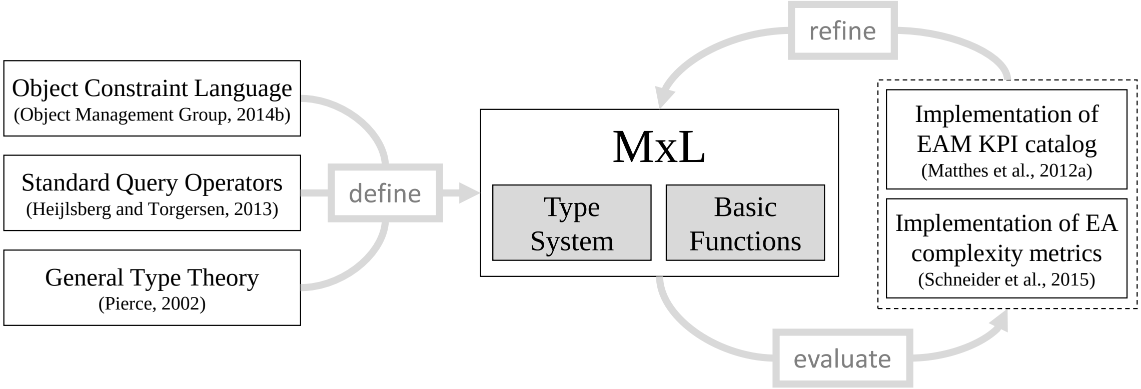

Object Constraint Language
(Object Management Group, 2014b)
Standard Query Operators
(Heijlsberg and Torgersen, 2013)
General Type Theory
(Pierce, 2002)
define
MxL
Type System
Basic Functions
refine
evaluate
Implementation of EAM KPI catalog
(Matthes et al., 2012a)
Implementation of EA complexity metrics
(Schneider et al., 2015)

Figure 4.9.: Evaluation and refinement of MxL based on its application for defining EAM KPIs (Matthes et al., 2012a) and EA complexity metrics (Schneider et al., 2015).

consider the expressiveness of MxL as sufficient for the definition of complex queries based on complex linked data in general.

4.3. Views and View Templates for Hybrid Wikis

On the top of the data model and analytical abstractions layer, the collaborative EUA framework in Figure 3.2 defines concepts for views which are based on respective view templates. View templates represent abstract viewpoints as described by Schaub et al. (2012) and have to be instantiated with a data model in order to generate concrete views. In the following section, we describe both the view and view template layer, and thus address the requirements as derived in Section 3.2.3.

4.3.1. Extending Hybrid Wikis by Views

Figure 4.10 shows the extended Hybrid Wiki meta-model including view and view template concepts (Reschenhofer and Matthes, 2016a).

The *View* class describes abstract views representing a composition of multiple data *Visualizations*. Specific examples for *Views* are *Dashboards* and *Reports*. Since assigned to a dedicated *Workspace*, a *View* is considered to be domain-specific, i.e., it visualizes domain-specific data defined through domain-specific data models and transformed through domain-specific analytical abstractions. Furthermore, the *View* class implements the *AccessControlled* interface (cf. Figure 2.5) to enable the specification of which users are allowed to see and edit it.

The *Visualization* represents a self-contained visual component which can be part of potentially multiple *Views*. It is defined by a *VisualizationType* which contains the actual implementation and thus defines the type of the visualization (e.g., bar chart, graph view, etc.). Since implementing the *AccessControlled* interface, the access to *VizualizationTypes* can be restricted, e.g., to specify which users are allowed to modify them. The kind of data which can be visualized

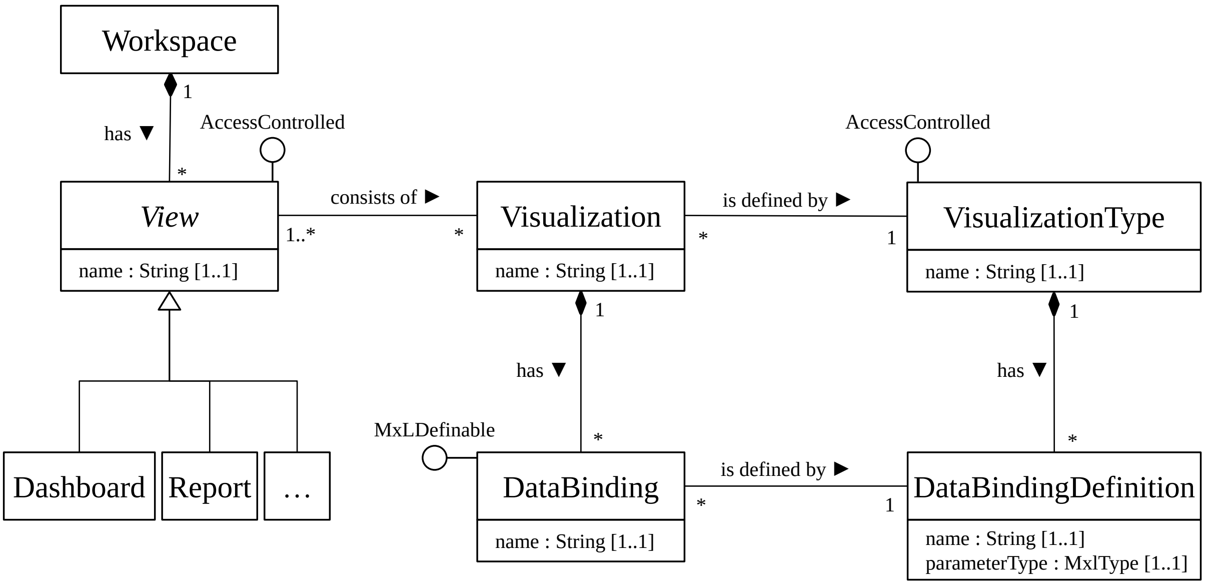


Figure 4.10.: A detailed excerpt of the meta-model (cf. Figure 2.4) with a focus on view and view template concepts. The definitions of the interfaces *MxlDefinable* and *AccessControlled* are depicted in Figures 4.5 and 2.5 respectively.

by a certain *VisualizationType* is specified by a data input interface defined by a set of typed *DataBindingDefinitions*. By using the MxL type system and the corresponding MxL type notation, we can define complex typed *DataBindingDefinitions* which are compliant to the types of the underlying analytical abstractions. For example, if a *DataBindingDefinition* is of type *Number*, a user which instantiates a corresponding *Visualization* has to make sure to define an MxL query which returns a numerical value for the respective *DataBinding*.

When creating a *Visualization* based on a *VisualizationType*, the user has to instantiate each of the *VisualizationType*'s *DataBindingDefinitions* and thus define a corresponding *DataBinding*. A *DataBinding* links data model elements or analytical abstractions to *Visualizations* (Soriano et al., 2007). By using MxL, users are able to define complex queries and to transform the underlying data or analytical abstractions to match them with the corresponding *DataBindingDefinition*'s type. As a consequence, *DataBindings* implement the *MxlDefinable* interface as defined in Figure 4.5 and thus have semantic dependencies to *MxlReferables*, e.g., *AttributeDefinitions* or *CustomFunctions*.

In this sense, the concepts defined in Figure 4.10 relate to the conceptual framework as defined by Hauder et al. (2012) and Schaub et al. (2012) and as already described in Section 2.3.2: *VisualizationTypes* represent *Abstract Viewpoints*, whereas the *Abstract view model* is defined by it *DataBindingDefinitions*. *Visualizations* are the analogue to the concrete *Viewpoints*. The corresponding *DataBindings* explicitly specify the *View model* which describes the *Visualization*'s information demand.

A view's binding to a domain-specific data model is separated from its actual implementation, and only connected through a well-defined data input interface. Through this separation, technology-savvy users, e.g., web developers and web designers, are able to implement generic visualization types, while domain experts are responsible for binding them to domain-specific data models (cf. requirement REQ 13). This also compares to the Enterprise Mashup (EM) environment as defined by Pahlke et al. (2010): Virtualized components (*VisualizationTypes*) can be assembled by end-users to EM applications (*Views*) through binding them to different kinds of resources (*DataBinding*).

4.3.2. Definition of Data Bindings

The actual logic of a *Visualization* is implemented in its *VisualizationType*, whereas the *DataBindingDefinitions* represent parameters by which a concrete *Visualization* instance can pass concrete data to the implementation. For example, a *VisualizationType* "Directed Graph" represents a *VisualizationType* showing a graph with directed edges. The actual set of nodes and edges can be defined as *DataBindingDefinitions*, whereas they have to be bound to an underlying data model or analytical abstractions when creating an instance of the "Directed Graph". Therefore, a "Directed Graph" can be defined with two *DataBindingDefinitions* "nodes" of type *Sequence<id:String, label:String>* and "edges" of type *Sequence<Structure<from: String, to: String> >*. The *DataBindingDefinitions* do not refer to any domain-specific concepts. As a consequence, the *VisualizationType* "Directed Graph" can be considered as a generic view template which can be reused in different domains for the visualization of diverse graph-like structures.

In order to instantiate a "Directed Graph", users have to select an existing *VisualizationType* and subsequently bind it to the underlying data model or analytical abstractions by formulating respective queries for the *DataBindingDefinitions*. Referring back to the exemplary data model in Figure 4.7, we can visualize the application landscape consisting of *Business Applications* and their *Information Flows* as a graph. For this purpose, the *DataBinding* "nodes" can be defined as follows:

```
find 'Business Application'
  .select({id: id, label: name})
```

This MxL query retrieves all entities of type *Business Application*, and maps each of them to a *Structure* which conforms to the corresponding *DataBindingDefinition* of the *VisualizationType*. This means that the *id* and *name* of each business application is mapped to a corresponding *id* and *label* attribute. Since the *id* attribute is mapped to an eponymous *StructureAttribute*, we can also write the following query:

```
find 'Business Application'
  .select({id, label: name})
```

With this notation, MxL interprets an unpaired identifier within the constructor of a *Structure* as a *StructureAttribute*, whose name is defined by the identifier's name, and whose value is the identifier's value in its current context.

The *DataBinding* "edges" can be defined as follows:

```
find 'Information Flow'
  .select({from: Source.id, to: Target.id})
```

Analogous to the *VisualizationType* "Directed Graph" and its instantiation for visualizing an application landscape based on the exemplary EA model in Figure 4.7, we can also define a very simple *VisualizationType* "Simple Number" to show the value of the exemplary database heterogeneity metric as described in Section 4.2.2.3. The number which has to be shown is defined as a corresponding numerical *DataBindingDefinition*.

Figure 4.11 illustrates an exemplary dashboard consisting of the aforementioned visualizations. The UML object diagram shows the relations between *Visualizations* and *VisualizationTypes*, and between *DataBindings* and *DataBindingDefinitions*. Furthermore, it also highlights the implicitly derived semantic dependencies from *DataBindings* and analytical abstractions as *MxlDefinable* to analytical abstractions and data model elements as *MxlReferables*. In this sense, it represents the analysis model as holistic perspective on data model elements, analytical abstractions, and views including semantic dependencies between each other (cf. Section 3.1).

4.3.3. Assessment of the Expressiveness of the View Concepts

As described in Section 4.3.1, a view's binding to a domain-specific data model is separated from its actual implementation, and only connected through a well-defined data input interface. In this Enterprise Mashup-like architecture, end-users are empowered to tailor views to their individual needs (Pahlke et al., 2010). Furthermore, making the parameterized view templates explorable by different kinds of users fosters collaboration and innovation (Soriano et al., 2007). Regarding the technical expressiveness, the design decision of separating the binding to concrete data from the view's implementation has the following two main consequences.

First, visualizations consume data through the data input interface which is constituted by a set of *DataBindingDefinitions*. Since using the MxL type notation to define *DataBindingDefinitions*, and MxL queries to define corresponding *DataBindings*, the expressiveness of the input data of views is derived from MxL's expressiveness and described by MxL's type system. As argued in Section 4.2.3, MxL supports the definition of complex analytical abstractions based on complex linked data, which in turn also applies to *DataBindings*.

Second, the expressiveness of the visualizations themselves strongly depends on the respective technical platform. Although the expressiveness of the data input is only restricted by MxL's type system, technical constraints of the environment in which the *VisualizationTypes* are implemented might impose further limitations with respect to how the data can actually be processed and visualized. This also includes a potential parsing process from the data retrieved through an MxL query in order to adapt it to the technical environment. However, while the expressiveness might suffer from limitations of the technical environment, it can also benefit from an already existing ecosystem. For example, libraries and frameworks already existing for a chosen technical platform can be used in order to implement *VisualizationTypes*. As described later on in Chapter 5, our prototype will be web-based. On the one hand, this allows us to reuse

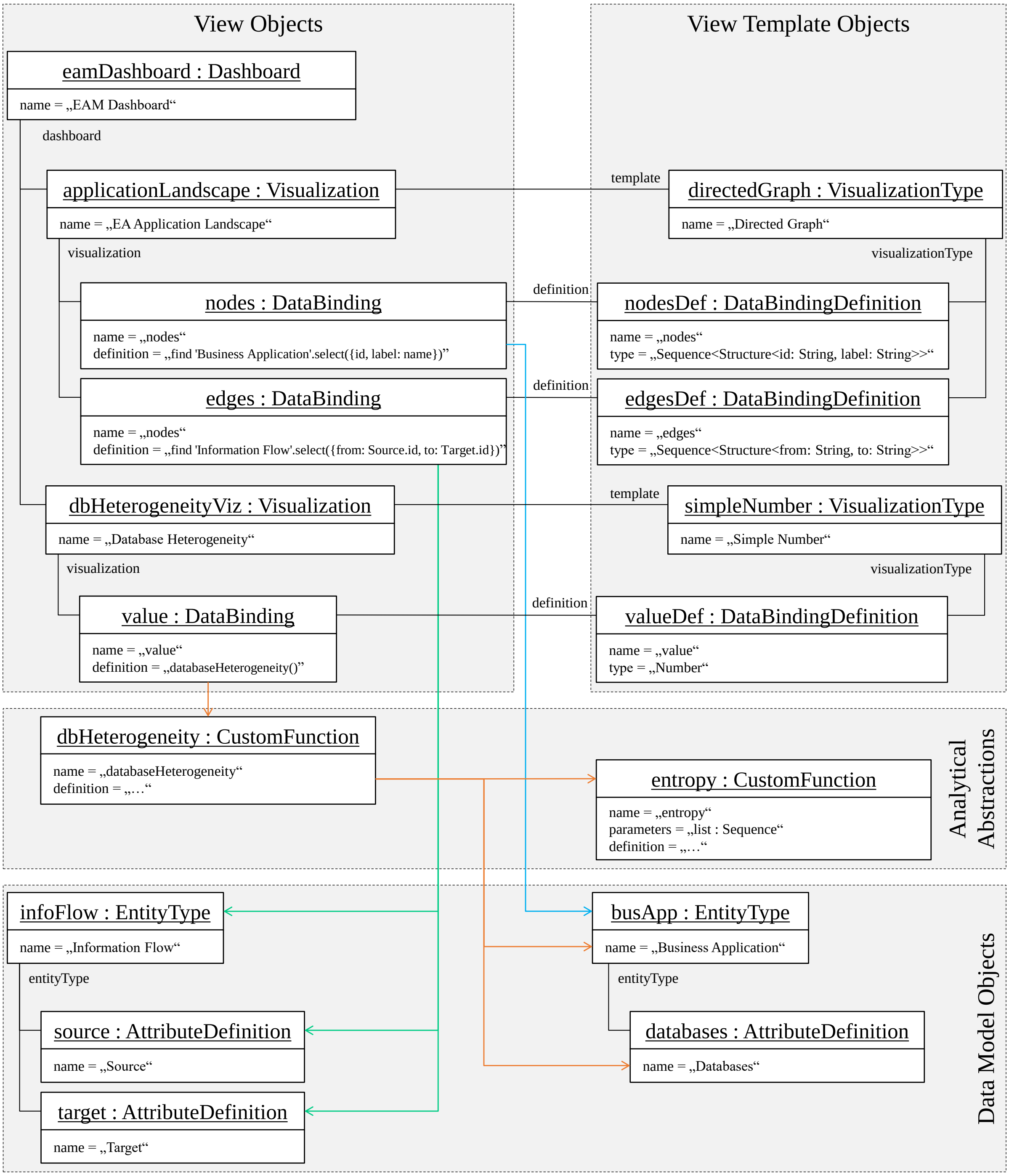

View Objects
View Template Objects
eamDashboard : Dashboard
name = „EAM Dashboard“
dashboard
applicationLandscape : Visualization
name = „EA Application Landscape“
template
directedGraph : VisualizationType
name = „Directed Graph“
visualization
visualizationType
definition
nodes : DataBinding
name = „nodes“
definition = „find 'Business Application'.select({id, label: name})”
nodesDef : DataBindingDefinition
name = „nodes“
type = „Sequence<Structure<id: String, label: String>>“
definition
edges : DataBinding
name = „nodes“
definition = „find 'Information Flow'.select({from: Source.id, to: Target.id})”
edgesDef : DataBindingDefinition
name = „edges“
type = „Sequence<Structure<from: String, to: String>>“
dbHeterogeneityViz : Visualization
name = „Database Heterogeneity“
template
simpleNumber : VisualizationType
name = „Simple Number“
visualization
visualizationType
definition
value : DataBinding
name = „value“
definition = „databaseHeterogeneity()”
valueDef : DataBindingDefinition
name = „value“
type = „Number“
dbHeterogeneity : CustomFunction
name = „databaseHeterogeneity“
definition = „…“
entropy : CustomFunction
name = „entropy“
parameters = „list : Sequence“
definition = „…“
Analytical Abstractions
infoFlow : EntityType
name = „Information Flow“
entityType
busApp : EntityType
name = „Business Application“
entityType
source : AttributeDefinition
name = „Source“
databases : AttributeDefinition
name = „Databases“
target : AttributeDefinition
name = „Target“
Data Model Objects

Figure 4.11.: A UML object diagram illustrating the analysis model for an exemplary *Dashboard* "EAM Dashboard" including semantic dependencies between the different artifacts (green, orange, and blue). The concepts are defined in Figures 4.3, 4.10, and 4.5 respectively. The exemplary data model is depicted in Figure 4.7.

existing web data visualization frameworks[2]. On the other hand, parsing the data retrieved by MxL queries to JSON objects is straight-forward, since various language features of MxL, e.g., the notation for complex objects in MxL, are inspired by JSON.

In summary, the expressiveness of views strongly depends on the technical platform in which they are implemented. However, particularly in a web-based environment which already provides a huge ecosystem of data visualization frameworks and libraries, we consider the expressiveness of views and view templates as described in the previous sections as sufficient for the visualization of complex linked data.

4.4. Meta-Analysis

As motivated in Section 3.2.4, creating awareness of important aspects and activities is to create awareness of important aspects and activities is essential in a collaboration environment (Martin and Sommerville, 2004). In the context of EUA, this includes the identification of co-workers with which a given person has to interact and coordinate in order to achieve a certain analytics-related task (Cataldo et al., 2006), e.g., the definition or adaption of a metric or visualization

As shown by the exemplary analysis model in Figure 4.11, the different elements can be highly connected to each other, i.e., changing one element of the analysis model can have an impact to multiple other elements. Establishing awareness also means that users know which elements and thus which other users are affected by different kinds of interactions with specific parts of the analysis model. Therefore, visualizing the analysis model as shown in Figure 4.11 is only a part of a possible solution: Maintaining and showing respective meta-information for each element (e.g., its owner) is also necessary to support users to identify co-workers affected by changes or other kinds of interactions (cf. requirement Req 20).

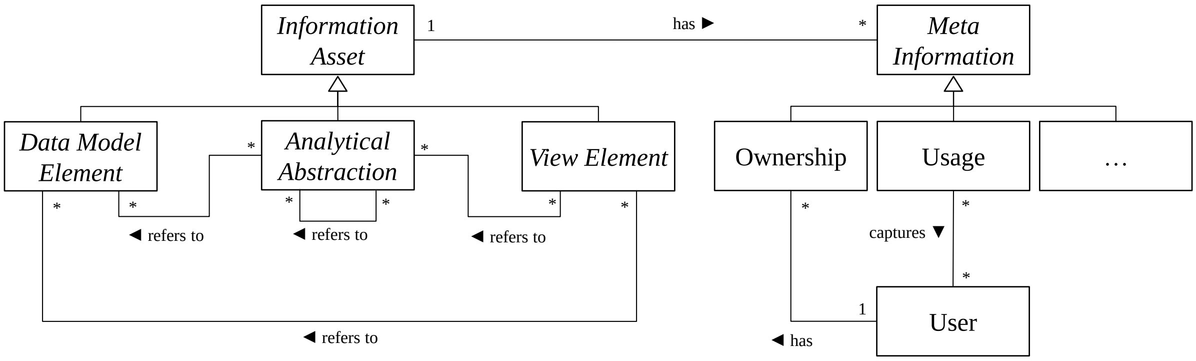


Figure 4.12.: A conceptual perspective on the artifacts of a EUA tool and their meta-data (Reschenhofer et al., 2016b).

Figure 4.12 shows a conceptual meta-model as a foundation for an analysis model (Reschenhofer et al., 2016b). In line with Khatri and Brown (2010), we subsume the basic EUA elements as described in Sections 4.1, 4.2, and 4.3 (namely data model elements, analytical abstractions, and

[2] `http://keshif.me/demo/VisTools`, last accessed on: 04.10.2016

view elements) under the general concept *Information Asset*. By their definition, *Information Assets* describe documented facts with a potential value for a stakeholder. Therefore, we consider *Data Model Elements*, *Analytical Abstractions*, and *View Elements* as *Information Assets* (Waltl et al., 2015). *Information Asset* can have an arbitrary number of *Meta Information* which represents descriptive information about its structure, context, or semantics (Dinter et al., 2015). While all types of *Information Assets* can have a common set of generic *Meta Information*, e.g., the owner of an *Information Asset* (Khatri and Brown, 2010), its users (Vaduva and Vetterli, 2001), its creators (Hüner et al., 2011), or its revision history (Ginige et al., 2010a), there are also specific types of *Meta Information* for specific *Information Assets*. For example, Frank et al. (2008) describe a meta-model for metrics (analytical abstractions) including different meta attributes, e.g., a goal, a purpose, or presumptions. Therefore, when exploring the analysis model, a respective exploration view has to handle each type of *Information Asset* individually and to show not only generic meta-information, but also specific one.

Information Asset are connected to each other through semantic dependencies which represents an additional kind of meta-information. Those dependencies are determined by *MxLReferences* and automatically extracted from MxL expressions (cf. Figure 4.5). By interpreting the semantic dependencies between views, analytical abstractions, and data model elements as information flows (if A refers to B, the information flows from B to A), we can represent the analysis model as an information flow graph. As a consequence, end-users are able to explore the analysis model via information flow analysis, which is a proven means to empower end-users to coordinate processes and facilitate collaboration (Durugbo et al., 2013). By augmenting the nodes of this information flow graph with meta-information (e.g., owners or users of the element represented by the node), we enable users to not only explore which elements of the analysis model are affected by certain interactions with one of its nodes, but also to identify co-workers which are related to them. In this sense, the information flow graph augmented with (user-related) meta-information enables artifact-centric social network analysis (Cross et al., 2002; Freeman, 2004; Kilduff and Brass, 2010). For this reason, we name it *Social Information Flow Graph* (SIFG).

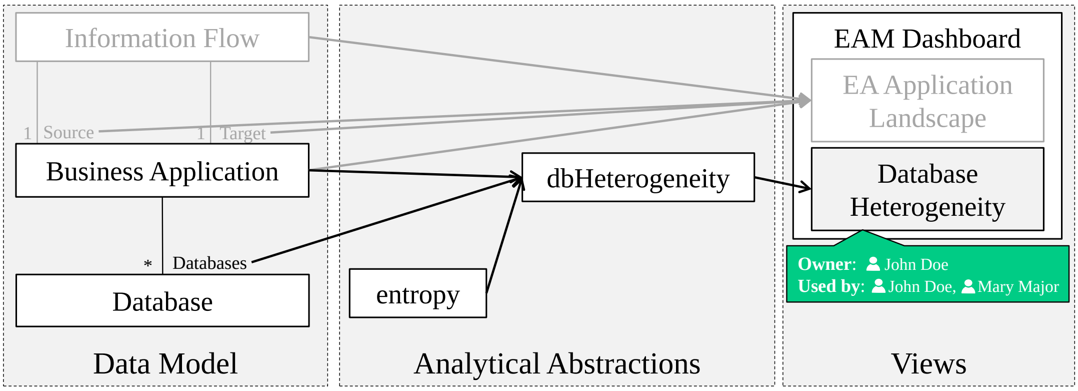


Figure 4.13.: The illustration of a Social Information Flow Graph (SIFG) for the example in Figure 4.11.

Figure 4.13 illustrates the SIFG for the example as described in Section 4.3.2. First and foremost, it shows how abstract information flows starting from the data model via analytical abstractions to views. However, it augments each node with respective meta-information, and relates them to users through different roles. For example, the owner of the "dbHeterogeneity" metric can observe which elements serve as input for the metric and which visualizations are actually using his metric, and which users are responsible for those elements. In this way, the SIFG enables new opportunities and use-cases for end-user-oriented social network analysis (Reschenhofer et al., 2016b).

CHAPTER 5

Prototypical Implementation and User Interface Design

While in Chapter 4 we described the conceptual design of a tool empowering end-users to collaboratively analyze evolving complex linked data, the following sections discuss the highlights of a respective prototypical implementation based on the *SocioCortex* platform. In contrast to the previous chapter, we will also discuss technical aspects, e.g., decisions regarding the selection of technical platforms and frameworks, and particularly decisions regarding the design of the User Interface.

While the conceptual design is based on the Hybrid Wiki meta-model (Matthes et al., 2011), our prototype builds on the technical implementation of a Hybrid Wiki system named *Tricia*. By extending this system as described in the following sections, we renamed it to SocioCortex (SC). Figure 5.1 illustrates the architecture of our prototype, and highlights logical components which were already provided by Tricia, and which were added in the context of this thesis.

Tricia already supported managing unstructured complex linked data as well as the collaborative and iterative design of respective data models. Furthermore, it provided additional useful features, e.g., access control, version history, support for file attachments, and full text search (cf. Section 2.1.2). SocioCortex and thus our prototype builds on those foundations, and extends it with the integration of analytical abstraction as described in Section 4.2.1. View and view Template elements (cf. Section 4.3.1) are not built into the SocioCortex back-end, but are implemented on the front-end side within the SC Visualizer client which is described in detail in Section 5.3.2.

As motivated in Section 3.1, multiple end-users with different roles and expertises interact with the system to explore or manage the data model, analytical abstractions, and views. In order to provide different UIs for those different use-cases, SocioCortex was implemented as a platform based on which different applications can consume its services through a well-defined technical interface, i.e., a Representational State Transfer (REST) Application Programming

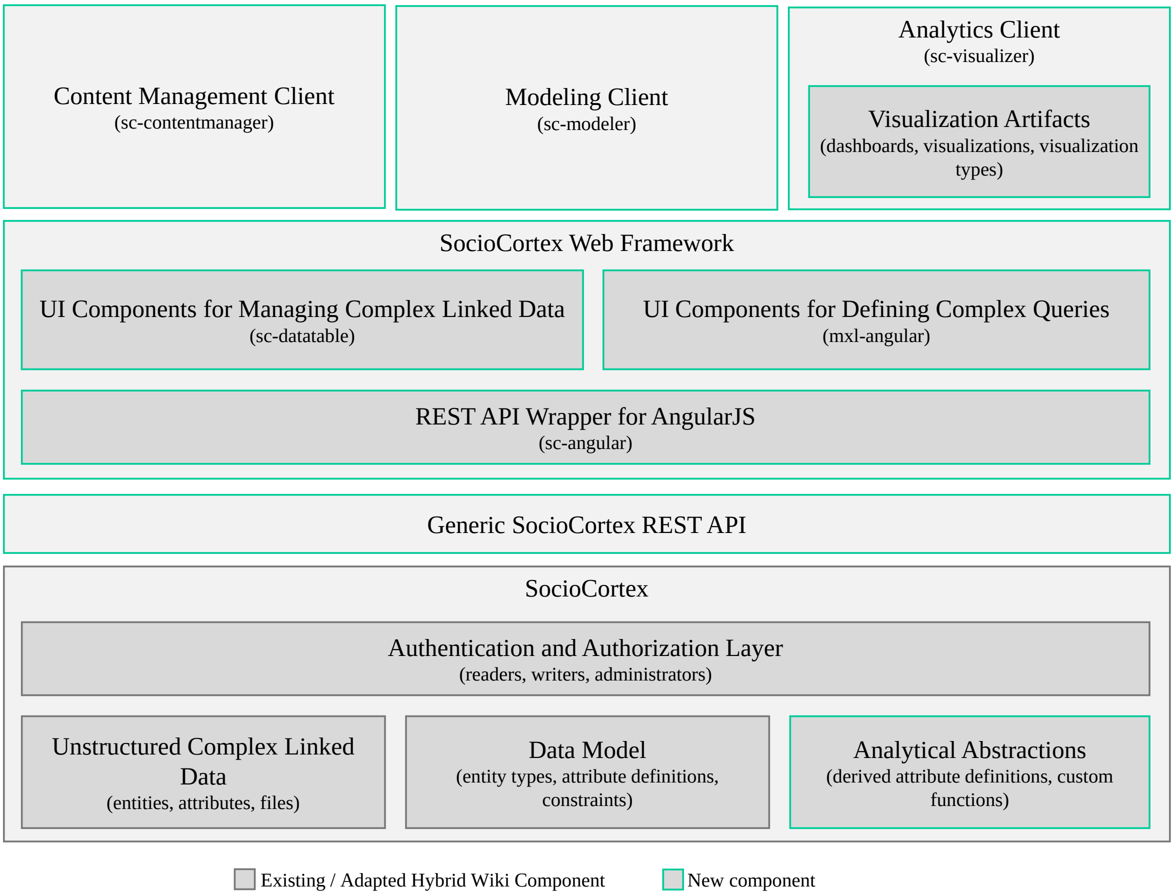

Content Management Client
(sc-contentmanager)
Modeling Client
(sc-modeler)
Analytics Client
(sc-visualizer)
Visualization Artifacts
(dashboards, visualizations, visualization types)
SocioCortex Web Framework
UI Components for Managing Complex Linked Data
(sc-datatable)
UI Components for Defining Complex Queries
(mxl-angular)
REST API Wrapper for AngularJS
(sc-angular)
Generic SocioCortex REST API
SocioCortex
Authentication and Authorization Layer
(readers, writers, administrators)
Unstructured Complex Linked Data
(entities, attributes, files)
Data Model
(entity types, attribute definitions, constraints)
Analytical Abstractions
(derived attribute definitions, custom functions)
Existing / Adapted Hybrid Wiki Component
New component

Figure 5.1.: The logical architecture of our prototype

Interface (API). Therefore, while Tricia was a standalone web application including both a back-end and a front-end, SocioCortex as a platform only refers to the extended back-end part. Section 5.1 outlines the most important features of the SocioCortex platform as well as its extensions compared to its predecessor Tricia.

While the core of the SocioCortex application was reduced to its back-end, the SC REST API enables the implementation of use-case-specific and light-weight front-ends for different kinds of platforms. As shown in Figure 5.1, our prototype encompasses three different AngularJS-based web clients which are described in detail in Section 5.3: A content management client to view and edit the content of the platform, a modeling client to enable end-users to define data models and analytical abstractions, and a visualization client empowering end-users to create dashboards and visualization types. Further examples for SC clients are a PDF report generator (Velten, 2016) and a legal workbench (Waltl et al., 2016).

Common SC-related functionality and UI components of those web clients are provided by a set of modules in order to avoid redundancies and to improve reusability. For example, the module *sc-angular* wraps the access to the SC REST API and thus provides a more convenient access to the services provides by SocioCortex. The *mxl-angular* and *sc-datatable* modules contain reusable UI components for defining MxL queries and for managing complex linked data respectively. These modules are further described in Section 5.2.

5.1. The SocioCortex Backend

SocioCortex is a Java-based Enterprise 2.0 platform and the successor of Tricia which was initially developed at the sebis chair at TUM. The most important improvements of SocioCortex compared to its predecessor Tricia are the following:

- Improvement of MxL (e.g., implementation of the type system as described in Section 4.2.2.1) and its integration in SocioCortex
- Replacement of the tightly-coupled web front-end by a technical REST API
- Integration of Adaptive Case Management (ACM) concepts. Since not relevant in the context of this thesis, we refer to the PhD thesis of Hauder (2015) for more details.

In Section 5.1.2, we will outline the implementation of MxL and its integration in SocioCortex. Subsequently, we describe the REST API in Section 5.1.3. However, first we will elaborate on the core components of SocioCortex.

5.1.1. The Core Components of SocioCortex

As illustrated in Figure 5.2, Tricia implemented the Model View Controller (MVC) architecture pattern (Büchner, 2007). When a client performs an HTTP request (1), the web server as a dispatcher forwards it to a handler (*controller*) based on the requested URL (2). The handler loads (and eventually manipulates) the model (3) and passes it to a view (4). The view is instantiated based on a view template (5) which is usually defined as an HTML document. The view is then returned to the web server (6) which creates a respective response for the client (7).

Since reduced to a back-end platform, the interaction model of SocioCortex (cf. Figure 5.3) skips the view generation step and instead parses the model object obtained by the handler into a machine-readable JSON object or JSON array. Thereby, the serialization process creates a specific JSON representation for each kind of model. In contrast to Tricia, in SocioCortex the actual view will be generated on the client based on the JSON data. The logical core components of SocioCortex are:

Web Server Two main tasks of the web server are processing authentication and mapping the incoming request to the corresponding handler.

Tricia implemented a session-based authentication mechanism. This means that based on an initial request containing the user's credentials, the server creates a session, validates the user credentials, assigns the user identity to the session, and returns a respective session id. By including this session id in each future request, the server can restore the session and thus determine the user's identity again.

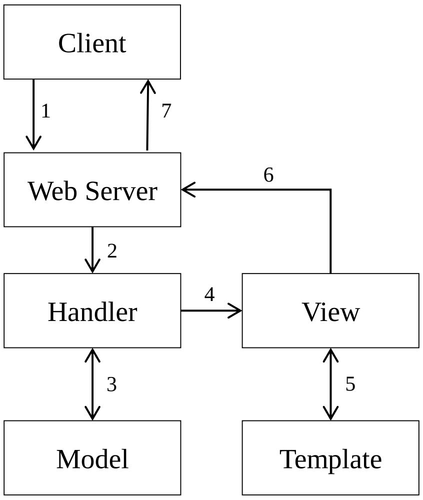


Figure 5.2.: Interaction model in Tricia.

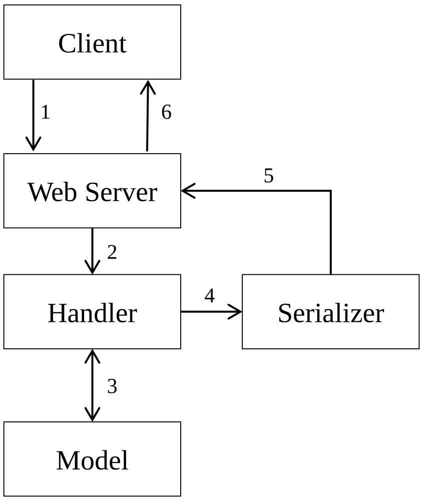


Figure 5.3.: Interaction model in SocioCortex.

In contrast to Tricia, SocioCortex communicates with its clients via a REST API. A defining feature of a REST API is its stateless character, i.e., each request to the web server is an independent transaction. As a consequence, the session-based authentication mechanism of Tricia is no longer suitable for SocioCortex as REST API-based platform. Therefore, we implemented new authentication mechanisms, namely *Basic Authentication* and *JSON Web Tokens*[1] (JWT). Both of them are stateless, i.e., each request to the web server contains all information required to authenticate the respective client without having to retain session information on the server across multiple requests.

While with Basic Authentication, the user credentials are Base64-encoded and embedded into the Hyper Text Transfer Protocol (HTTP) header of each request, JSON Web Tokens (JWT) is a token-based approach. This means that initially the client requests an authentication token by sending the user credentials. The server validates the credentials and creates an authentication token in JSON format—the JWT. A JWT contains a header, a payload, and a signature. The header usually consists of meta-information, e.g., the name of the algorithm used for signing the token. The payload captures the user's identity and potentially further properties (e.g., an expiration date). The signature is created by the web server using an RSA algorithm (Rivest et al., 1978) and ensures the integrity of the JWT. In all future requests, the JWT has to be included in the HTTP header so that the web server can validate the signature, check the token's expiration date, and authenticate the user.

After successfully completing the authentication step, the web server selects a handler which is responsible to process the incoming request based on its URL.

Handlers In a first step, a SocioCortex handler determines the HTTP method of the REST request (e.g., "GET", "POST", "PUT", or "DELETE") and checks if the respective method is implemented by the handler (e.g., "doGet", "doPost", "doPut", "doDelete"). After

[1] `https://jwt.io/`, last accessed on: 04.10.2016

that, the handler ensures that the authenticated user is authorized to perform the action as determined by the HTTP method.

If the authorization step is successful, the HTTP method-specific business logic is executed. Typical steps are the extraction of request parameters, the deserialization of the request payload, manipulation of the SocioCortex model, and the serialization of the response to the client. For example, a "PUT" request to a specific SocioCortex resource defines an update operation consisting of the following steps:

1. Extraction of the resource's ID from the request Uniform Resource Locator (URL)
2. Deserialization of the update instructions contained in the request payload
3. Update of the entity with the given ID and the given update instructions
4. Serialization of the updated entity

Deserialization and serialization are performed by resource-specific *Serializers*.

Model The SocioCortex Model layer captures all resource types which are persistable within SocioCortex's database. This includes the key concepts as defined by the extended Hybrid Wiki meta-model in Figure 4.1. For example, SocioCortex implements data model elements (*EntityType* and *AttributeDefinition*) and definitions of analytical abstractions (*CustomFunction* and *DerivedAttributeDefinition*) as built-in entities (Büchner, 2007) implemented by a *PersistentEntity* class (Büchner, 2007). Each *PersistentEntity* is mapped to a table in the underlying database. *PersistentEntities* are defined through properties of different types representing its attributes and relations. They are mapped to respective columns in the *PersistentEntity*'s database table. Furthermore, SocioCortex supports *Event Listeners* as functions which are automatically triggered on certain data-related events, e.g., changes of a *PersistentEntity*. This mechanism is used, e.g., for automatically maintaining the version history, i.e., if the instance of a *PersistentEntity* is changed, SocioCortex automatically creates a new entry in this instance's version history.

View and view template concepts are not mapped to respective built-in model entities in SocioCortex, but are modeled and implemented on the client (cf. Section 5.3.2) and stored in SocioCortex as ordinary entities. Since entities support access control and version history, those features inherently also apply to dashboards, visualizations, and visualization types.

Serializers While most components of Tricia were adapted in order to fit the needs of a REST API-based platform (e.g., implementation of stateless authentication mechanisms), the serializers are a new class of components which were added to SocioCortex. A serializer parses an object from its internal representation to a JSON representation which is compatible to the REST API definition (serialization), and vice versa (deserialization).

For each model class which has to be exposed via the REST API, there is a respective model-specific serializer class which provides two essential methods, namely "serialize" and "deserialize". By passing serialization options to those methods, the serialization and deserialization process can be configured, e.g., by including or excluding certain properties.

As described in Section 5.1.3, those serialization options can also be defined via the REST API.

5.1.2. Implementation of MxL

Another essential back-end component of SocioCortex is the Model-based Expression Language (MxL) which empowers end-users to formulate queries over the SC data model at run-time. A prior version of MxL was already implemented in Tricia (Reschenhofer, 2013). In the following section, we will outline highlights of the implementation of MxL on the one hand, and elaborate on the changes and improvements which were implemented in the context of this thesis on the other hand.

To evaluate MxL expressions, they have to be translated from a textual representation to executable instructions. This process is called *interpretation*, and consists of multiple consecutive steps, namely scanning, parsing, and type checking. Figure 5.4 illustrates the interpretation pipeline using an exemplary MxL query.

First, the textual representation of an MxL query is processed by the MxL scanner which is based on the open source lexical analyzer generator *JFlex*[2]. The scanner transforms a stream of characters into a stream of tokens, whereas a lexical grammar specifies which consecutive characters have to be bundled to one token. In this way, the scanner already determines numbers, operators, identifiers, keywords, and strings.

Based on a stream of tokens, the MxL parser builds the Abstract Syntax Tree (AST) in the next step. The MxL parser is generated by the LALR parser generator *Beaver*[3]. Analogous to the scanner, the MxL parser processes the tokens based on syntactical grammar rules in Extended Backus-Naur Form (Scowen, 1993). As shown in Figure 5.4, the AST represents a hierarchy of different kinds of expressions, e.g., *FunctionApplicationExpression*, or *IdentifierExpression*.

In the final step of the interpretation, the MxL type checker performs a static analysis of the AST and adds semantic annotations to its nodes. In the example in Figure 5.4, the type checker first observes the *FindByTypeExpression* and determines its type. For this purpose, the MxL type checker accesses the user-defined data model in SocioCortex. By providing a *Workspace* as query context, the type checker is able to find an *EntityType* with the name defined by the identifier of the *FindByTypeExpression*. Therefore, the type of the *FindByTypeExpression* is set to be *Sequence<Business Application>*. Knowing the input as well as the parameterized signature of the *average* function (cf. Table A.12), the type checker infers the parameter type (*Function<Business Application, Number>*). Since *Function points* is not defined as a global identifier, the parameter of the *average* function is automatically interpreted as an implicit lambda expression mapping each *Business Application* of the source sequence to the value of the respective *Function points* attribute. As a consequence, the type checker successfully finishes the semantic validation and returns a semantically annotated AST.

The annotations of the type checker serve as concrete instructions for the execution of the expression. For example, the identifiers *average* and *Function points* are resolved once at compile-time

[2] `http://jflex.de/`, last accessed on: 04.10.2016

[3] `http://beaver.sourceforge.net/`

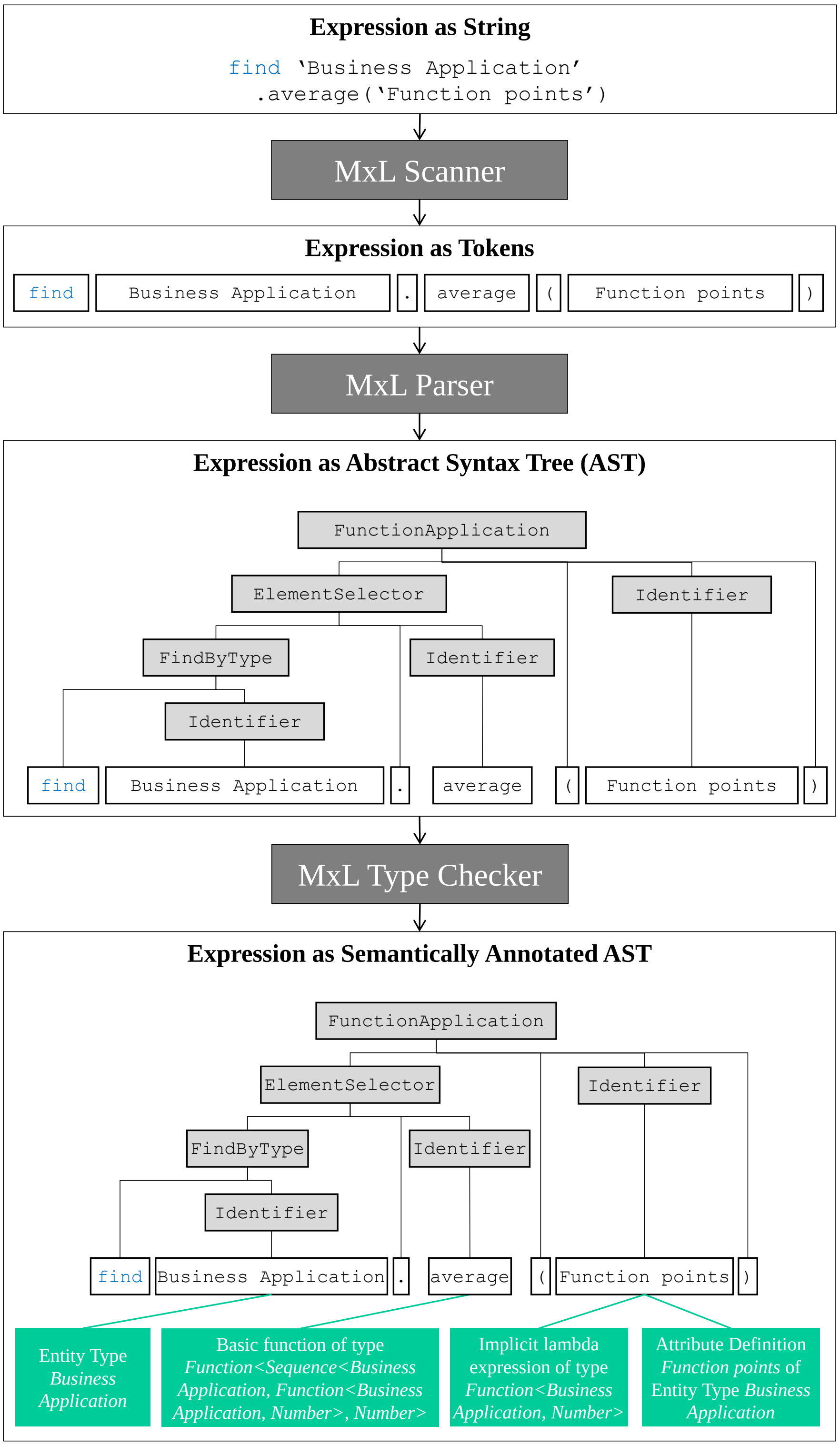

Expression as String
find 'Business Application'
.average('Function points')
MxL Scanner
Expression as Tokens
find
Business Application
.
average
(
Function points
)
MxL Parser
Expression as Abstract Syntax Tree (AST)
FunctionApplication
ElementSelector
Identifier
FindByType
Identifier
Identifier
find
Business Application
.
average
(
Function points
)
MxL Type Checker
Expression as Semantically Annotated AST
FunctionApplication
ElementSelector
Identifier
FindByType
Identifier
Identifier
find
Business Application
.
average
(
Function points
)
Entity Type *Business Application*
Basic function of type *Function<Sequence<Business Application, Function<Business Application, Number>, Number>*
Implicit lambda expression of type *Function<Business Application, Number>*
Attribute Definition *Function points* of Entity Type *Business Application*

Figure 5.4.: MxL interpretation pipeline illustrated by an exemplary MxL expression based on the data model in Figure 4.7.

and linked to the corresponding basic function or attribute definition respectively. When executing the query, the execution engine already knows how to process the individual expressions based on the type checker's annotations. For further details on the interpretation of MxL expressions we refer to the master's thesis by Reschenhofer (2013).

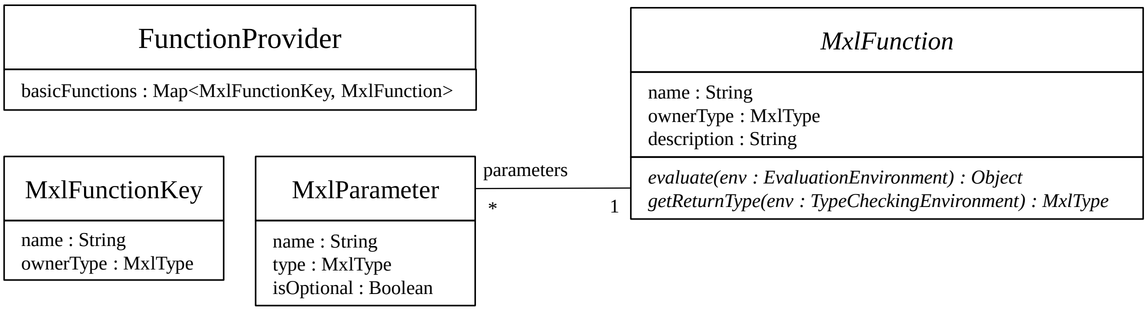


Figure 5.5.: An illustration of the concrete implementation of MxL's basic functions by using a combination of the *prototype* and *multiton* design patterns (Gamma et al., 1994; Edwin, 2014).

As described in Section 4.2.2.2, MxL implements a whole number of basic functions, especially functions which can be applied on sequences representing *Standard Query Operators* (Heijlsberg and Torgersen, 2013). Basic functions are implemented using a combination of the *prototype* (Gamma et al., 1994) and *multiton* (Edwin, 2014) design patterns, whereas the multiton pattern as a variation of the singleton pattern. In contrast to the singleton design pattern, the multiton refers to not only one static instance of a class, but to a map of multiple static instances. As illustrated in Figure 5.5, *MxlBasicFunction* represents the multiton class, whereas the *MxlFunctionProvider* as a singleton serves as an access point to instantiated basic functions. A basic function is referred to through its name and owner type, which at the same time form the attributes of the *MxlFunctionKey*. Consequently, a combination of a basic function's name and owner type has to be unique. *MxlFunctions* have two essential abstract methods—namely *evaluate* and *getReturnType*—which have to be implemented by the developer of concrete basic functions. The following example illustrates the implementation of the "split" method for objects of type *String* (indicated by the first parameter of *MxlFunction*'s constructor). It takes one string parameter specifying the delimiter of the operation:

```
new MxlFunction(BasicTypes.string(), "split",
    MxlParameter.get("delimiter", BasicTypes.string()),
    "Splits this string around matches of the given delimiter.") {

  @Override
  protected Object run(EvaluationEnvironment env) throws Exception {
    String str = env.getThisObject().toString();
    String delimiter = (String) env.getVariableValueNotNull("delimiter");

    return Lists.newArrayList(str.split(delimiter));
  }
```

```
    @Override
    protected MxlType getReturnType(TypeCheckingEnvironment env) {
        return BasicTypes.sequence(BasicTypes.string());
    }
}
```

Through the *EvaluationEnvironment* and *TypeCheckingEnvironment* parameters, developers can access relevant run-time and build-time information during the execution or type checking of the basic function respectively. In this example, the developer can determine the string object for which the basic function was invoked by using the *getThisObject* method of the *EvaluationEnvironment* parameter, and the "delimiter" parameter through its *getVariableValueNotNull* method. For inferring the return type of the function, the *TypeCheckingEnvironment* is irrelevant, since the result of the basic function is always a sequence of strings.

When the SocioCortex back-end application is initialized, the *MxlFunctionProvider* collects all defined *MxlFunctions* and registers them in its dictionary. Consequently, the system can efficiently determine a specific basic function through the *MxlFunctionProvider* and subsequently check its type or evaluate it by providing a respective *TypeCheckingEnvironment* or *EvaluationEnvironment* respectively. Developers can simply extend the set of basic functions by implementing a concrete *MxlFunction* and registering it in the *MxlFunctionProvider*. The extendability of basic functions directly addresses Requirement Req 7 as defined in Section 3.2.2.

MxL expressions are validated and evaluated under the identity of the currently authenticated user which implies that the type checker and evaluation engine only have access to data, data model elements, and analytical abstractions to which the user has at least read access to. As a consequence, a query might generate different results for different users, given they have a different access right profile. On the one hand, this allows the definition of personalized analytical abstractions and visualizations since the query only retrieves data which is accessible by the current user. On the other hand, this also leads to the fact that in the current prototype, users can not define analytical abstractions which yield the same result for each user. This issue is discussed again in Section 7.3.

When persisting MxL expressions as part of the definition of *DerivedAttributeDefinitions* and *CustomFunctions*, SocioCortex does not store the textual representation of an expression, but serializes the semantically annotated AST as generated by the type checker. On doing this, SocioCortex also persists *MxlReferences* (cf. Figure 5.6) as built-in model objects. For example, if defining the exemplary expression in Figure 5.4, SocioCortex will automatically create *MxlReference* objects for the *EntityType Business Application* and for the *AttributeDefinition Function points*.

Maintaining the *MxlReferences* as built-in model objects has two beneficial consequences: First, SocioCortex can generate a holistic MxL dependency graph in an efficient way. This dependency graph serves as input for the generation of the information flow graph as described in Section 4.4. Thereby, an *MxlReference* from an *MxlDefinable* to an *MxlReferable* is interpreted as an information flow from the *MxlReferable* to the *MxlDefinable*.

The second advantage of maintaining *MxlReferences* is that for each *MxlReferable*, the system can efficiently determine all *MxlDefinables* which are referring to it. This information is helpful to estimate the impact of changes of an *MxlReferable* on the one hand, or to keep affected *MxlDefinables* consistent in case of actual changes of an *MxlReferable*. In general, the system tries to refactor *MxLDefinables* automatically in order to keep them semantically consist. If this is not possible, its owner is asked to do it manually. In the following, we describe the consequences of different kinds of changes of *MxlReferables* as implemented in the prototype of today:

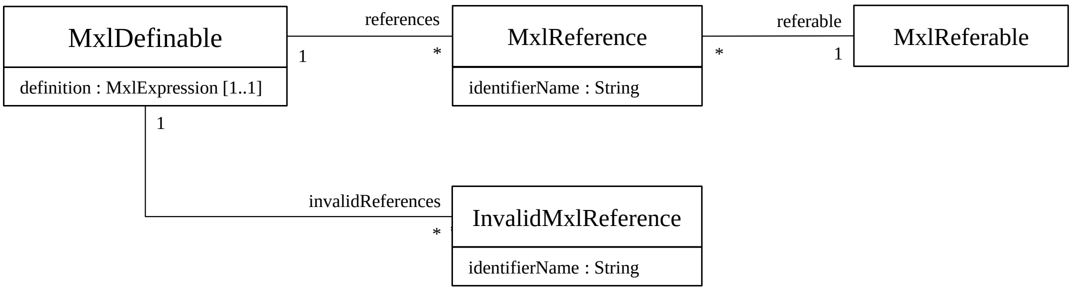


Figure 5.6.: An illustration of the concrete implementation of *MxlReferences* based on the conceptual model in Figure 4.5.

Change of an *MxlReferable*'s name Changes of names of *MxlReferables* can be handled automatically by the system. For this purpose, it simply replaces the name of the identifier in each affected *MxlDefinable* with the new name of the changing element. Since the identifier name does not affect the type of MxL expressions, they do not even have to be revalidated by the MxL type checker.

Change of an *MxlReferable*'s type (or multiplicity) If the type of an *MxlReferable* changes, all *MxlDefinables* have to be revalidated by the MxL type checker. If the revalidation fails, the respective *MxlReferences* are turned into *InvalidMxlReferences* (cf. Figure 5.6) which in turn serves as an indication for the owner of the *MxlDefinable* that it has to be manually adapted. The change of an *MxlReferable*'s type not only triggers the revalidation of *MxlDefinables* which are directly referencing them, but also of transitively dependent *MxlDefinables*.

Creation of an *MxlReferable* When creating an *MxlReferable* with a specific name, the system checks if there are any corresponding *InvalidMxlDefinables*, and triggers the revalidation of respective *MxlDefinables*. If the revalidation fails, they keep their *invalid* marker. Otherwise, the system considers the *MxlDefinable* as valid and—if it is an *MxlReferable* at the same time—triggers the revalidation of incoming *MxlReferences*.

Deletion of an *MxlReferable* When deleting an *MxlReferable* with at least one incoming *MxlReference*, the system provides three options to the user:

1. Mark affected *MxlDefinables* as *invalid*
2. If applicable, replace the identifier of the deleted *MxlReferable* with its current value.

This option is only valid for *MxlDefinables* representing data objects, e.g., wiki pages containing an embedded expression (Reschenhofer et al., 2014a)

3. Perform a cascaded deletion, i.e., delete all (transitively) affected *MxlDefinables*

Deleting an *MxlReferable* with no incoming *MxlReferences* has no consequences.

If the system would enable end-users to define subtype relationships between entity types, the refactoring of *MxlDefinables* would become significantly more difficult. For example, when renaming an *AttributeDefinition* which overrides a corresponding attribute of the super type, it has to be decided if *MxlDefinables* refer to the renamed *AttributeDefinition* which was originally defined as *MxlReferable*, or to the *AttributeDefinition* of the super type whose name still matches the original identifier name within the *MxlDefinable*'s expression. The implementation of subtype relationships also has analogous effects on handling the creation or deletion of *MxlReferables*.

5.1.3. SocioCortex REST API

In SocioCortex, clients communicate with the back-end via a well-defined REST API. A corresponding REST API description defines which REST services are actually provided by SocioCortex[4].

In total, SocioCortex provides the following four types of REST API operations:

Create-Read-Update-Delete (CRUD) Operations The key concepts of the extended Hybrid Wiki meta-model in Figure 4.1 are implemented as REST resources and thus accessible and manipulable by applying an HTTP method to a corresponding URL. Thereby, the HTTP methods "POST", "GET", "PUT", and "DELETE" are used to create, read, update, and delete (CRUD) resources. For example, a "GET" request to "/workspaces" retrieves all workspaces to which an authenticated user has at least read access to. A "DELETE" request to "/workspaces/{id}" deletes the workspace with the specified ID. Figure 5.7 shows the resource model of the SC REST API summarizing the relevant resources for in the context of this thesis, as well as their properties and relations to each other.

MxL Operations In addition to the CRUD-based manipulation of SocioCortex's key resources, the platform also provides services to analyze and evaluate MxL expressions in a particular context, whereas the context can be a workspace, an entity type, or an entity. For example, a "POST" request to "/workspaces/{id}/mxlValidation" with an MxL expression in its payload triggers the static analysis of the query in the context of the specified workspace, e.g.:

```
{
  "expression": "find 'Business Application'.average('Function points')"
}
```

[4] http://sociocortex.com/documentation/, last accessed on: 04.10.2016

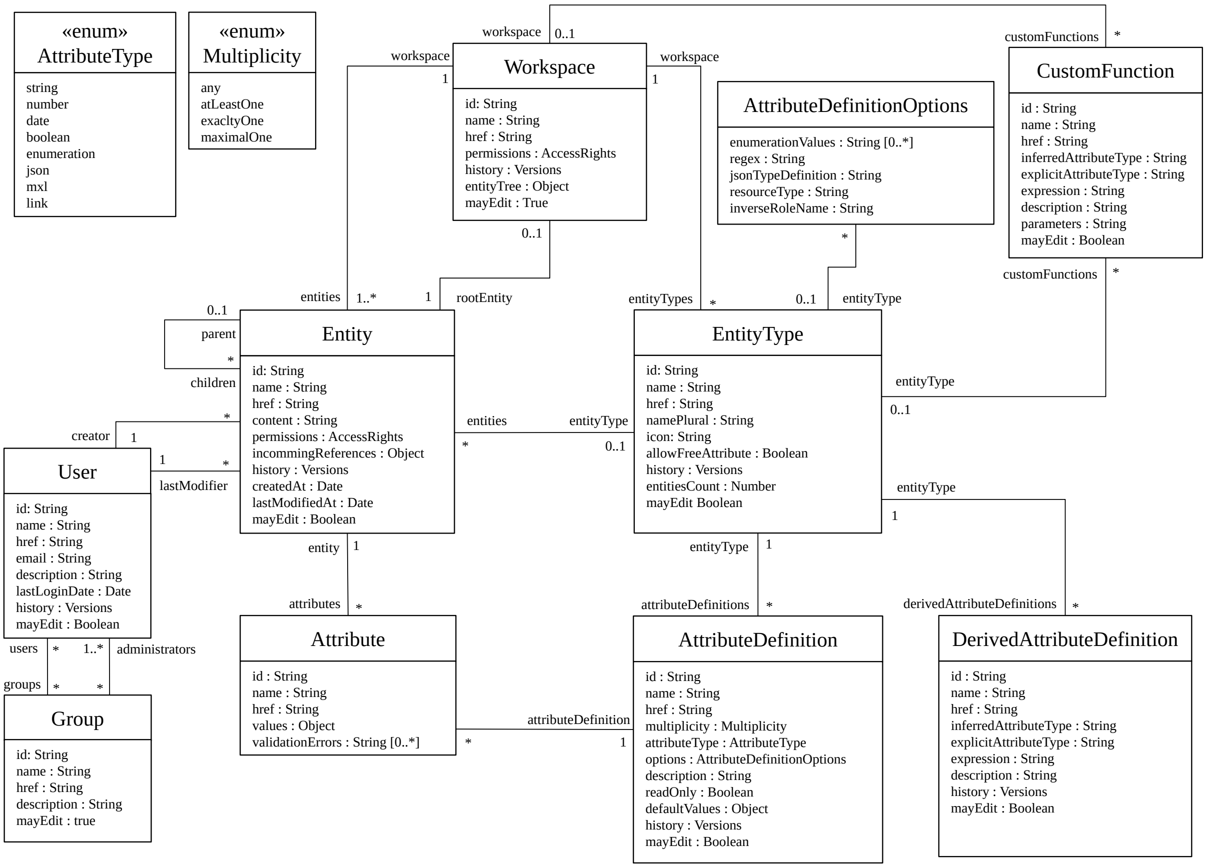


Figure 5.7.: A model of the resources exposed via the SC REST API.

In addition to the "expression" property, the payload can optionally define an "expectedType". With this information, the type checker is able to interpret certain expressions differently, e.g., as implicit lambdas. For example, the semantic analysis of the following query would fail because the identifier is unknown:

```
{
  "expression": "'Function points'"
}
```

However, by adding the "expectedType", the type checker will interpret the expression as implicit lambda, i.e., the identifier is successfully interpreted as the attribute of the implicit lambda's parameter of type *Business Application* (cf. Figure 4.7):

```
{
  "expression": "'Function points'",
  "expectedType": "Function<Business Application, Number>"
}
```

The result of the "mxlValidation" request contains a semantically annotated AST as well as a dependency graph representation including all elements the query is transitively depending on. With the same request to "/workspaces/{id}/mxlQuery", SocioCortex is returning the result of the query's evaluation instead of its static analysis.

A third type of MxL operation (GET "/workspaces/{id}/autoCompletionHints") returns a list of MxL identifiers which are available in the specified context, e.g., names of workspace-specific entity types and their attribute definitions. This operation is particularly helpful to support the user in formulating queries, e.g., by providing auto-completion in textual query editors (cf. Section 5.2.3) or a drop-down list in visual query editors.

Search Operations One of the main features of SocioCortex and its predecessor Tricia is its support for full text search with facets. Thereby, a search can be refined by filtering the hits by structural information. This service is exposed via the REST API and accessible through two operations—"/searchHints" and "/searchResults"—which accept the same set of filter parameters, but return the search result in a different level of detail. The former only returns basic information including the search hit's name and a disambiguation string, while the latter includes a parameterizable set of resource-specific attributes and meta-information.

Authentication Operations As described in Section 5.1.1, SocioCortex supports JSON Web Tokens (JWT) as stateless authorization mechanism. For this purpose, the SC REST API provides an operation ("/jwt") to request an authentication token for given credentials.

When requesting resources via the REST API, the set of attributes and meta-information included in the response depends on the actual request type. By default, requesting a list of resources (e.g., "/workspaces") only returns minimal objects including the resource's ID, name, and URL, whereas requesting a single resource (e.g., "/entities/id") returns all of its attributes and meta-information. Through query string parameters, the attributes and meta-information to be included in the response can be explicitly specified (e.g., "/entities/id?attributes=Function points&meta=workspace"). Those parameters are available for all operations returning resources.

5.2. SocioCortex Frameworks and Libraries

By providing a REST API, SocioCortex allows other applications to use and consume its services. Since building on standard data formats (JSON) and protocols (HTTP), different kinds of applications running on different kinds of platforms can act as clients for SocioCortex.

As illustrated in Figure 5.1, the three SocioCortex clients relevant in the context of this thesis are web clients. They use the same abstraction layer which wraps the access to the REST API and provides a convenient access to the services of SocioCortex. Furthermore, those web clients also share specific UI components which are defined as part of reusable frameworks. The following sections describe those web-based frameworks.

5.2.1. sc-angular: An AngularJS-based Wrapper for the SC REST API

All three web clients illustrated in Figure 5.1 are based on the JavaScript MVC framework *AngularJS*[5]. Consequently, we named the AngularJS-based framework for accessing the SC REST API *sc-angular*[6].

Key features of AngularJS (version 1.x) are the support for *two-way data binding* as well as a broad ecosystem including libraries, frameworks, and respective documentations. For example, there are AngularJS modules for consuming REST APIs which are reused. On the one hand, reusing existing libraries reduces the development effort. And on the other hand, this ensures that the *sc-angular* framework complies to standards and principles which are implemented by current AngularJS frameworks and that are already familiar to current AngularJS developers.

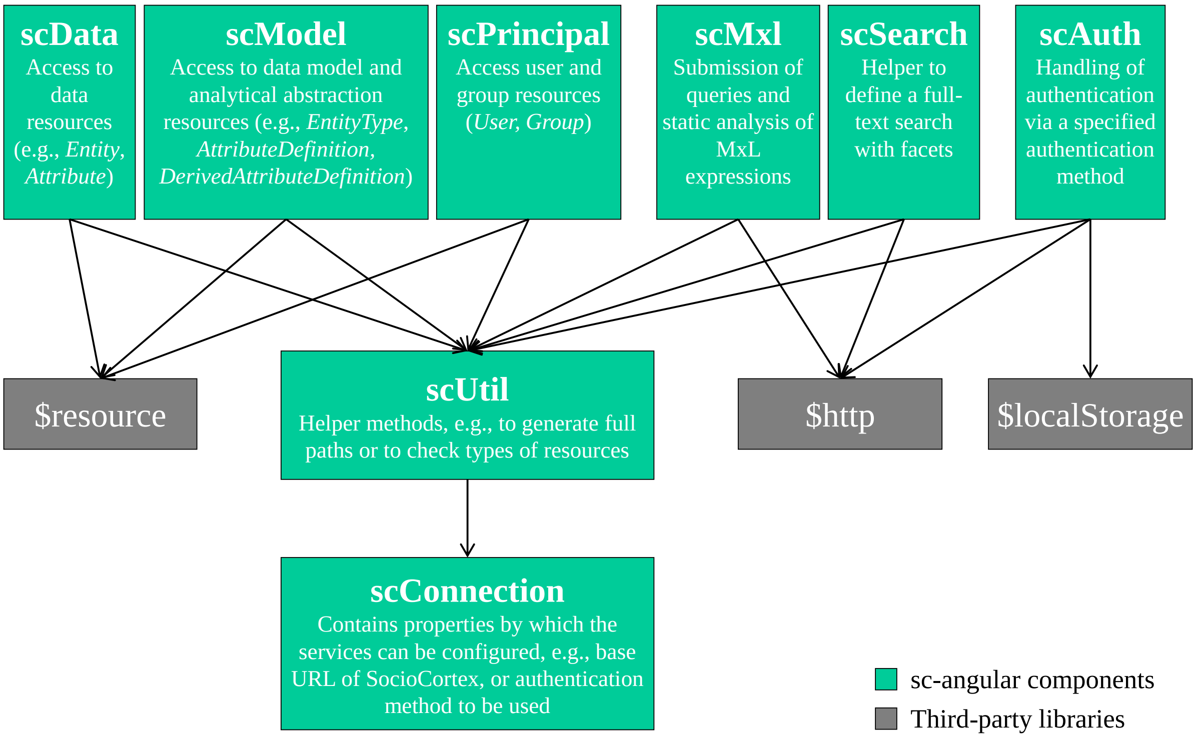

scData
Access to data resources (e.g., *Entity*, *Attribute*)
scModel
Access to data model and analytical abstraction resources (e.g., *EntityType*, *AttributeDefinition*, *DerivedAttributeDefinition*)
scPrincipal
Access user and group resources (*User*, *Group*)
scMxl
Submission of queries and static analysis of MxL expressions
scSearch
Helper to define a full-text search with facets
scAuth
Handling of authentication via a specified authentication method
$resource
scUtil
Helper methods, e.g., to generate full paths or to check types of resources
$http
$localStorage
scConnection
Contains properties by which the services can be configured, e.g., base URL of SocioCortex, or authentication method to be used
🟩 sc-angular components
▨ Third-party libraries

Figure 5.8.: The components of the sc-angular framework and its dependencies to each other.

The sc-angular framework represents a collection of AngularJS services which fulfill different purposes, e.g., providing wrappers for the REST API operations, and dealing with authentication against SocioCortex. Figure 5.8 provides an overview over those services and illustrates dependencies between them.

The *scConnection* component represents an AngularJS value and has three properties which are configurable by the user: The base URL of the SocioCortex back-end, the API version, and

[5] `https://angularjs.org/`, last accessed on: 04.10.2016
[6] `https://github.com/sebischair/sc-modeler`, last accessed on: 04.10.2016

the authentication method to be used (options are "basic" and "jwt", cf. Section 5.1.1). For example, the *scConnection* value can be configured as follows:

```
angular.module('someSocioCortexAngularApp', ['sociocortex'])
  .value('scConnection', {
    baseUri: 'https://server.sociocortex.com',
    apiVersion: 'v1',
    authenticationMethod: 'jwt'
  });
```

In this example, we create a new AngularJS app named "someSocioCortexAngularApp", with "sociocortex" representing the sc-angular module as its only dependency. By using the *value* method of the AngularJS module, we can define a new value for *scConnection* and overwrite the default settings.

Based on those configurable constant properties, the *scUtil* service provides helper methods, e.g., to generate full URLs based on relative ones. Furthermore, the *scUtil* service also defines a constant containing the URLs for the resource types of SocioCortex.

The main purpose of sc-angular is to provide a convenient way to retrieve and manipulate the REST API's resources, and to abstract technical details like URL paths and patterns. This functionality is divided into the three services *scData*, *scModel*, and *scPrincipal* which abstract the access to the data elements, data model elements, and users and groups respectively. For implementing those three services, we use the *$resource* service of the *ngResource* module[7] which provides interaction support for REST APIs. More specifically, $resource allows the definition of resource objects through which developers can apply CRUD operations. By default, a resource object already defines methods to create, delete, and read a single resource and to retrieve a list of resources. The update operation can be defined as shown by the following example:

```
var Entity = $resource(scUtil.getFullUrl("entities/:id"),
  {
    update: {
      method: "PUT"
    },
    queryByEntityType: {
      method: "GET",
      url: scUtil.getFullUrl("entityTypes/:id/entities"),
      isArray: true
    }
  });
```

The $resource service takes a URL pattern as its parameter, whereas the full URL is generated by *scUtil*'s helper method *getFullUrl*. Based on this URL pattern, ngResource is able to derive the configurations for HTTP requests to create, delete, and read REST resources. Additional operations (e.g., "update" to edit an existing entity, or "queryByEntityType" to retrieve all enti-

[7] `https://docs.angularjs.org/api/ngResource`, last accessed on: 04.10.2016

ties for a given entity type) can be defined by providing an explicit HTTP request configuration as shown in the example.

scMxl and *scSearch* provide methods to perform MxL and search operations respectively. Since the corresponding REST API operations are not about the CRUD-based manipulation of REST resources, *scMxl* and *scSearch* do not build on the ngResource service, but use the default *$http* service of AngularJS. *scMxl* provides the methods *query*, *validate*, and *autoComplete* which map to corresponding REST API operations (cf. Section 5.1.3). Analogously, *scSearch* defines two functions to abstract the access to the "searchHints" and "searchResults" REST API operations as defined in Section 5.1.3.

The *scAuth* service is an abstraction layer to handle authentication and to store the authenticated user identity on the client-side. It implements *login* method which accepts the user credentials as a pair of user name and password. Based on the selected authentication method, it either generates a corresponding *Basic Authentication* field or requests a JSON Web Tokens (JWT) from the SocioCortex back-end using the "/jwt" REST API operation. *scAuth* ensures that each request submitted by the sc-angular framework either adds the generated *Basic Authentication* field or the JWT to its header. Furthermore, the *scAuth* service also stores the *Basic Authentication* field or the JWT in the browser's local storage using the $localStorage service of AngularJS. In this way, the authentication state is retained on the client when closing and reopening the web client. Of course, if the JWT expires, the user has to login again in order to retrieve a fresh authentication token. By invoking the *logout* method of the *scAuth* service, the authentication state can be deleted explicitly.

5.2.2. sc-datatable: A UI Component for Managing Complex Linked Data

As described in Section 2.2.1, one of the characterizing features and success factors of spreadsheets is its grid UI which enables end-users to intuitively view, manipulate, and analyze data without having to explicitly define a data structure. On the other hand, the implicit and hidden spreadsheet design is also one of the main sources for errors (Hermans, 2012).

In contrast to spreadsheets, in our prototype the data model is explicitly designed by data modelers (cf. Section 3.1). Based on this data models, we can support users in maintaining the data in a consistent and structured way. In order to preserve flexibility, the Hybrid Wiki approach allows users to deviate from the defined data structures when entering data into the system. However, the user is explicitly notified about inconsistencies, e.g., by visually highlighting them.

In an effort to combine the intuitive UI of the spreadsheet grid on the one hand, and the explicit approach to data modeling as implemented in SocioCortex on the other hand, we developed the *sc-datatable*[8]. This data table empowers the user to edit SocioCortex's data in a spreadsheet-like way, e.g., by using the keyboard to navigate through the cells, quickly editing one or multiple cell values, and copying and pasting cell ranges. In order to ensure its reusability in multiple web clients, sc-datatable was designed as reusable AngularJS component which primarily consists of a respective AngularJS directive.

[8] `https://github.com/sebischair/sc-datatable`, last accessed on: 04.10.2016

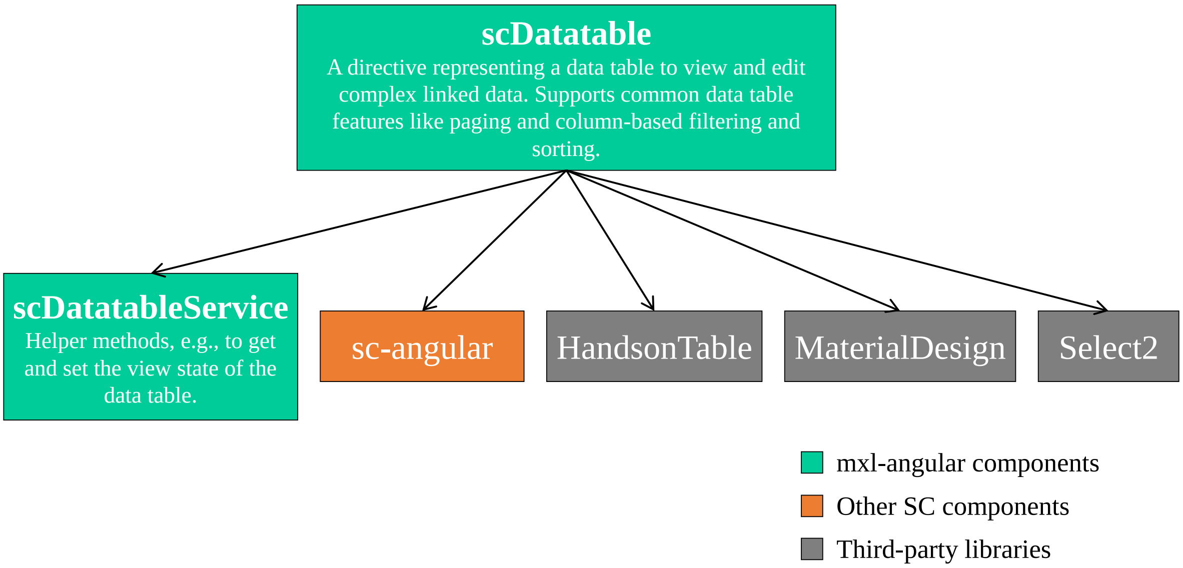

scDatatable
A directive representing a data table to view and edit complex linked data. Supports common data table features like paging and column-based filtering and sorting.
scDatatableService
Helper methods, e.g., to get and set the view state of the data table.
sc-angular
HandsonTable
MaterialDesign
Select2
mxl-angular components
Other SC components
Third-party libraries

Figure 5.9.: The components of the sc-datatable library and its dependencies to each other.

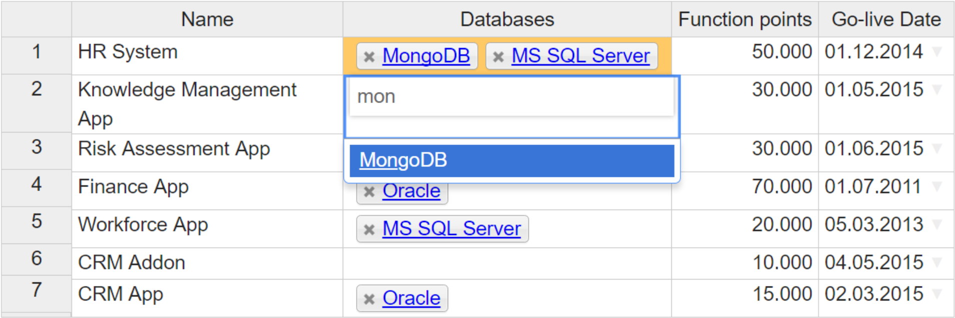
Figure 5.9 shows the logical sub components of sc-datatable library. The *scDatatable* directive is the most important one, and represents the actual data grid. In general, the data table shows the entities for a specified entity type, i.e., the table's columns represent the attribute definitions, and its rows the entities of the corresponding entity type. Consequently, each cell of the data table refers to a single attribute of an entity. The data is obtained from SocioCortex via the sc-angular framework.

The data table supports common data view features, e.g., paging, column selection, and filtering and sorting by column values. The data table directive is implemented by using the open source JavaScript spreadsheet library *Handsontable*[9] for its basic structure, the UI framework *AngularMaterial*[10] for basic controls, and the UI library *Select2*[11] for multi-valued cell controls.

In order to interact with the host application in which the data table is embedded, sc-datatable implements the AngularJS service *scDatatableService*. This service provides methods to, e.g., get and set the view state of the data table. The view state includes the data tables filter and sort configuration, set of selected columns including their width, and current paging size and number. Getting and setting the view state enables the host application to store and persist the current view state, e.g., in the user's profile and to restore a user-specific view as soon as the user revisits the page containing the table.

Supporting complex linked data as formally defined by the meta-model in Figure 4.3 implies that the data table provides means to display and edit multi-valued relations and attributes of primitive and complex data types within its cells. Figures 5.10 demonstrates how an input field for a multi-valued relation looks like. Thereby, multiple values per cell are enabled by *Select2*'s

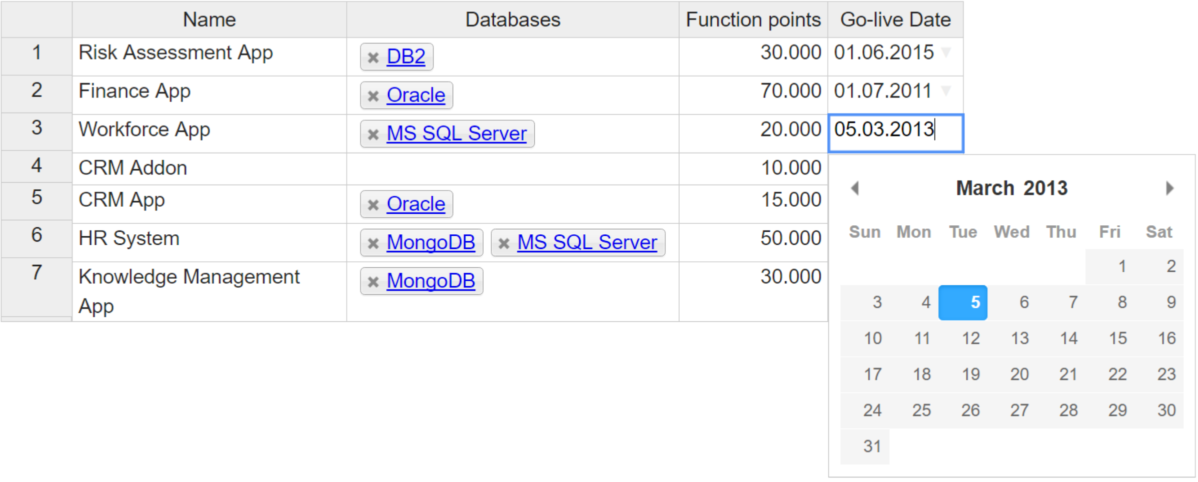
[9] `https://handsontable.com/`, last accessed on: 04.10.2016
[10] `https://material.angularjs.org`, last accessed on: 04.10.2016
[11] `https://select2.github.io/`, last accessed on: 04.10.2016



	Name	Databases	Function points	Go-live Date
1	HR System	× MongoDB × MS SQL Server	50.000	01.12.2014
2	Knowledge Management App	mon MongoDB	30.000	01.05.2015
3	Risk Assessment App		30.000	01.06.2015
4	Finance App	× Oracle	70.000	01.07.2011
5	Workforce App	× MS SQL Server	20.000	05.03.2013
6	CRM Addon		10.000	04.05.2015
7	CRM App	× Oracle	15.000	02.03.2015

Figure 5.10.: The sc-datatable supports complex linked data, i.e., multi-valued attributes and relations as well as composite attribute.

multi-value select boxes. For relations, an auto-completion mechanism supports the user in selecting an entity which can be referred to based on the attribute definition's type constraint. Within the cell, a relation is rendered as a hyperlink.



	Name	Databases	Function points	Go-live Date
1	Risk Assessment App	× DB2	30.000	01.06.2015
2	Finance App	× Oracle	70.000	01.07.2011
3	Workforce App	× MS SQL Server	20.000	05.03.2013
4	CRM Addon		10.000	
5	CRM App	× Oracle	15.000	
6	HR System	× MongoDB × MS SQL Server	50.000	
7	Knowledge Management App	× MongoDB	30.000	

◂ March 2013 ▸

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Figure 5.11.: The sc-datatable supports all types captured by the extended Hybrid Wiki meta-model in Figure 4.3.

Figure 5.11 shows another data type-specific input field, namely a calendar for cells containing a date. Furthermore, the data table renders a drop-down list for enumeration values, a check box for a boolean value, a textual input field with respective validations for strings and numbers, and a JSON editor for composite objects. Within a cell, those values are transformed into respective textual representations.

As a consequence from rendering data either to a textual representation within a cell, or into an input field to edit the value, the data table has to transform its data from an internal canonical representation to a UI-specific one, e.g., to a type-specific input field representation. Table 5.12 illustrates how data is transformed from one representation to another in the context of sc-datatable. Based on the input gained from sc-angular, the sc-datatable first transforms the data into an internal canonical data representation, based on which further transformations follow:

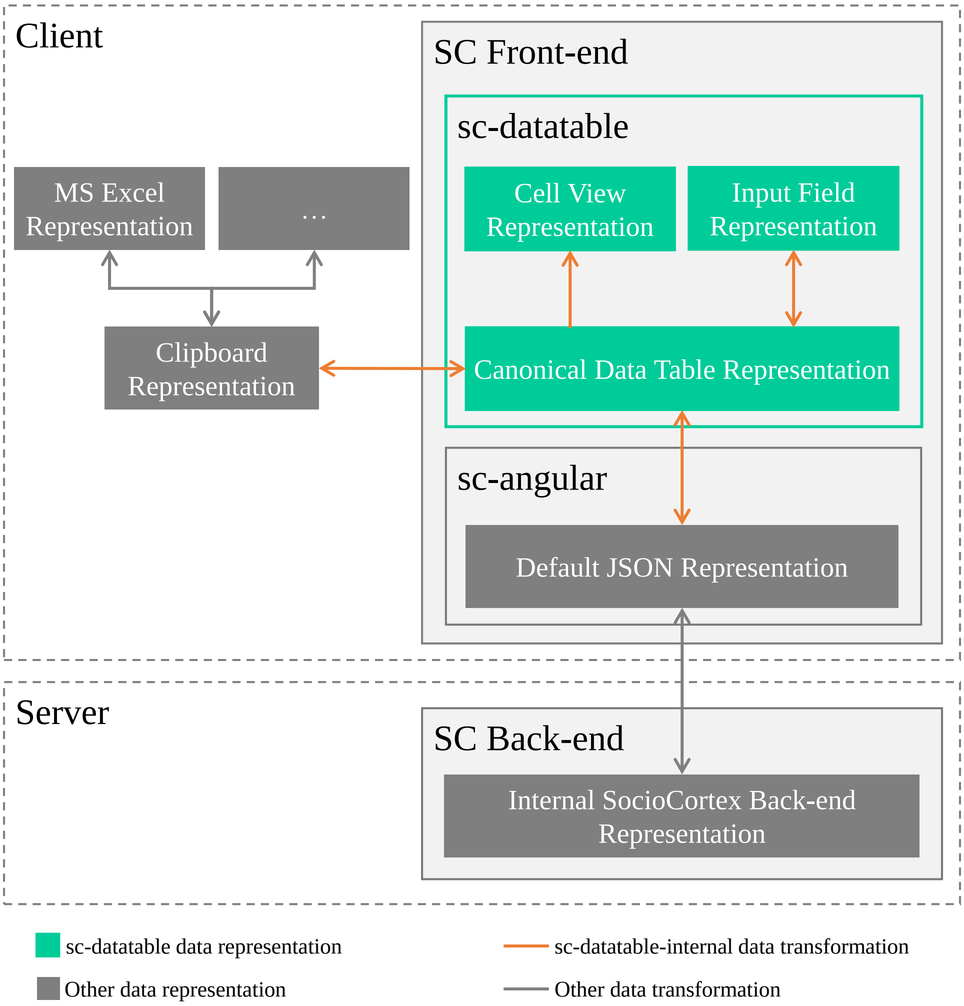

Client
SC Front-end
sc-datatable
MS Excel Representation
…
Cell View Representation
Input Field Representation
Clipboard Representation
Canonical Data Table Representation
sc-angular
Default JSON Representation
Server
SC Back-end
Internal SocioCortex Back-end Representation
sc-datatable data representation
Other data representation
sc-datatable-internal data transformation
Other data transformation

Figure 5.12.: The different representations of data and data models, and how they are transformed.

Transformation to Cell View Representation This transformation generates the Hyper Text Markup Language (HTML) markup which renders the value within a cell. While strings, numbers, and dates are transformed into a simple textual representation, relations are transformed to HTML hyperlinks. This is a one-way transformation, i.e., data cannot be transformed from the cell view representation to the canonical one.

Transformation to Input Field Representation This transformation generates the data type-specific input field as demonstrated in Figures 5.10 and 5.11. This is a two-way transformation, i.e., existing values are transformed from their canonical representations into the input field representation, and if the value is changed, it is transformed back to its canonical representation.

Transformation to Clipboard Representation Single cells and cell ranges can be copied into the client's clipboard and, e.g., pasted into a MS Excel worksheet. For this, the data table defines a transformation from its canonical representation to a data format which is compatible with common spreadsheet tools, e.g., MS Excel. This is a two-way transformation, i.e., cells and cell ranges can also be copied from a MS Excel worksheet and pasted into the sc-datatable component. This transformation addresses particularly the interoperability feature as motivated in Section 2.2.1.

If a user is editing the value of a cell, this cell is visually highlighted as shown in Figure 5.10. This means that editing cell values only change the client state. The propagation of the client state to the server has to be triggered manually by clicking on a save button. Alternatively, the client state can be reset and overwritten with the server state, i.e., all changes can be reverted.

Technically, the data table is implemented as a reusable AngularJS directive and can be embedded in any AngularJS application as follows:

```
<sc-datatable entity-type-id="{{entityTypeId}}"></sc-datatable>
```

The only mandatory parameter of this directive is the ID of the entity type whose entities should be shown in the data table.

5.2.3. mxl-angular: A UI Component for Defining Complex Queries

An essential feature of an End-User Analytics (EUA) tool is the empowerment of end-users to define ad-hoc queries and calculations (Tamm et al., 2013; Hermans, 2012). For this purpose, the tool has to provide a respective UI supporting the end-user to explore queryable elements in order to define consistent expressions based on them.

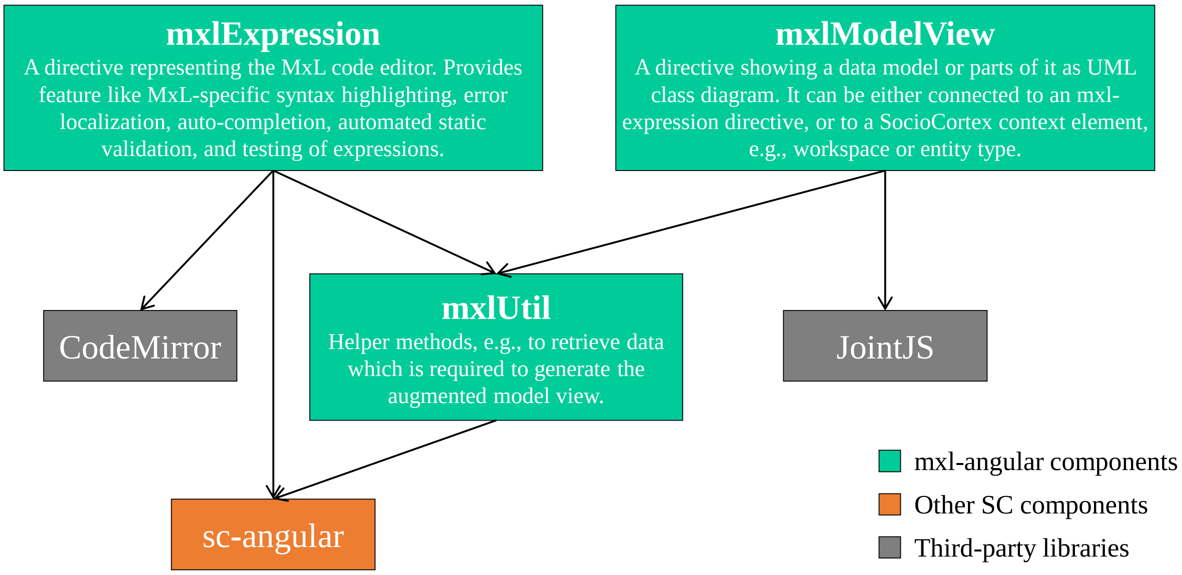
In the context of this thesis and as shown in the conceptual architecture of our prototype in Figure 5.1, defining queries is a use-case in multiple web clients, e.g., to define *DerivedAttributeDefinitions* in the SC Modeler, or to define *DataBindings* in the SC Visualizer. For this reason, we encapsulate the query editor and related UI components into the *mxl-angular* framework[12].

As illustrated in Figure 5.13, the central component within the mxl-angular framework is the *mxl-expression* directive which represents the MxL query editor (Reschenhofer et al., 2014b). It uses the CodeMirror[13] library with various extensions and defines MxL-specific adaptions.

By implementing MxL-specific syntax rules, the code editor is able to visually highlight MxL keywords (e.g., *this* and *find*), strings, comments, and other kinds of MxL tokens (cf. Figure 5.14). Furthermore, by connecting the code editor to a specific SocioCortex workspace, it can generate domain-specific auto-completion hints. This means that the auto-completion hints not only include generic elements (e.g., keywords or basic functions), but also workspace-specific data model elements or analytical abstractions. The example in Figure 5.14 demonstrates how the name of the attribute "Function points" is proposed to the user based on his input. The code editor also provides a textual description and type information for the auto-completion hints.

If the MxL expression is not changed for a particular configurable time-span (default is one

[12] `https://github.com/sebischair/mxl-angular`, last accessed on: 04.10.2016
[13] `https://codemirror.net/`, last accessed on: 04.10.2016

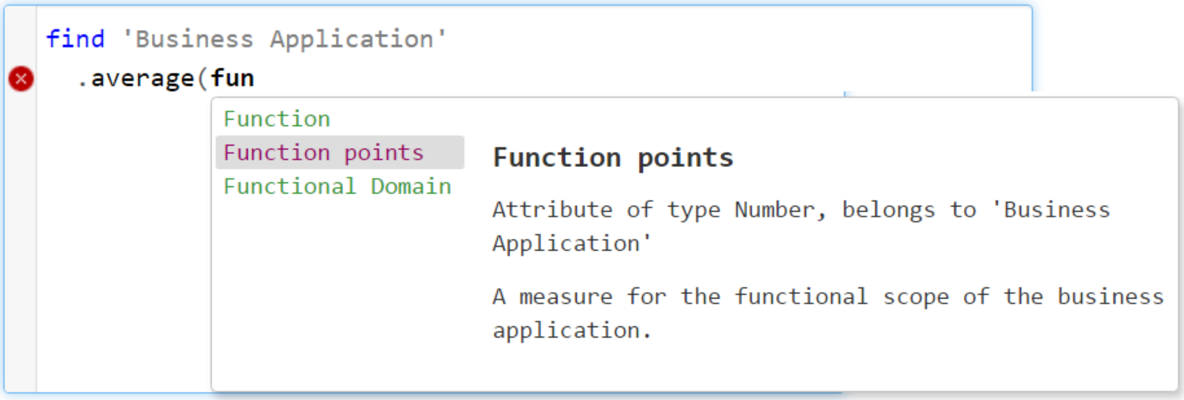


mxlExpression
A directive representing the MxL code editor. Provides feature like MxL-specific syntax highlighting, error localization, auto-completion, automated static validation, and testing of expressions.

mxlModelView
A directive showing a data model or parts of it as UML class diagram. It can be either connected to an mxl-expression directive, or to a SocioCortex context element, e.g., workspace or entity type.

mxlUtil
Helper methods, e.g., to retrieve data which is required to generate the augmented model view.

Figure 5.13.: The components of the mxl-angular framework and its dependencies to each other.



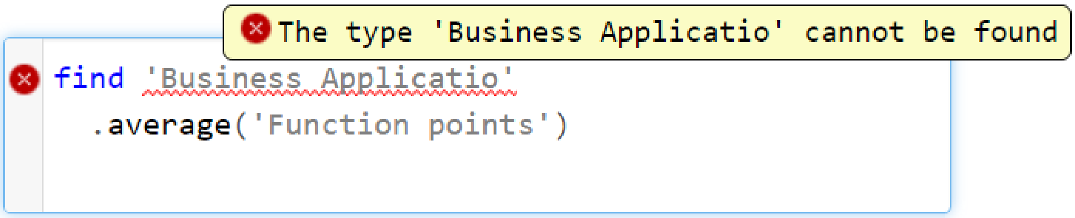
```
find 'Business Application'
  .average(fun
```

Function points

Attribute of type Number, belongs to 'Business Application'

A measure for the functional scope of the business application.

Figure 5.14.: The MxL code editor implements syntax highlighting and auto-completion support.

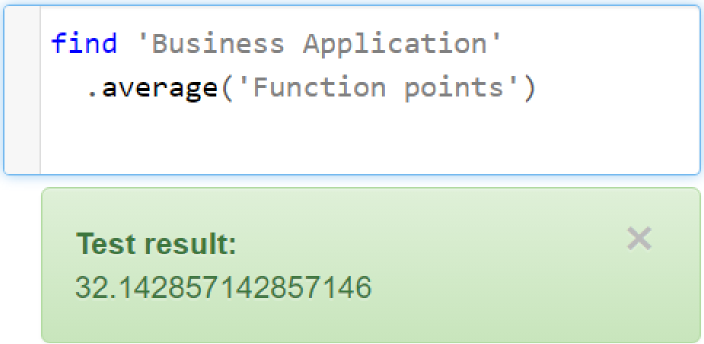


The type 'Business Applicatio' cannot be found

```
find 'Business Applicatio'
  .average('Function points')
```

Figure 5.15.: MxL expressions inside the code editor are statically analyzed. In case of errors, those are localized and visualized within the query.

second), the code editor automatically triggers a static analysis of its input. For this, it uses the *scMxl* service of the sc-angular framework. As a result of the invocation of the *validate* method of *scMxl*, the code editor knows if the expression is semantically valid. If this is not the case, it extracts the error information and location from the *validate* method's response, and visually highlights the erroneous part of the expression. The example in Figure 5.15 shows how an unknown entity type name is localized and underlined in red. As described in Section 2.2.1, immediate visual feedback (Tanimoto, 1990) is one of the essential features of EUA tools. In this case, it empowers end-users to check immediately if the formulated query is syntactically and semantically valid or not.



```
find 'Business Application'
  .average('Function points')
```

Test result:
32.142857142857146

Figure 5.16.: By testing the query, the users gets immediate feedback if it behaves like intended.

In order to check not only the syntactic and semantic validity of the expression, but also its result, end-users can simply trigger the execution of the query and analyze its response. For this, the code editor uses the *query* method of the *scMxl* service. Figure 5.16 shows the code editor as well as the test result obtained from the query's execution.

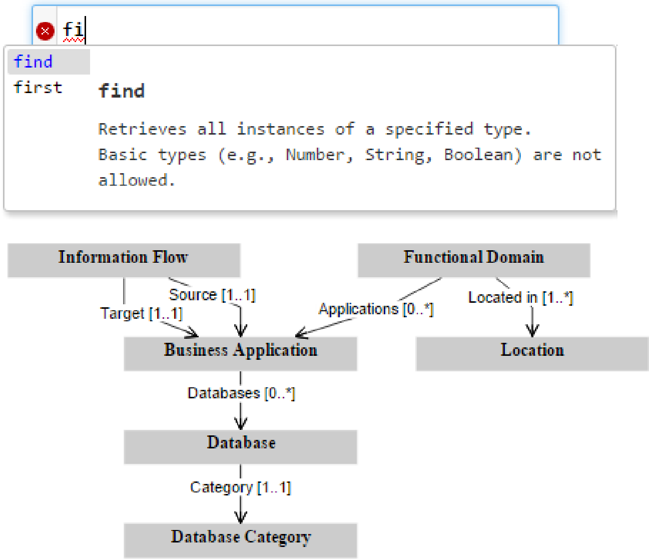
Technically, the MxL code editor is implemented as an AngularJS directive. It can be simply reused by web developers and embedded in their code as follows:

```
<mxl-expression style="height:200px;" ng-model="mxlValue"
  sc-workspace="{{workspaceId}}" mxl-expected="{{expectedType}}" >
</mxl-expression>
```

The *ng-model* attribute binds the given AngularJS model variable to the textual value of the code editor, i.e., if the input of the code editor is changing, this change is automatically propagated to the respective model variable. Through the *sc-workspace* attribute, developers can specify the context in which the expressions of the code editor should be semantically analyzed and evaluated. The directive also supports analogous *sc-entitytype* and *sc-entity* attributes. Furthermore, the *mxl-expected* attribute enables web developers to specify an expected type used for the semantic analysis of the expression (cf. Section 5.1.3).

For formulating a query, the user is assumed to be familiar with the underlying data model (Valencia-García et al., 2011). In case of continuously evolving data models as described in Section 1.1, the process of familiarizing oneself with the data model is also a continuous one. In fact, those cases end-users have to familiarize themselves with the underlying data model each time they intend to query it. For this reason, we developed an augmented model view which displays the current data model as a UML class diagram (Reschenhofer and Matthes, 2016b). The model view represents the second key component of the mxl-angular framework.

Reasons for selecting UML as a visual language to represent data models are two-fold: On the one hand, UML is widely adopted in related EUA research (Valencia-García et al., 2011; Störrle, 2011; Hermans et al., 2010). On the other hand, UML is the basis for Object Constraint Language (OCL) which in turn served as foundation for the design of MxL (cf. Section 4.2.2). Therefore, we considered UML as a natural choice to formulate OCL-like queries with MxL.



```
fi
find
first
```

find

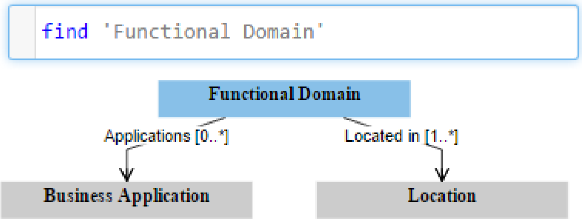
Retrieves all instances of a specified type. Basic types (e.g., Number, String, Boolean) are not allowed.

Figure 5.17.: Initially, the model view shows a global overview over the data model including entity types and relations between them (Reschenhofer and Matthes, 2016b).

Figure 5.17 shows an MxL code editor and an augmented model view for the exemplary data model in Figure 4.7. To identify a starting point for the formulation of a query, the augmented model view initially provides a holistic perspective on the data model. This global view includes the entity types for the workspace that is configured to be the context in which the MxL query is validated and evaluated. In order to reduce the visual complexity of the model view, attributes are omitted in the global perspective, while relations are still included.

As soon as the static analysis of the code editor's query identifies a reference to a data model element, the augmented model view is automatically updated and switches to a local view. The local view only contains directly referenced data model elements and—to ensure navigability through the data model—adjacent entity types. Furthermore, attributes of referenced entity types are also shown. In this sense, the local view is a measure to cope with the visual complexity of large data models.

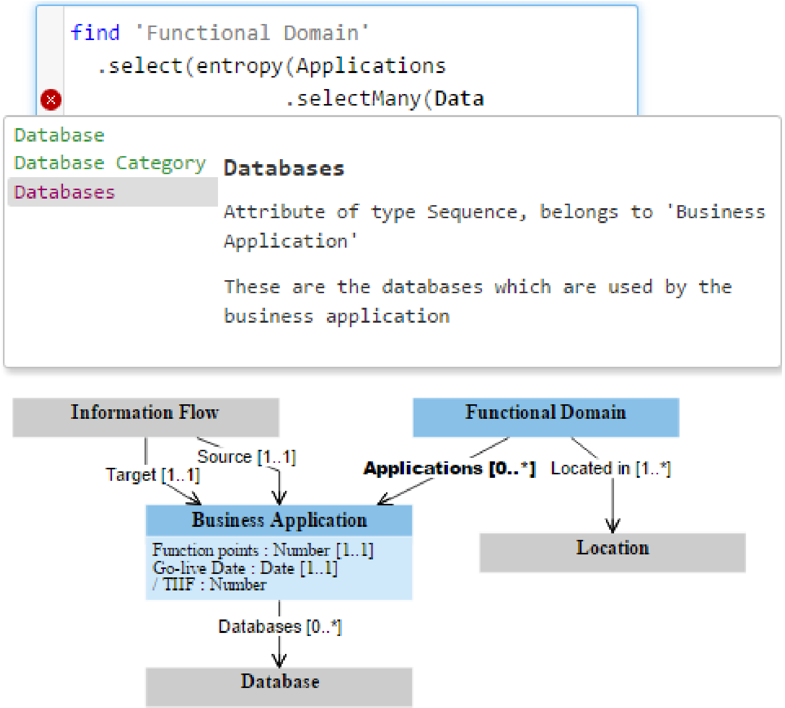
For example, assuming that we want to determine the data base heterogeneity for each function domain using the metric already introduced in Section 4.2.2.2, we would first start with retrieving all entities of type *Functional Domain* with the *find* construct. As demonstrated in Figure 5.18,



```
find 'Functional Domain'
```

Figure 5.18.: As soon as the expression within the code editor refers to a data model element, the model view will switch to a local view of the data model (Reschenhofer and Matthes, 2016b).

the augmented model highlights the respective entity type, and hides non-adjacent entity types. If there would be attributes for the entity type *Functional Domain*, they would be displayed.



```
find 'Functional Domain'
  .select(entropy(Applications
                  .selectMany(Data
```

Database
Database Category
Databases

Databases

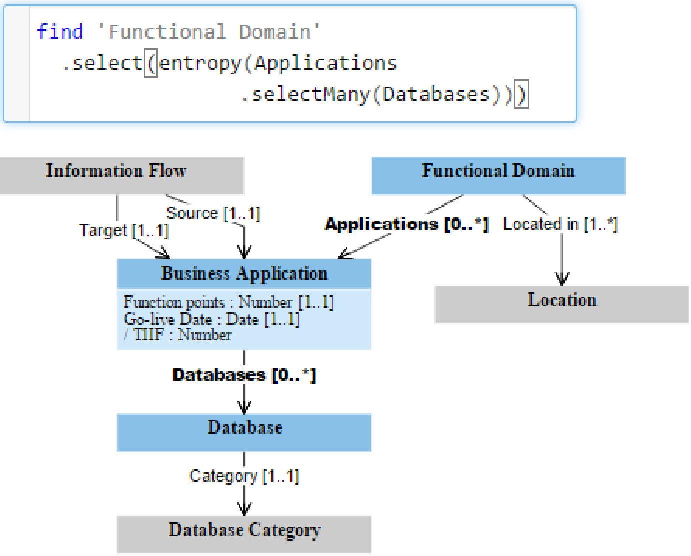
Attribute of type Sequence, belongs to 'Business Application'

These are the databases which are used by the business application

Figure 5.19.: The model view provides contextual information to the auto-completion hints (Reschenhofer and Matthes, 2016b).

By navigating further to through the data model, the augmented model view is updated accordingly and extends or reduces the set of visible entity types and attributes (cf. Figure 5.19). For example, by navigating to the entity type *Business Application* via the *Applications* relation, the attributes of this entity type become visible to the user. In this way, it provides contextual information to the auto-completion hints of the MxL query editor, i.e., the model view enriches the textual description and type information of auto-completion hints with an actual context.

Figure 5.20 shows the finished database heterogeneity metric and how the augmented model



```
find 'Functional Domain'
  .select(entropy(Applications
                   .selectMany(Databases)))
```

Figure 5.20.: Model elements which are referenced by the expression of the code editor are visually highlighted, e.g., by blue background and bold labels (Reschenhofer and Matthes, 2016b).

view visually highlights referenced data model elements either by a blue background or by bold labels.

Technically, the augmented model view is implemented as a separate AngularJS directive (*mxl-model-view*, cf. Figure 5.13). To generate the UML visualization, we use the open-source library *JointJS*[14].

The directive can be used independently from the code editor in order to visualize a manually created set of data model elements. For this purpose, the *mxlUtil* service of the mxl-angular framework (cf. Figure 5.13) provides helper methods, e.g., to query all data model elements of a given workspace. For this purpose, the *mxlUtil* service reuses the sc-angular framework and its services.

To connect the augmented model view with an MxL code editor, they have to be bound to the same AngularJS model variable:

```
<mxl-expression style="height:200px;" ng-model="mxlValue"
  sc-workspace="{{workspaceId}}" mxl-expected="{{expectedType}}"
  mxl-model-elements="referencedModelElements" >
</mxl-expression>

<mxl-model-view ng-model="referencedModelElements">
</mxl-model-view>
```

In this example, the AngularJS model variable *referencedModelElements* is bound to the code

[14] `http://jointjs.com/opensource`, last accessed on: 04.10.2016

editor via the *mxl-model-elements* attribute, and to the model view via the *ng-model* attribute. Due to AngularJS' two-way data binding, updates to this model variable through the code editor will be automatically propagated to the model view. The model view component watches the model variable provided by the *ng-model* attribute and triggers a regeneration of the data model visualization if the variable is changing.

Dividing the code editor and the model view into two separate components has two main benefits: First, the model view can be used without the code editor, e.g., to simply show an overview over all entity types defined within a workspace. Second, the model view can be arbitrarily arranged on a web page independently from the code editor's position.

5.3. SocioCortex Web Clients

As illustrated in Figure 5.1, the prototype for a tool empowering end-users to collaboratively manage and analyze evolving complex linked data consists of three separate web clients addressing different user roles.

The *SC Content Manager* represents the UI for casual users whose main purpose is the maintenance of data, whereas the *SC Modeler* is the tool for modeling experts allowing them to define and manage models of data, processes, and analytical abstractions. Those two clients are described in detail in Section 5.3.1.

As a third web client, the *SC Visualizer* empowers end-users to create model-based visualizations and compose them to domain-specific dashboards. The SC Visualizer implements the view and view template concepts as defined in Figure 4.10. Therefore, it is the central web client for analytics professionals and analytics end-users (cf. description of user roles in EUA in Section 3.1) and thus considered to be the most important web client in the context of this thesis. We elaborate on its design and the key aspects of its implementation in Section 5.3.2.

All three clients use the sc-angular framework in order to communicate with the SocioCortex back-end. By using the same configuration for the *scAuth* service (i.e., the same authentication method), the user has to login only once and is automatically authenticated to all these clients.

5.3.1. The SC Content Manager and the SC Modeler

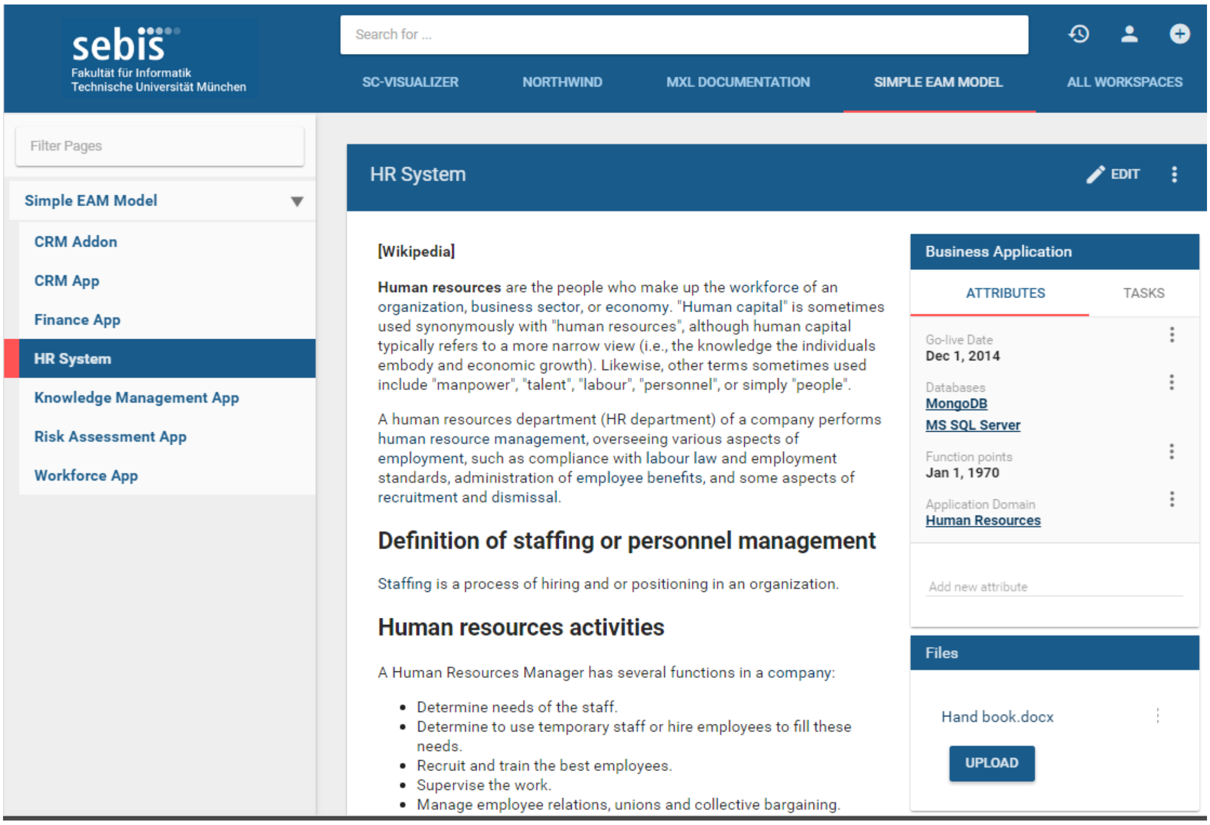
The SC Content Manager[15] and SC Modeler[16] address different use-cases: The former web client provides means to manage instances of data in a collaborative way, while the latter one allows users to define respective models. Taken together, they emulate the basic functionality of Tricia's front-end. However, in contrast to Tricia, the SC Content Manager and SC Modeler communicate with the back-end via a well-defined REST API on the on hand, and strictly separate data maintenance and data modeling on the other hand.

The initial design of the SC Content Manager was developed by Katenbrink (2015) in his bache-

[15] `https://github.com/sebischair/sc-contentmanager`, last accessed on: 04.10.2016

[16] `https://github.com/sebischair/sc-modeler`, last accessed on: 04.10.2016

lor's thesis. Based on lessons learned from the application of Tricia, and particularly based on an usage analysis with *Google Analytics*[17], mock-ups capturing the basic UI design and functionality of the SC Content Manager were created. Furthermore, Katenbrink (2015) applied principles and patterns from related work on the design of social web UIs (Crumlish and Malone, 2009; Kalbach, 2007; Krug, 2014; Scott and Neil, 2009). Based on the mock-ups, Michelsen (2016) implemented the first prototype of the SC Content Manager.



[Wikipedia]

Human resources are the people who make up the workforce of an organization, business sector, or economy. "Human capital" is sometimes used synonymously with "human resources", although human capital typically refers to a more narrow view (i.e., the knowledge the individuals embody and economic growth). Likewise, other terms sometimes used include "manpower", "talent", "labour", "personnel", or simply "people".

A human resources department (HR department) of a company performs human resource management, overseeing various aspects of employment, such as compliance with labour law and employment standards, administration of employee benefits, and some aspects of recruitment and dismissal.

Definition of staffing or personnel management

Staffing is a process of hiring and or positioning in an organization.

Human resources activities

A Human Resources Manager has several functions in a company:

- Determine needs of the staff.
- Determine to use temporary staff or hire employees to fill these needs.
- Recruit and train the best employees.
- Supervise the work.
- Manage employee relations, unions and collective bargaining.

Go-live Date
Dec 1, 2014

Databases
MongoDB
MS SQL Server

Function points
Jan 1, 1970

Application Domain
Human Resources

Add new attribute

Hand book.docx

Figure 5.21.: A single entity represented as a wiki page in the SC Content Manager.

Figure 5.21 shows a screenshot of a wiki page representing a single entity of SocioCortex. In the top navigation bar, users can select from a workspace and load its entities. In the screenshot, the workspace "Simple EAM Model" is selected, wherefore its entities are navigable through the sidebar on the left. In SocioCortex, entities can be organized hierarchically, i.e., originating from the root entity of a workspace, entities define a hierarchical structure which is displayed in the SC content manager in a respective tree view.

Selecting an entity loads its textual content as well as its attributes into the main area of the page. The textual content can be manipulated through a respective rich text editor. Analogous to the sc-datatable as described in Section 5.2.2, the attributes of an entity are edited via data type-specific input fields. Clicking on the entity type at the top of the attributes box forwards the user to a view containing the sc-datatable which shows all entities of the selected entity type.

[17] `https://www.google.com/analytics/`, last accessed on: 04.10.2016

Through those basic UI elements, end-users can either edit a single entity or multiple entities of a particular entity type.

While the SC Content Manager is the web client for casual users acting as data owners and custodians (Roth et al., 2014), the SC Modeler provides support for modeling expert. Thereby, it not only provides means to define data models, but also to define analytical abstractions and process structures based on Adaptive Case Management (ACM). However, since not relevant in the context of this thesis, we will not describe the SC Modeler's support for ACM in detail and refer to the work by Hauder (2015) and the master's thesis of Schrade (2016).

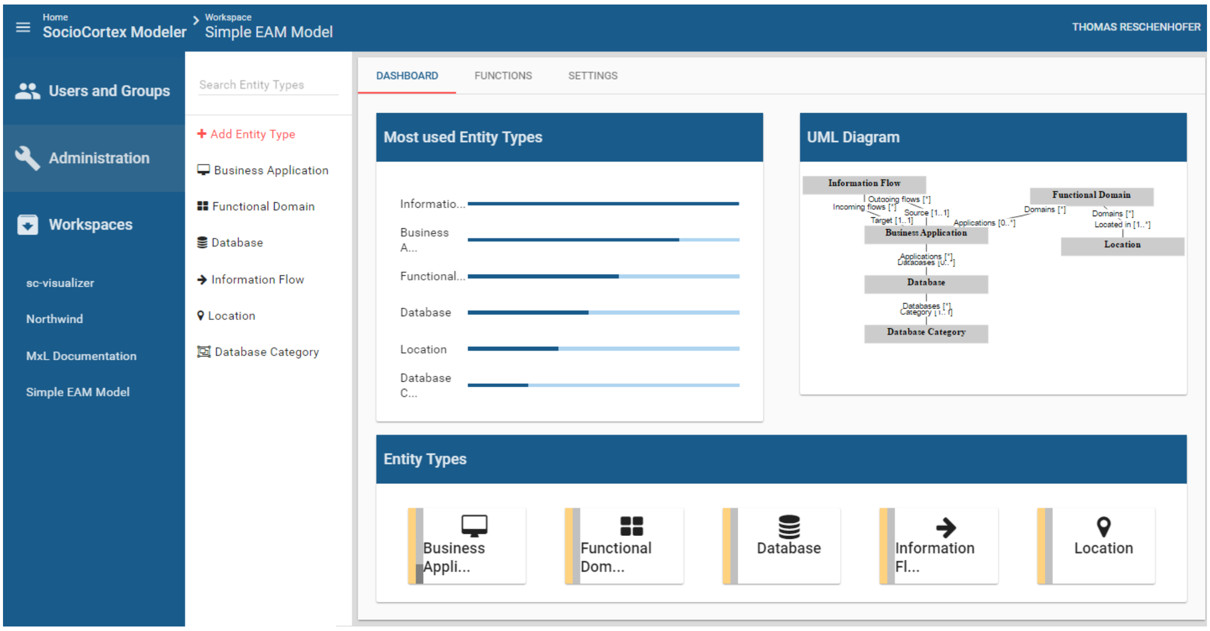


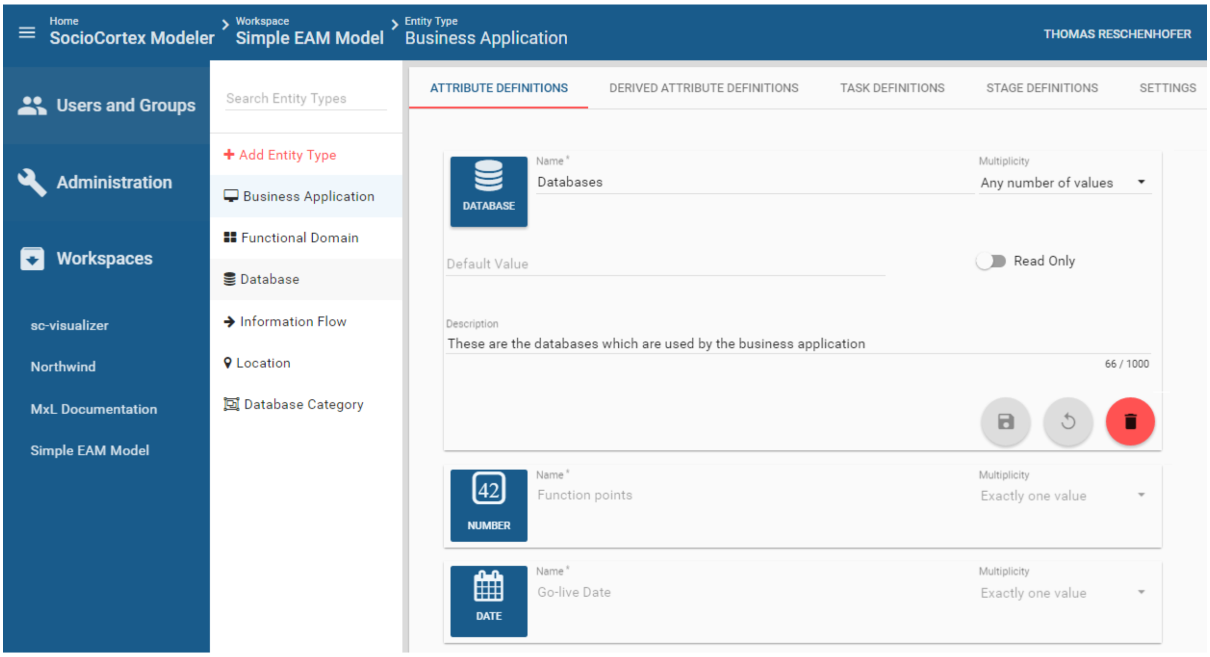
Figure 5.22.: Overview over a workspace's data model in the SC Modeler.

Selecting a workspace in the SC Modeler forwards the user to a dashboard as shown in Figure 5.22. In addition to a list of entity types in a secondary sidebar, the dashboard provides some statistics in a visual way. For example, a bar chart visualizes the number of instances for the most used entity types, while a UML class diagram shows an overview of the workspace's data model. For the class diagram, the SC Modeler uses the model view of the mxl-angular framework (cf. Section 5.2.3). Furthermore, the visualization on the bottom of the page lists a selection of entity types with each of them having two bars visualizing two different metrics:

Inconsistency The yellow bar indicates the inconsistency of the entities of the entity type. It is a measure for the ratio of entities that have attributes which are inconsistent with respect to the attribute definitions of the entity type. A high inconsistency measure and thus a high yellow bar indicates that the data model does not fit to the respective data or that it is too specific which in turn implies the need for adaptions.

Unstructuredness The gray bar is an indicator for the unstructuredness of the entity type's entities. Since entities can have free attributes (attributes which are not defined by a respective attribute definition), they might contain structures which are not yet captured by the data model. A high unstructuredness metric and thus a high gray bar indicates that the data model is not specific enough which in turn implies the need for adaptions.

The workspace view of the SC Modeler also includes tabs to manage custom MxL functions and to configure general workspace settings.



Name * Databases

Multiplicity Any number of values

Default Value

Read Only

Description These are the databases which are used by the business application

66 / 1000

Name * Function points

Multiplicity Exactly one value

Name * Go-live Date

Multiplicity Exactly one value

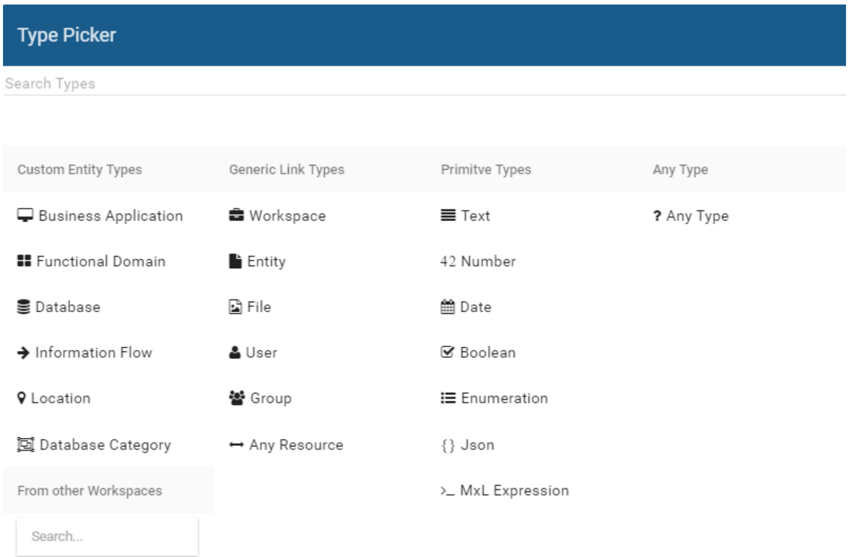
Figure 5.23.: Overview over an entity type's attribute definitions in the SC Modeler.

By clicking on an entity type, the user navigates to the entity type view as shown in Figure 5.23. In this view, users can manage the attribute definitions for the selected entity type. Users can change the data type, name, and multiplicity of attribute definitions as well as type-specific properties. Furthermore, the order of attribute definitions can be adapted via an intuitive Drag&Drop mechanism.

For changing the type of an attribute, the SC Modeler implements a type picker. As shown in Figure 5.24, the type picker includes primitive and complex types as defined by the extended Hybrid Wiki meta-model in Figure 4.3. Furthermore, it also enables users to define relations to other entity types. For this purpose, it lists the entity types of the same workspace. However, it also allows to search for entity types of other workspaces in order to define relations across multiple workspaces.

Clicking on the "Derived Attribute Definitions" tab forwards the user to the definitions of derived attributes. Figure 5.25 demonstrates how the SC Modeler uses both directives of the mxl-angular framework, namely the MxL code editor and the MxL model view. In this way, the augmented model view visualizes the currently defined data model and thus supports the end-user in defining MxL expressions. The expression for the derived attribute "TIIF" ("Transitive Incoming Information Flows") was already described in Section 4.2.2.3.

Further tabs within the entity type view are "Task Definitions" and "Stages" referring to ACM



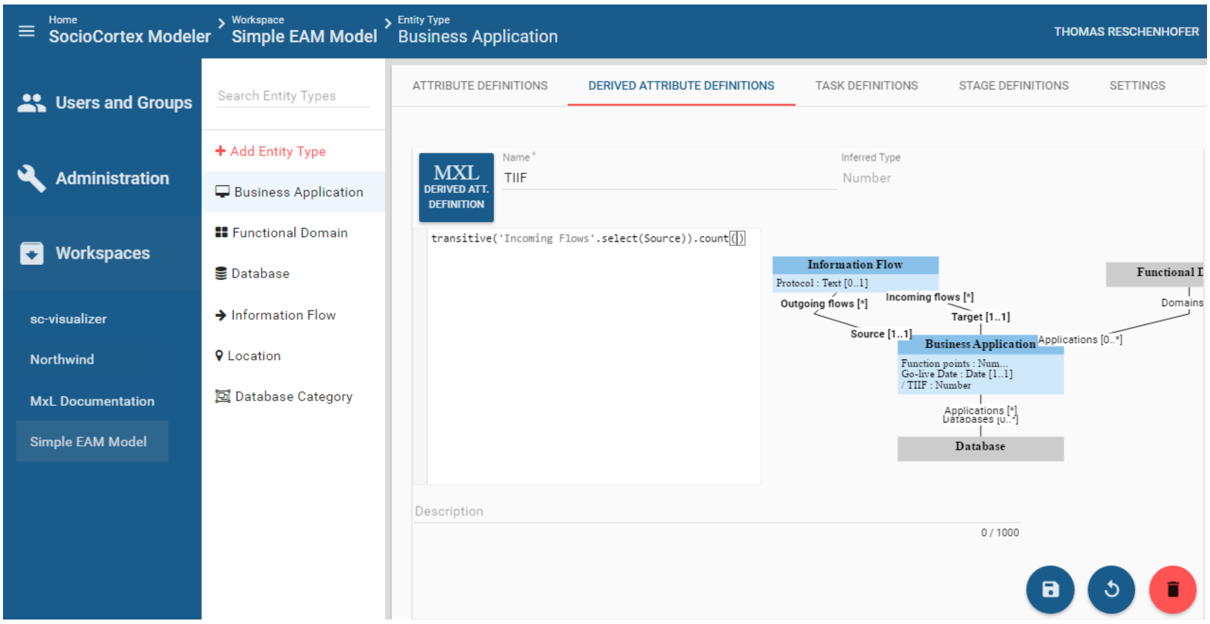
Type Picker

Search Types

Custom Entity Types	Generic Link Types	Primitve Types	Any Type
Business Application	Workspace	Text	? Any Type
Functional Domain	Entity	42 Number	
Database	File	Date	
→ Information Flow	User	☑ Boolean	
Location	Group	Enumeration	
Database Category	↔ Any Resource	{} Json	
From other Workspaces		>_ MxL Expression	

Search...

Figure 5.24.: In the SC Modeler, users can change the type of attribute definitions by using the type picker.



Name *
TIIF

Inferred Type
Number

```
transitive('Incoming Flows'.select(Source)).count()
```

Description
0 / 1000

Figure 5.25.: Defining a derived attribute in the SC Modeler.

concepts (Schrade, 2016). Via the "Settings" tab, users can edit general settings of the entity type, e.g., its name or its icon.

5.3.2. The SC Visualizer

The SC Content Manager and the SC Modeler empower end-users with different roles to manage and structure complex linked data. However, the actual End-User Analytics (EUA) tasks are performed through the SC Visualizer[18]. For this reason, we describe this web client in more detail, and particularly elaborate on its technical architecture as well as its UI design.

5.3.2.1. Architecture of the SC Visualizer

Analogous to the SC Content Manager, and the SC Modeler, the SC Visualizer is an AngularJS-based web application (Bürgin, 2015). Therefore, it uses the sc-angular framework to consume the services and manipulate the resources of SocioCortex. Furthermore, it also integrates mxl-angular and applies the MxL code editor and model view components.

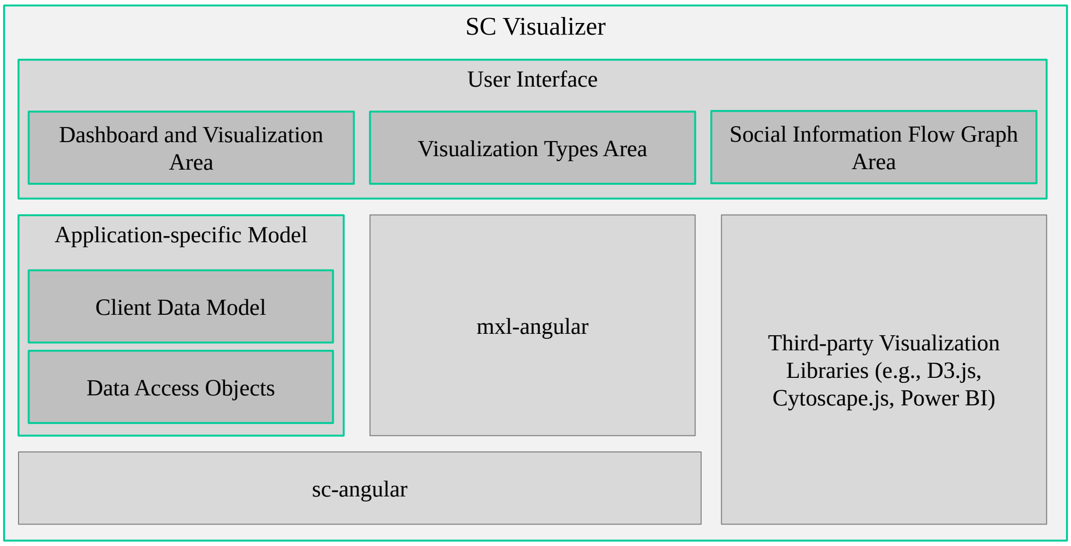

SC Visualizer
User Interface
Dashboard and Visualization Area
Visualization Types Area
Social Information Flow Graph Area
Application-specific Model
Client Data Model
Data Access Objects
mxl-angular
Third-party Visualization Libraries (e.g., D3.js, Cytoscape.js, Power BI)
sc-angular

Figure 5.26.: The conceptual architecture of the SC Visualizer.

In contrast to the other SC web clients as described in Section 5.3.1, the SC Visualizer not only represents the generic structures as received from SocioCortex, but also implements an application-specific business logic. The SC Visualizer's architecture is illustrated in Figure 5.26, and also includes an application-specific client data model which is mapped to the generic SocioCortex resources via *Data Access Objects* (DAO). Additionally, the SC Visualizer integrates multiple common JavaScript Visualization Libraries, e.g., D3.js[19], Cytoscape.js[20], and Power

[18] `https://github.com/sebischair/sc-visualizer`, last accessed on: 04.10.2016
[19] `https://d3js.org/`, last accessed on: 04.10.2016
[20] `http://js.cytoscape.org/`, last accessed on: 04.10.2016

BI[21]. On the top of those components, the SC Visualizer architecture defines different UI components for managing dashboards and visualizations, managing and developing visualization types, and for exploring the Social Information Flow Graph (SIFG).

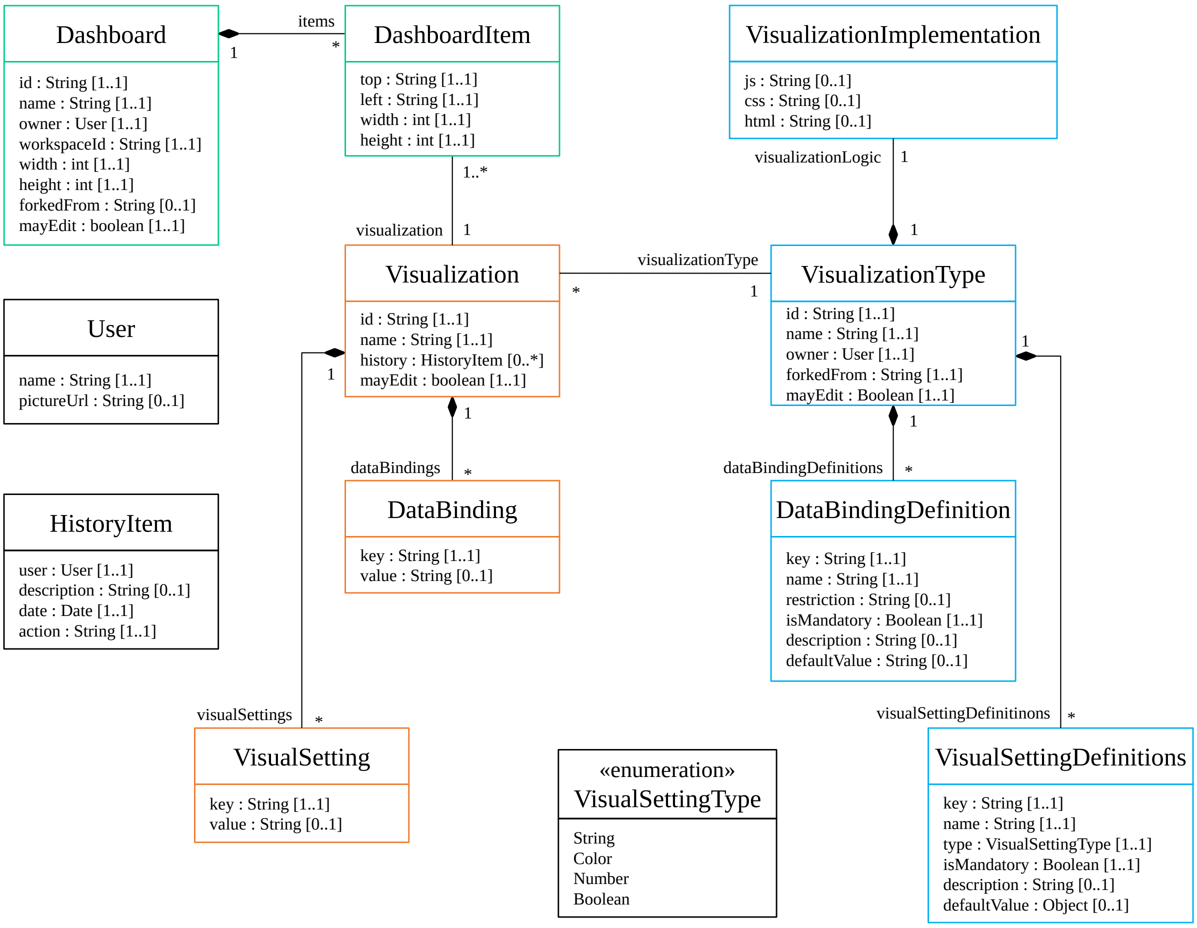


Figure 5.27.: The implemented client data model of the SC Visualizer as a concrete manifestation of the conceptual meta-model in Figure 4.10. The coloring of the classes indicates how they are composed and mapped to respective entity types.

Figure 5.27 depicts the implemented client data model. It is a concrete manifestation of the conceptual data model as described in Section 4.3.1 and as shown in Figure 4.10. It is implemented as respective classes and attributes in *TypeScript*[22], which is a typed superset of JavaScript.

As a concrete *View* class, the SC Visualizer implements *Dashboard*. A *Dashboard* has an explicit owner and is assigned to a specific SocioCortex workspace. In order to reuse a dashboard and its visualizations, users can fork it and independently adapt it to personal demands. To keep track of which dashboards were forked from others, each *Dashboard* maintains the relation to its origin via the *forkedFrom* attribute. The *mayEdit* flag indicates if the current user is eligible to adapt the dashboard.

[21] `https://github.com/Microsoft/PowerBI-JavaScript`, last accessed on: 04.10.2016

[22] `https://www.typescriptlang.org/`, last accessed on: 04.10.2016

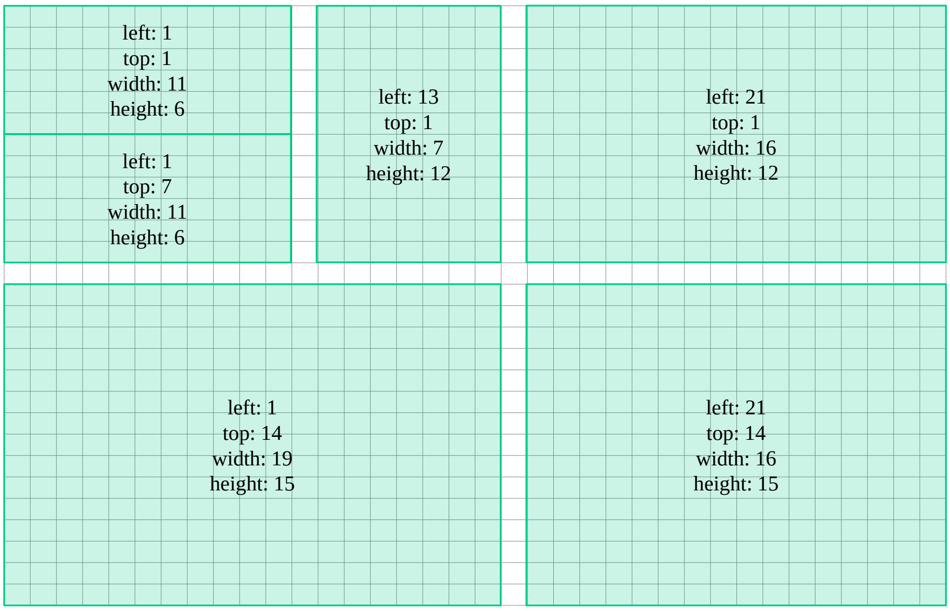


Figure 5.28.: An example illustrating the grid system of the SC Visualizer dashboard.

Furthermore, the dashboard's *width* and *height* attributes define the size of the dashboard's visual grid on which its visualizations are placed. On this grid system, end-users can place any number of *DashboardItems* and align them arbitrarily on the whole dashboard surface. The position and size of *DashboardItems* are implemented as respective attributes. Figure 5.28 illustrates an exemplary dashboard including dashboard items, and shows their respective position and size values. Therefore, the SC Visualizer provides an intuitive and flexible way to define, assemble, and arrange different kinds of visualizations. In this sense, the SC Visualizer complies to Enterprise Mashup (EM) systems (Soriano et al., 2007; Hoyer et al., 2008; López et al., 2009; Pahlke et al., 2010) which allow end-users to tailor the application's behavior to their individual needs.

Each *DashboardItem* is the container for exactly one *Visualization*. In addition to its ID, name, and a *mayEdit* flag, each *Visualization* contains its version history as respective attribute. The version history is automatically maintained by SocioCortex and is defined by a temporally ordered list of *HistoryItems* which in turn capture who changed what at which time. In line with the conceptual meta-model in Figure 4.10, a *Visualization* is defined by a *VisualizationType* and is bound to data via *DataBindings*. The *DataBinding* only has a key which refers to a corresponding *DataBindingDefinition* as well a value containing an actual MxL query which represents a data transformation tailoring the underlying data to the *Visualization* information demand. In addition to *DataBindings*, a *Visualization* also contains *VisualSettings* through which end-users do not define data to be bound to the *Visualization*, but static visual properties, e.g., colors and labels. Analogous to *DataBindings*, *VisualSettings* are also defined by corresponding *VisualSettingDefinitions*.

The *VisualizationType* represents reusable visual components (Pahlke et al., 2010). Similar to *Dashboards*, *VisualizationTypes* have an explicit owner. Furthermore, they also provide a forking mechanism in order to allow the reusability of their implementation and definition. Again, their relationship to potential origins is maintained by the *forkedFrom* attribute.

VisualizationTypes define a data input interface constituted by a set of *DataBindingDefinitions* through which data obtained through MxL queries from the underlying SocioCortex platform is entered into its logic. A *DataBindingDefinition* has a unique *key* attribute as well as a name which also serves as label shown to the user. The *restriction* attribute allows users to define the type of the *DataBindingDefinition* by using the MxL type notation (cf. Section 4.3.2). Furthermore, the *DataBindingDefinition* has self-explanatory attributes *isMandatory*, *description*, and *defaultValue*. *VisualSettingDefinitions* are analogously defined with the exception that they define a *type* attribute of type *VisualSettingType* instead of an MxL restriction. Currently implemented *VisualSettingTypes* are *String*, *Color*, *Number*, and *Boolean*.

The implementation of a *VisualizationType* is defined by an associated *VisualizationImplementation* object. This object consists of three attributes for defining the JavaScript (JS), Cascading Style Sheets (CSS), and HTML part of the visualization implementation. While the CSS and HTML attributes allow web designers and developers to specify styles and templates, the JS attribute contains the actual business logic of the *VisualizationType*. The value of this attribute is interpreted as the body of a *render* function which defines three parameters:

element This parameter represents the root element of the visualization component.

data The *data* parameter encapsulates the data obtained from executing the MxL queries as defined by the *DataBindings*. For each *DataBinding*, the *data* object contains a correspondingly named property whose value is the result of the query execution. For example, if the *VisualizationType* defines a *DataBindingDefinition* named "numbers" of type *Sequence<Number>*, the *data* object has a property "numbers" whose value will be a JSON array containing numbers.

settings Analogous to the *data* object, the *settings* object encapsulates the visual settings as configured by the end-user at run-time.

As described in Section 4.3.1, the SocioCortex back-end does not implement the view and view template concepts as defined in Figure 4.10 as built-in model classes. Therefore, the client data model is mapped to respective entity types and attribute definitions. Thereby, the green-colored concepts of the client data model in Figure 5.27 (*Dashboard* and *DashboardItem*) are composed and mapped to an entity type *Dashboard*, the orange-colored concepts (*Visualization*, *DataBinding*, and *VisualSetting*) are composed and mapped to an entity type *Visualization*, and the blue-colored concepts (*VisualizationType*, *VisualizationImplementation*, *DataBindingDefinition*, and *VisualSettingDefinition*) are composed and mapped to an entity type *Visualization Type*.

To map objects of the client data model to a corresponding representation as generic entity, the SC Visualizer defines a Data Access Object (DAO) for each of the entity types *Dashboard*, *Visualization*, and *Visualization Type*. As shown in Figure 5.29, a DAO class provides methods to read and write corresponding client model elements. Thereby, they parse the client-side representation into a generic SocioCortex entity representation, and vice versa. To transfer the client state from the front-end to the back-end, the DAO object uses the sc-angular framework

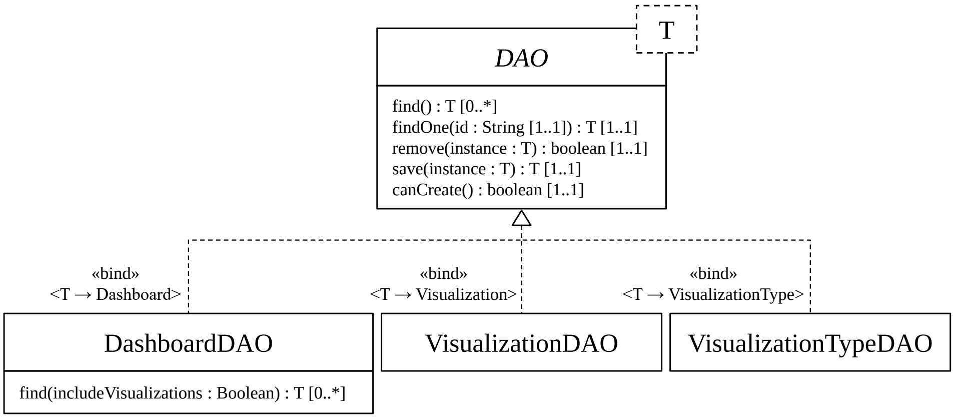

T
DAO
find() : T [0..*]
findOne(id : String [1..1]) : T [1..1]
remove(instance : T) : boolean [1..1]
save(instance : T) : T [1..1]
canCreate() : boolean [1..1]
«bind»
<T → Dashboard>
«bind»
<T → Visualization>
«bind»
<T → VisualizationType>
DashboardDAO
find(includeVisualizations : Boolean) : T [0..*]
VisualizationDAO
VisualizationTypeDAO

Figure 5.29.: The Data Access Object (DAO) classes map the client model to generic SocioCortex entities and attributes, and manage the communication via sc-angular.

and particularly its *scData* service. However, each concrete DAO class can implement its methods individually, e.g., the *find* method of the *DashboardDAO* class defines an optional boolean parameter *includeVisualizations* which indicates if each dashboard's visualizations should be includes in the response. This is efficiently implemented by a corresponding MxL query and by using the *scMxl* service.

On the UI level, the SC Visualizer has three main areas which are also represented by respective items in the navigation bar at the top of the page:

Dashboards In this area, users can explore existing dashboards and create new ones. If users have respective access rights, they can also edit existing ones, i.e., add new visualizations, reconfigure existing ones, rearrange them on the dashboard grid, or remove them (Reschenhofer and Matthes, 2016a).

Visualization Types In this area, end-users acting as web developers can explore and adapt existing visualization types and create new ones either from scratch or by forking existing ones.

Social Information Flow Graph (SIFG) In this area, end-users can perform a meta-analysis of existing dashboards and their relations and dependencies to underlying analytical abstractions and data model elements (Reschenhofer et al., 2016b).

Those three areas are explained in detail in the following sections.

5.3.2.2. Dashboards and Visualizations in the SC Visualizer

On the *Dashboards* area of the SC Visualizer, end-users can explore a list of existing dashboards. This list not only shows basic information (e.g., the dashboard's owner), but also provides links to directly edit the dashboard (if the user has respective access rights) and to open the dashboard in the SIFG area in order to analyze its dependencies.

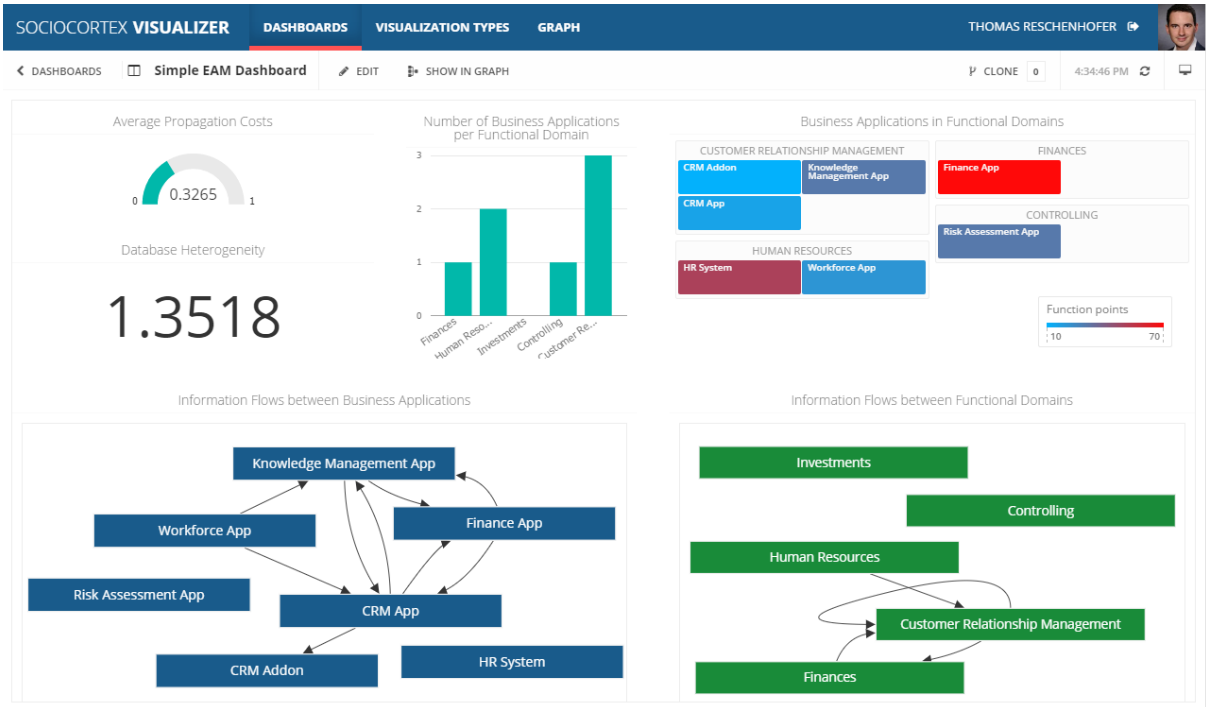


Figure 5.30.: An exemplary dashboard in the SC Visualizer.

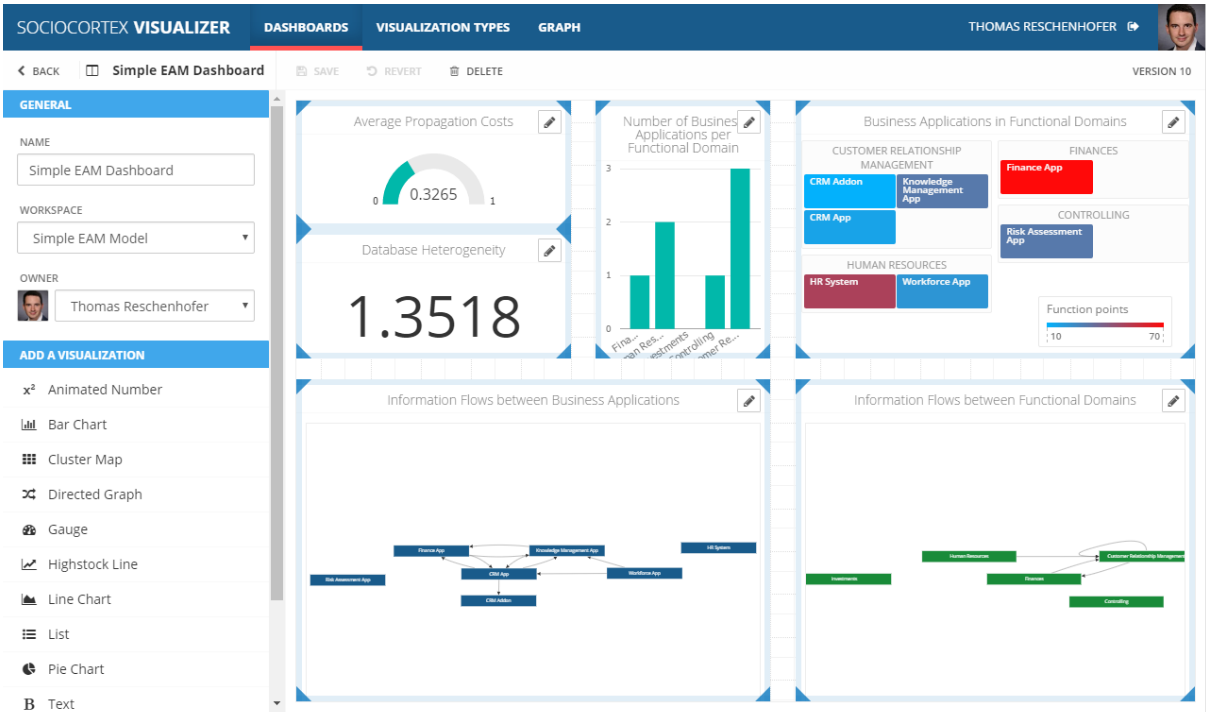
Selecting a dashboard opens it and loads its visualizations. Figure 5.30 shows an exemplary dashboard containing visualizations and visualizing metrics which are already described in previous sections. For example, the *Database Heterogeneity* visualization visualizes the heterogeneity metric as defined in Section 4.2.2.3, while the *Information Flows between Business Applications* visualization was already explained in Section 4.3.2.

The second-level navigation allows users to navigate back to the list of dashboards and to edit the selected one. Furthermore, by clicking the "Show in Graph" link the user can open the SIFG view with only the selected dashboard as subject of the meta-analysis. The number besides the "Clone" label indicates how often this dashboard was forked, while clicking the right-most button in the second-level navigation bar opens the dashboard in full-screen mode.

Addressing a shortcoming as identified in our related empirical study on spreadsheets (Reschenhofer and Matthes, 2015a), each visualization can be downloaded as editable MS PowerPoint[23] slide. In this way, end-users can still perform minor visual adaptions of visualizations and use generated visualizations as input for MS PowerPoint presentations.

Clicking the "Edit" button switches the dashboard into the edit mode as shown in Figure 5.31. In this mode, the grid is becoming visible on which users can rearrange and resize visualizations by an intuitive Drag&Drop mechanism. Figure 5.28 illustrates the configuration of the location and size of the dashboard's visualizations. Furthermore, in the edit mode a sidebar appears on the left-hand side containing input fields to specify the dashboard's attributes, e.g., its name, its

[23] `https://products.office.com/en/powerpoint`, last accessed on: 04.10.2016



NAME
Simple EAM Dashboard

WORKSPACE
Simple EAM Model

OWNER
Thomas Reschenhofer

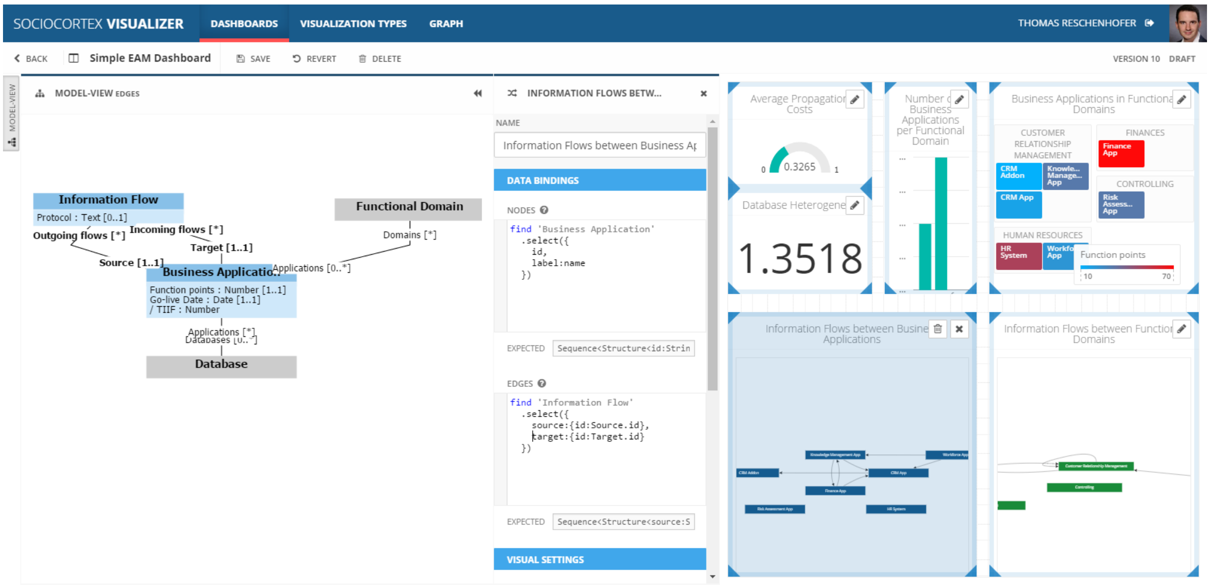
Figure 5.31.: A dashboard of the SC Visualizer in edit mode.

owner, and the SocioCortex workspace it is assigned to. Underneath the dashboard attributes, the SC Visualizer shows a catalog of available visualization types which can be created and instantiated in the dashboard. This visualization type catalog compares to the widget catalog as described by Soriano et al. (2007).

Each visualization is configured separately whereas its configuration can be opened by clicking the "Edit" icon in the top right corner of a visualization. Since visualizations are configured independently from each other, they can also be configured in parallel and collaboratively by multiple users. As demonstrated in Figure 5.32, a visualization's configuration consists of the specification of its name and providing queries and values for data bindings and visual settings as defined by the corresponding visualization type. For the instantiation of a data binding, end-users have to provide an MxL query which complies to the data binding's restriction. Of course, users can refer to and reuse existing analytical abstractions, e.g., custom functions. The MxL code editor as well as the augmented model view component facilitate the formulation of the query in general and the familiarization with the underlying data model in particular.

Addressing an essential feature of EUA tools as described in Section 2.2.1, a visualization is rendered as soon as one of its data bindings or visual settings is adapted without having to commit and save those changes. In this way, the user gets an immediate visual feedback if the defined query leads to the intended effect without having to commit changes (Abraham et al., 2008; Tanimoto, 1990; Rothermel et al., 2000).

To commit changes to the visualizations and the dashboard, users have to click on the "Save"



NAME

Information Flows between Business Ap

DATA BINDINGS

NODES

```
find 'Business Application'
  .select({
    id,
    label:name
  })
```

EXPECTED Sequence<Structure<id:Strin

EDGES

```
find 'Information Flow'
  .select({
    source:{id:Source.id},
    target:{id:Target.id}
  })
```

EXPECTED Sequence<Structure<source:S

VISUAL SETTINGS

Figure 5.32.: Configuration of a visualization of a dashboard within the SC Visualizer.

button in the second-level navigation bar. Alternatively, they can revert all changes and restore the original state of the dashboard and its visualizations.

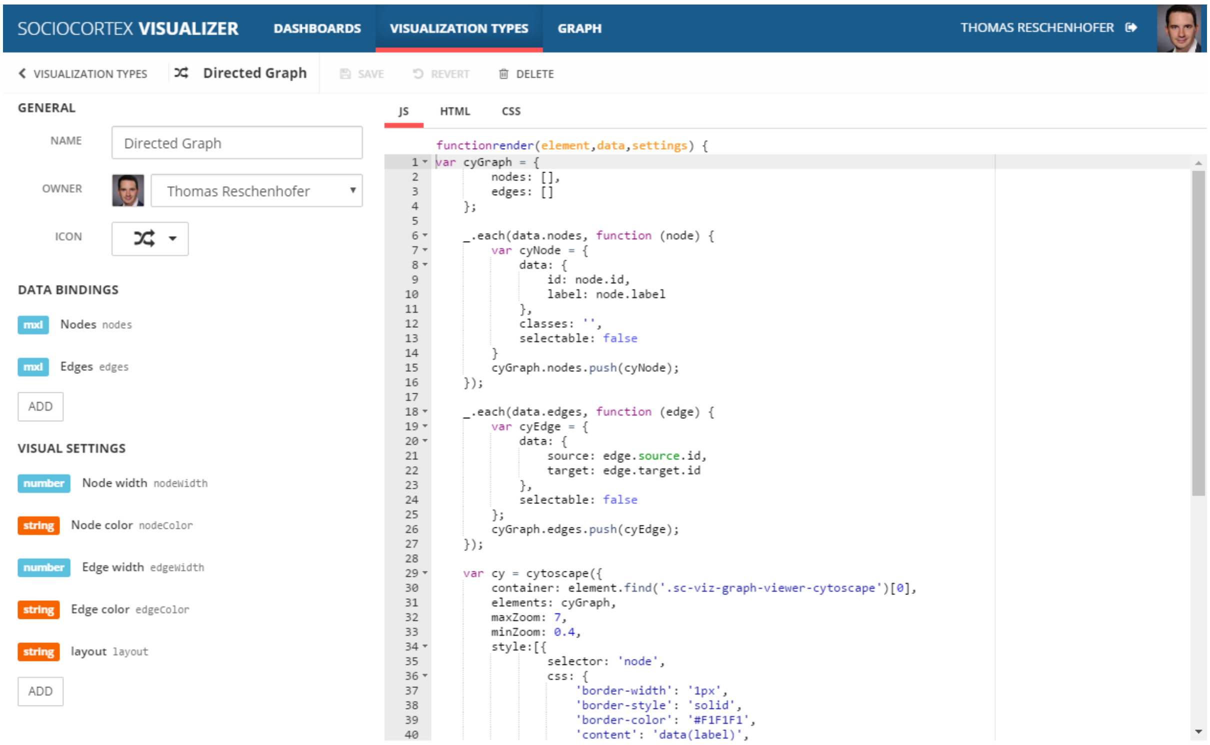
5.3.2.3. Managing Visualization Types

While the *Dashboards* area is addressing analytics end-users and analytics professionals (Tamm et al., 2013), the *Visualization Types* area is targeting web developers and designers. The landing page of this area is again a list of all existing visualization types to which the current user has access to. Each item of the list includes information about the visualization type's owners and also the number of visualizations which are instantiating it.

Selecting a visualization type from the list forwards the user to its details as shown in Figure 5.33. In contrast to dashboards, a visualization type is immediately opened in the edit mode, i.e., the visualization type's attributes are already editable. Those attributes include a name, an owner, and an icon, and can be specified within the sidebar.

In addition to those attributes, users can specify a set of data binding definitions and visual setting definitions. Figure 5.34 illustrates the specification of the data binding definition "Nodes" which was already described in Section 4.3.2. The user provides a name which acts as label for the corresponding data bindings. Using this name, the SC Visualizer also generates an identifier by which the developers can access the value of the data binding within the visualization type's implementation. Adding a restriction by using the MxL type notation ensures that users define consistent and complying MxL queries when instantiating the visualization type. Furthermore, defining an optional description using the Markdown[24] syntax allows users to attach a textual information to the data binding definition.

[24] http://daringfireball.net/projects/markdown/, last accessed on: 04.10.2016



GENERAL

NAME Directed Graph

OWNER Thomas Reschenhofer

ICON

DATA BINDINGS

mxl Nodes nodes

mxl Edges edges

VISUAL SETTINGS

number Node width nodeWidth

string Node color nodeColor

number Edge width edgeWidth

string Edge color edgeColor

string layout layout

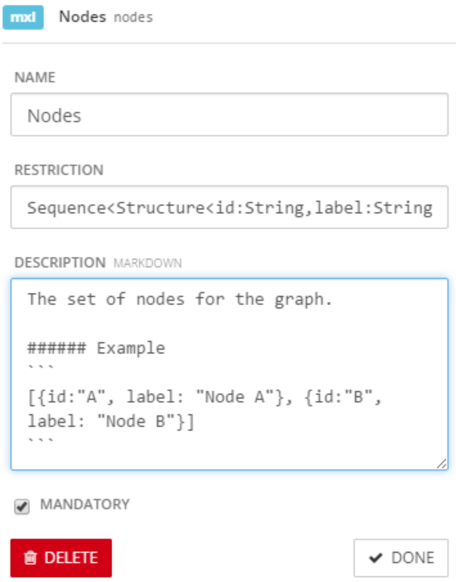
```
functionrender(element,data,settings) {
var cyGraph = {
        nodes: [],
        edges: []
    };

    _.each(data.nodes, function (node) {
        var cyNode = {
            data: {
                id: node.id,
                label: node.label
            },
            classes: '',
            selectable: false
        }
        cyGraph.nodes.push(cyNode);
    });

    _.each(data.edges, function (edge) {
        var cyEdge = {
            data: {
                source: edge.source.id,
                target: edge.target.id
            },
            selectable: false
        };
        cyGraph.edges.push(cyEdge);
    });

    var cy = cytoscape({
        container: element.find('.sc-viz-graph-viewer-cytoscape')[0],
        elements: cyGraph,
        maxZoom: 7,
        minZoom: 0.4,
        style:[{
                selector: 'node',
                css: {
                    'border-width': '1px',
                    'border-style': 'solid',
                    'border-color': '#F1F1F1',
                    'content': 'data(label)',
```

Figure 5.33.: A screenshot of the SC Visualizer showing a page for editing the visualization Type "Directed Graph".



mxl Nodes nodes

NAME

Nodes

RESTRICTION

Sequence<Structure<id:String,label:String

DESCRIPTION MARKDOWN

```
The set of nodes for the graph.

###### Example
```
[{id:"A", label: "Node A"}, {id:"B",
label: "Node B"}]
```
```

☑ MANDATORY

Figure 5.34.: A form within the SC Visualizer to define a data binding definition.

The main content of the visualization type view is dedicated to its implementation (cf. Figure 5.33). Through the three tabs *JS*, *CSS*, and *HTML* the user can access the different parts of the implementation.

Particularly the JS tab is of interest, since in this part the data obtained from executing the queries as defined by the data bindings is entered. The editable JavaScript code represents the body of the render function. As already explained in the previous section, the *data* and *settings* parameters represent the input provided by the end-user when configuring an instantiated visualization type. For example, since the visualization type in Figure 5.33 defines a data binding named "Nodes", the SC Visualizer generates a property "nodes" for the *data* parameter whose value is the result of the execution of the data binding's MxL query. Since the restriction of this data binding is defined as *Sequence<Structure<id:String,label:String> >*, the developer of the visualization type can assume that the value of the "nodes" property is a JSON array consisting of JSON objects which in turn have at least the properties *id* and *label*. Based on this input and the visual configuration passed within the settings parameter, the developer can implement visualizations based on proven visualization libraries which are already integrated in the SC Visualizer, e.g., D3.js, Cytoscape.js, and Power BI.

5.3.2.4. The Social Information Flow Graph (SIFG)

The third part of the SC Visualizer is the Social Information Flow Graph (SIFG) which is addressing all kinds of end-users (Reschenhofer et al., 2016b). The SIFG can be generated either for one selected dashboard, or as a holistic graph including all dashboards to which the current user has read access to.

The generation of the SIFG is done in multiple steps and is illustrated in the UML sequence diagram in Figure 5.35. All steps except the last one are independent from any actual technology or technical platform which is used for the implementation of the SIFG.

In the first step, the SIFG component determines all dashboards which have to be analyzed and loads their details from the SocioCortex back-end. For this purpose, the SIFG component uses the *DashboardDAO* object, and invokes the *findAll* method with its *includeVisualizations* parameter set to be *true*. In case of only one dashboard which has to be specified, the SIFG component uses the *find* method of the *DashboardDAO* and thus only requests a single dashboard object including its visualizations. The *DashboardDAO* object generates a corresponding request by using the *scMxl* service of the sc-angular framework, and returns the response to the SIFG component.

In a second step, the SIFG component loads all visualization types in order to be able to add visualization type-specific information to the visualization instances of the analyzed dashboards.

Subsequently, the SIFG component iterates through the data bindings of all visualizations of the obtained dashboards, and invokes a static analysis of the MxL queries of each of them. For this, it uses the *validate* function of the *scMxl* service. The result of the static validation is an object which already includes all transitive dependencies originating from the data binding's MxL expression down to data model elements as well as all analytical abstractions in between. The response of the *validate* function is designed in a way that it already includes all details of

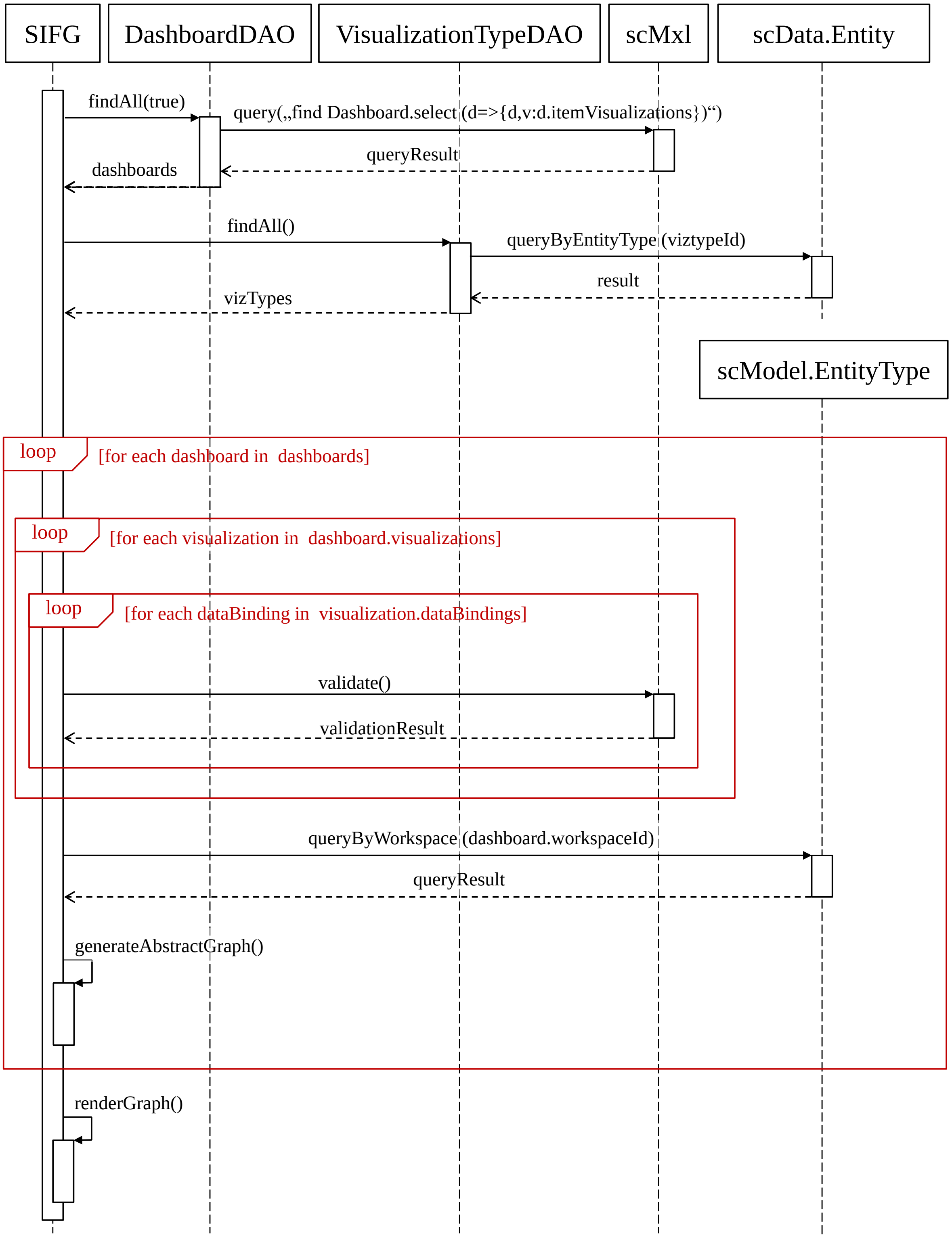

SIFG
DashboardDAO
VisualizationTypeDAO
scMxl
scData.Entity
findAll(true)
query(„find Dashboard.select (d=>{d,v:d.itemVisualizations})“)
queryResult
dashboards
findAll()
queryByEntityType (viztypeId)
result
vizTypes
scModel.EntityType
loop
[for each dashboard in dashboards]
loop
[for each visualization in dashboard.visualizations]
loop
[for each dataBinding in visualization.dataBindings]
validate()
validationResult
queryByWorkspace (dashboard.workspaceId)
queryResult
generateAbstractGraph()
renderGraph()

Figure 5.35.: A UML sequence diagram illustrating how the SC Visualizer generates the Social Information Flow Graph (SIFG).

data model elements and analytical abstractions which are required to render the SIFG. As a consequence, solely based on the results of the individual static analyses, the SIFG component can build an initial abstract graph for each dashboard already containing all directly and transitively referenced data model elements and analytical abstractions. For example, let us consider the following exemplary MxL query:

```
let applications = find 'Business Application' in

applications.sum(TIIF) / applications.count()^2
```

This metric is named *Average Propagation Cost* (Lagerström et al., 2014; Schneider et al., 2015) and calculates the average ratio of business applications which are affected by changing a random application. For this, it uses the derived attribute definition *TIIF* as described in Section 4.2.2.3. Performing a static analysis of the expression leads to a response of the *validate* method of the *scMxl* service, which encodes the expression's (transitive) static dependencies. The following object represents a condensed excerpt of this object:

```
{
  dependencies : [
    {
      customFunction : { name : "averagePropagationCost" },
      dependencies : [
        {
          entityType : { "name" : "Business Application" }
        },
        {
          derivedAttributeDefinition : { name : "TIIF" },
          dependencies : [
            {
              attributeDefinition : { name : "Target" }
            },
            {
              attributeDefinition : { name : "Source" }
            },
            {
              entityType : { name : "Information Flow" }
            }
          ]
        }
      ]
    }
  ]
}
```

In this example, we removed all details of each node, e.g., its unique key and context information. Nevertheless, this object already demonstrates how the (transitive) dependencies of an MxL expression area encoded within a deeply nested JSON object. For example, not only the derived attribute definition *TIIF* is included as dependency, but also the dependencies of *TIIF* itself.

For specific use cases, entity types which are not (yet) referenced might be of interest. Therefore, in a next step the SIFG component requests all missing entity types from the workspace assigned to each analyzed dashboard.

As a last step, the abstract graph consisting of abstract nodes and edges is rendered using a graph visualization library. In the current implementation, the SIFG component uses the JavaScript graph library *Cytoscape.js*. Furthermore, we use the library *dagre*[25] which implements different layout algorithms for directed graphs and which is already integrated in the Cytoscape.js library. The layout algorithm is configured to generate a graph whose general flow is directed from left to right.

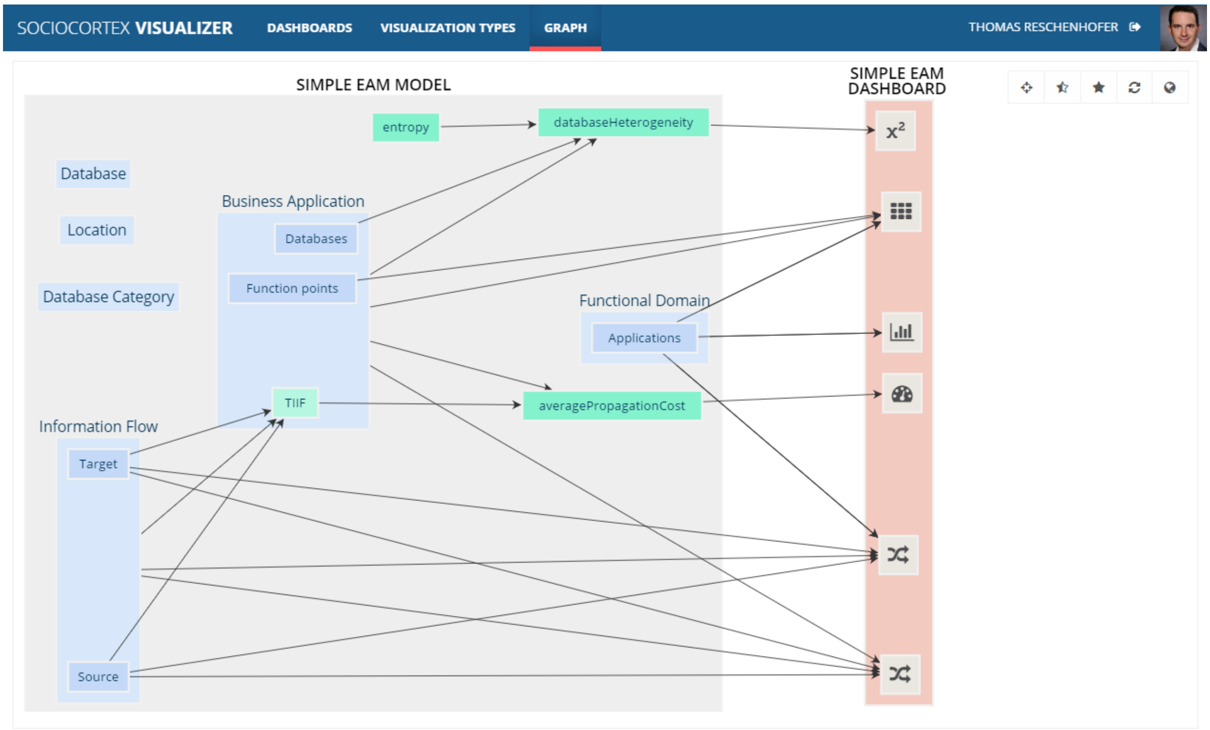


Figure 5.36.: The Social Information Flow Graph (SIFG) for the dashboard in Figure 5.30.

Figure 5.36 shows the Social Information Flow Graph for the exemplary dashboard in Figure 5.30. Dashboards are represented by red boxes, which in turn contain gray nodes referring to the dashboard's visualizations. Due to performance reasons as well as for the sake of visual clarity, visualizations are not rendered according to their visualization type, but only represented by their icon.

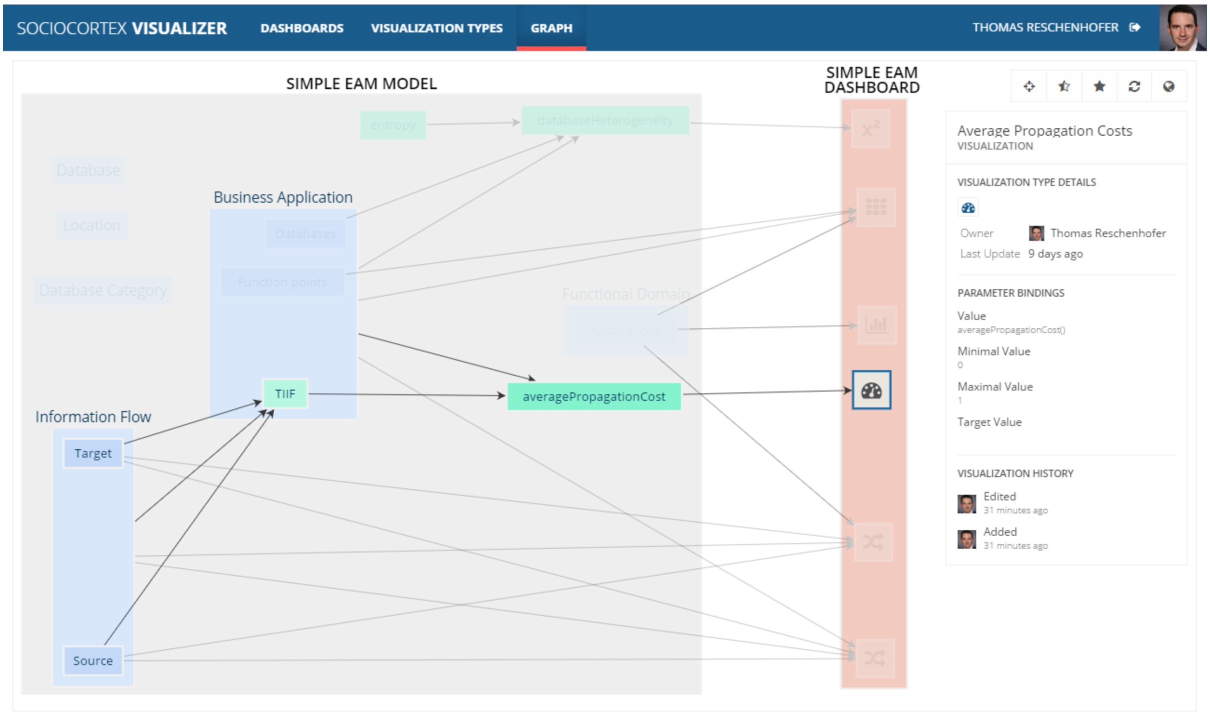
The big gray boxes represent workspaces. Within the workspace boxes, there are two different kinds of nodes, namely entity type nodes (light blue boxes) and nodes representing the workspace's custom functions (green boxes). An entity types again can have sub-nodes rep-

[25] `https://github.com/cpettitt/dagre`, last accessed on: 04.10.2016

resenting its attribute definitions (blue boxes) and derived attribute definitions (cyan boxes). Between nodes, directed edges represent an information flow which in turn is defined through a semantic dependency. This means that a semantic dependency from a node A to a node B is represented by a directed information flow edge from node B to node A.

The SIFG component supports basic interaction facilities for graph visualizations (Herman et al., 2000). For example, it supports *Geometric Zooming*, i.e., the user can zoom into the graph without changing the level of detail. Furthermore, the SIFG view supports pan movements, i.e., the user can simply change the perspective to the graph. By combining the pan feature with zooming, the end-user can focus an arbitrary section of the SIFG.

While initially the nodes of the graph are arranged automatically by a layout algorithm, they can be manually rearranged by the end-user. However, in its current state, the prototype is not able to store the view state and thus the positions of the nodes. This implies that reloading the SIFG leads to an automated rearrangement of the nodes and manual changes are lost. However, we admit that storing the view state would be a helpful feature. On the other hand, it also imposes new challenges to the implementation of the SIFG, e.g., restoring a view state if the set of nodes or edges changed.



Average Propagation Costs
VISUALIZATION

VISUALIZATION TYPE DETAILS

Owner Thomas Reschenhofer
Last Update 9 days ago

PARAMETER BINDINGS

Value
averagePropagationCost()

Minimal Value
0

Maximal Value
1

Target Value

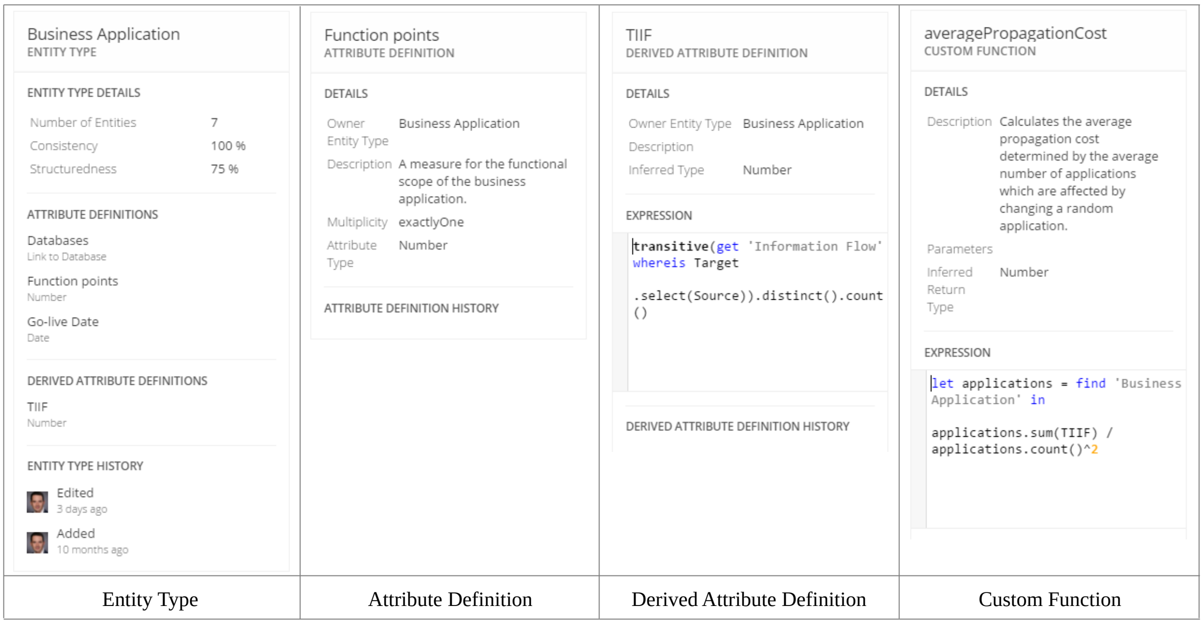
VISUALIZATION HISTORY

Edited
31 minutes ago

Added
31 minutes ago

Figure 5.37.: Selecting nodes allows users to explore their meta-data, and to identify information flow paths going through it.

In addition to those basic graph interaction facilities, the SIFG also supports features which enhance its usability and explorability as described by Heer and Boyd (2005). First of all, all nodes of the SIFG are clickable and selectable by the end-user. Selecting a node has two consequences:



Business Application
ENTITY TYPE

ENTITY TYPE DETAILS

Number of Entities	7
Consistency	100 %
Structuredness	75 %

ATTRIBUTE DEFINITIONS

Databases
Link to Database

Function points
Number

Go-live Date
Date

DERIVED ATTRIBUTE DEFINITIONS

TIIF
Number

ENTITY TYPE HISTORY

Edited
3 days ago

Added
10 months ago

Function points
ATTRIBUTE DEFINITION

DETAILS

Owner Entity Type	Business Application
Description	A measure for the functional scope of the business application.
Multiplicity	exactlyOne
Attribute Type	Number

ATTRIBUTE DEFINITION HISTORY

TIIF
DERIVED ATTRIBUTE DEFINITION

DETAILS

Owner Entity Type	Business Application
Description	
Inferred Type	Number

EXPRESSION

```
transitive(get 'Information Flow'
whereis Target

.select(Source)).distinct().count
()
```

DERIVED ATTRIBUTE DEFINITION HISTORY

averagePropagationCost
CUSTOM FUNCTION

DETAILS

Description	Calculates the average propagation cost determined by the average number of applications which are affected by changing a random application.
Parameters	
Inferred Return Type	Number

EXPRESSION

```
let applications = find 'Business
Application' in

applications.sum(TIIF) /
applications.count()^2
```

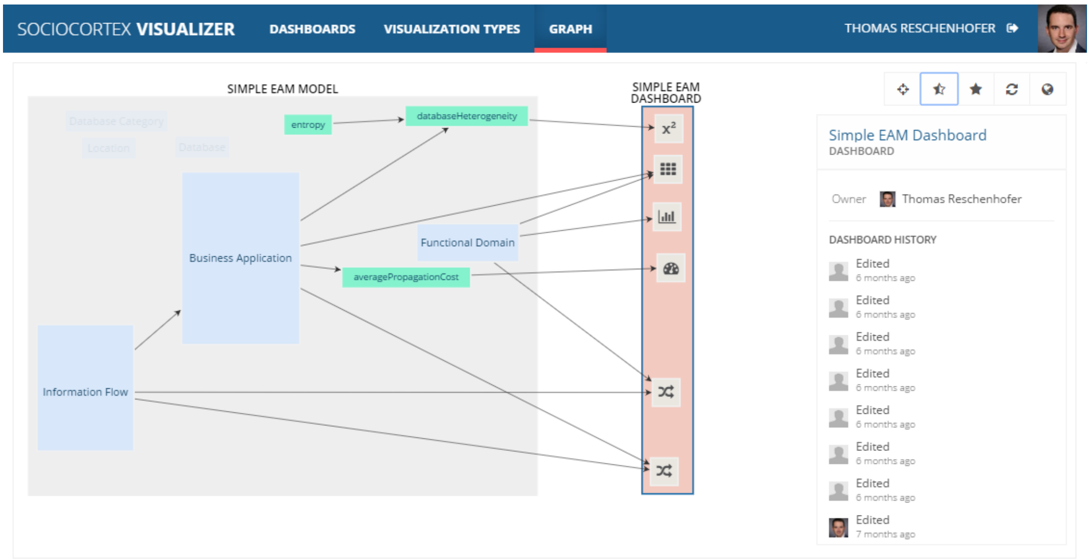
Entity Type | Attribute Definition | Derived Attribute Definition | Custom Function

Figure 5.38.: The exemplary content of the sidebar on selecting different kinds of nodes.

Displaying Meta-information The meta-data of the selected node is displayed in a sidebar on the right-hand side of the SIFG surface. The actual meta-information depends on the type of node which is selected. This means that in addition to generic node information (e.g., name, owner, and version history), the sidebar also includes node-specific meta attributes, e.g., a visualization's data bindings as shown in Figure 5.37. Figure 5.38 shows the content of the sidebar if another kind of node is selected. For example, on selecting an entity type, the sidebar includes the inconsistency and unstructuredness metrics as explained in Section 5.3.1. Furthermore, selecting an analytical abstraction (i.e., a custom function or a derived attribute definition) also shows the corresponding MxL expression. The intention of this is to support the user in understanding how the information flows to the selected node are actually derived.

Currently, the prototype implements a specific set of meta attributes for data model elements, analytical abstractions, and view elements. However, in order to adapt to new requirements, the set of meta attributes is easily extendable and displayable within the SIFG sidebar. This is particularly true for the view elements (dashboards, visualizations, and visualization types), since they are mapped to generic entities whose set of attributes is extendable at run-time.

Highlighting of Paths The direct and transitive predecessors and successors as well as edges from and to them are visually highlighted. This interaction feature empowers end-users to identify data model elements and analytical abstractions which serve as input for a particular element, and analytical abstractions and visualizations which are consuming a particular element. This information is helpful, e.g., to identify elements affected by changes, and subsequently to determine their owners and thus contact persons to interact and coordinate with (Cataldo et al., 2006). For example, Figure 5.37 highlights the information flow paths to the selected visualization. It not only highlights directly referenced elements (e.g., the custom function *averagePropagationCost*), but also transitively connected elements (e.g., the derived attribute definition *TIIF* as well as its input).



Owner Thomas Reschenhofer

Edited
6 months ago

Edited
6 months ago

Edited
6 months ago

Edited
6 months ago

Edited
6 months ago

Edited
6 months ago

Edited
6 months ago

Edited
7 months ago

Figure 5.39.: To reduce the visual complexity, container nodes (e.g., nodes representing entity types) can be collapsed.

In addition to the selectability of nodes, the SIFG supports the collapsing of container nodes to reduce the visual complexity of the graph and consequently to improve its explorability (Heer and Boyd, 2005). In the current implementation of the SIFG, collapsable container nodes are those representing workspaces, entity types, and dashboards. Collapsing a node means that its sub-nodes are removed from the view, and that all edges to and from the sub-elements are transformed to edges to and from the collapsing container node. For example, Figure 5.39 shows the SIFG with collapsed entity types, i.e., attribute definitions as well as derived attribute definitions are hidden. Information flow edges from and to those nodes are transformed to edges from and to the respective entity type nodes.

In the SIFG, nodes can either be collapsed individually by double-clicking them, or collectively by selecting a general level of detail to be shown through the buttons at the top right corner of the SIFG view.

CHAPTER 6

Evaluation

In Chapters 4 and 5, we elaborated on the conceptual and UI design of a tool empowering end-users to collaboratively analyze evolving complex linked data. Based on a prototypical implementation, we conducted multiple kinds of empirical evaluations to assess different aspects of our contribution. The purpose of the evaluation is manifold: On the one hand, it lays the foundation for the refinement of our conceptual contribution in future work. On the other hand, it serves as an assessment of both the validity of the developed artifacts and the utility and usability of the prototypical implementation.

Wohlin et al. (2012) outline that empirical strategies specifically include experiments, surveys, and case studies. The purpose of experiments is to get an understanding of relationships between specific factors, e.g., if a specific change of a software tool's UI improves the performance in completing a specific task. By contrast, surveys and in particular interviews intend to gather information on the knowledge, preferences, and behavior of individuals (Fink, 2016). As a complementary empirical strategy, case studies allow researchers to observe a software tool within a practical environment, and to study its performance in real-life situations (Yin, 2014).

An empirical evaluation should combine those strategies to be considered as convenient classification (Shull et al., 2001; Wohlin et al., 2012). Furthermore, Patton (2002) and Yin (2014) argue to use multiple sources of evidence for deriving conclusions in the context of an empirical study. As a consequence, we chose to perform different kinds of empirical strategies as part of this thesis' evaluation:

- An (online) experiment among different user groups to assess the usability and utility of the UI for defining complex queries.
- An interview series to gain to assess the usability and utility of the SIFG, and to identify concrete practical use-cases which are addressable by it.

- Three case studies multiple case studies to get feedback from applying the prototype in practical environments.

In this chapter, we describe those three parts of the evaluation in detail. In Section 6.1, we describe the experiment as an evaluation of the MxL query editor's usability and the augmented model view's utility. Furthermore, Section 6.2 summarizes the results of interviews about the Social Information Flow Graph (SIFG) and particularly the potential benefits of the SIFG as identified by the interviewees. Finally, in Section 6.3, we elaborate on the evaluation of the SC Visualizer in the context of three case studies, and highlight obtained key findings. In this sense, we evaluate how our prototype performs in a practical environment.

6.1. Evaluation of the UI for Formulating Complex Queries

The query formulation UI represents an integral part of our conceptual contribution and prototypical implementation. As discussed in Chapters 1 and 2, current EUA tools do not support the formulation of complex queries based on evolving complex linked data. Therefore, we developed MxL (cf. Section 4.2.2) and designed respective UI components (cf. Section 5.2.3) to empower and support end-users to define such complex queries.

Particularly in the light of data model evolution, familiarization with the data model is an important task and essential prerequisite to define complex queries. As described in Section 5.2.3, we developed a model view UI component which can be used in conjunction with the MxL query editor. By this, the augmented model view adds contextual information to the data model-specific auto-completion hints and thus hypothetically facilitates the end-user's familiarization with the underlying data model. In order to support this hypothesis, we evaluated the usability and utility of the augmented model view in an online experiment.

The goal of the experiment is not to derive statistically significant conclusions. On the contrary: We are mostly interested in qualitative feedback and assessments from the experimentees. These findings can serve as input for the further development and improvement of the conceptual design and prototypical implementation of a UI for defining complex queries on evolving data structures.

Initial results of this evaluation were already published in preliminary work (Reschenhofer and Matthes, 2016b). However, in the following sections we describe the setting and results in more detail and including more responses.

6.1.1. Experiment Setting

The evaluation of the MxL code editor was performed as a technology-oriented online experiment (Wohlin et al., 2012). For this purpose, we developed a dedicated web application implementing the experimentation logic. The URL of this web application as well as detailed instructions was sent by mail to a selected set of experimentees. Figure 6.1 summarizes and illustrates the general structure and process of the experiment.

In order to shorten the introduction phase to the MxL query editor and thus to emphasize the

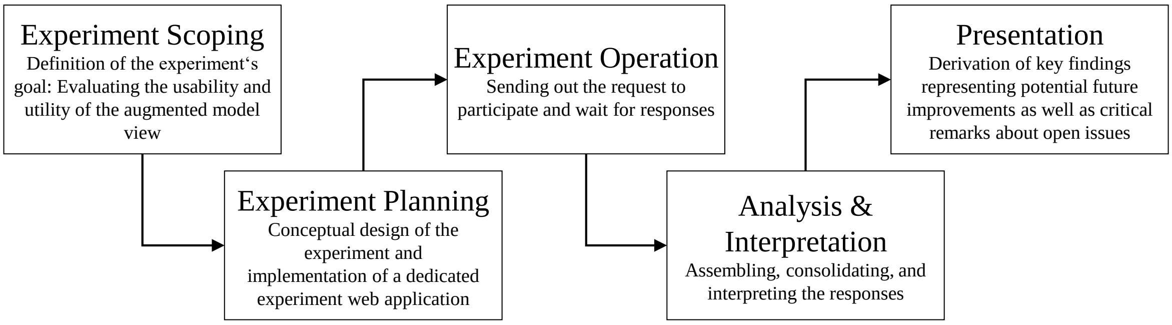

Experiment Scoping
Definition of the experiment‘s goal: Evaluating the usability and utility of the augmented model view
Experiment Planning
Conceptual design of the experiment and implementation of a dedicated experiment web application
Experiment Operation
Sending out the request to participate and wait for responses
Analysis & Interpretation
Assembling, consolidating, and interpreting the responses
Presentation
Derivation of key findings representing potential future improvements as well as critical remarks about open issues

Figure 6.1.: An illustration of the experiment process as defined by Wohlin et al. (2012).

augmented model view as main subject of the evaluation, we have chosen only individuals which already know either the current or a previous version of MxL. The set of experimentees to which we sent a request for participating in the experiment included 17 individuals of which 11 finished their tasks and provided a response. Table 6.1 provides an overview including a job description and a qualitative self-assessment of their knowledge in MxL. The knowledge level ranges from *low* to *high* and is described as follows:

Low The person knows MxL and its basic concepts, but never wrote an MxL expression (before participating to this experiment)

Medium The person already defined some basic MxL expressions in the past, e.g., simple arithmetic operations

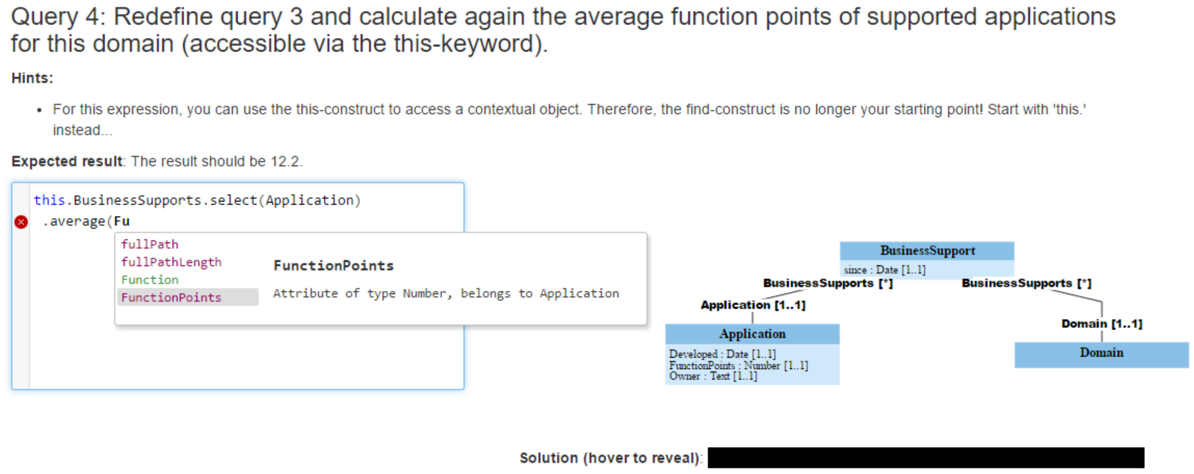
High The person already defined complex MxL queries based on complex linked data structure

The experiment itself consists of three steps implemented within a web application and thus doable in the browser:

1. In the first step, the experimentee is asked to define four specific MxL queries. After the third query, a change of the underlying data model is simulated. For each query, we provide a brief description and helpful hints as well as a hidden exemplary solution which can be uncovered on demand. In this step, the augmented model view is not yet provided.
2. In a second step, the experimentee has to define four analogous queries on an isomorphic data model. In this context, isomorphic means that the data models of those cases only differ in names of types, attributes, and relations. This time, the augmented model view is shown beside the MxL query editor.
3. In the final step, the experimentee has to fill out a questionnaire. This questionnaire includes questions about how they familiarized themselves with the underlying data model, and if they felt that the augmented model view is helpful in this process. More specifically, we asked them if they were able to formulate all queries with and without the augmented model view. In the final part of the questionnaire, we wanted to get feedback about (potential) current weaknesses and desirable improvements of the query formulation UI in general, and the augmented model view in particular.

Code	**Case**	**Self-Assessed Knowledge in MxL**	**Role and Company**
Practitioners			
P1	A	high	Enterprise Architect in a Swiss Bank (10,001+ employees)
P2	B	medium	Enterprise Architect in a German Bank (5,000 - 10,000 employees)
P3	A	low	Enterprise Architect in a German logistics company (10,001+ employees)
P4	B	low	Enterprise Architect in a German IT services provider (5,000 - 10,000 employees)
IS Researchers			
R1	A	low	Research associate at the Chair of Software Engineering of Business Information Systems (sebis)
R2	B	high	
R3	B	medium	
R4	B	medium	
Computer Science Students			
S1	B	medium	Former student doing a thesis at the sebis chair
S2	B	medium	
S3	A	high	

Table 6.1.: A list of experimentees which finished their tasks and provided a feedback through a questionnaire.



Query 4: Redefine query 3 and calculate again the average function points of supported applications for this domain (accessible via the this-keyword).

Hints:

- For this expression, you can use the this-construct to access a contextual object. Therefore, the find-construct is no longer your starting point! Start with 'this.' instead...

Expected result: The result should be 12.2.

```
this.BusinessSupports.select(Application)
 .average(Fu
```

```
fullPath
fullPathLength
Function
FunctionPoints
```

FunctionPoints

Attribute of type Number, belongs to Application

Solution (hover to reveal): [redacted]

Figure 6.2.: An exemplary challenge presented to the participants of the online experiment.

Figure 6.2 shows a screenshot of an exemplary query formulation challenge within our dedicated experiment web application. In this case, the augmented model view is visible and usable for formulating the query.

The data models of the both steps are isomorphic in order to ensure the comparability of the difficulty of queries. As a consequence, experimentees might perceive the formulation of queries in the second step as significantly easier regardless of the augmented model view, since they already defined analogous queries in the first step. For this reason, we divided the set of experimentees into two groups: Group A performs the three steps of experiment in the aforementioned order, i.e., in the first step without the augmented model view, and in the second step with. On the contrary, in group B the first two steps are exchanged, i.e., the augmented model view is shown in the first step rather than the second one.

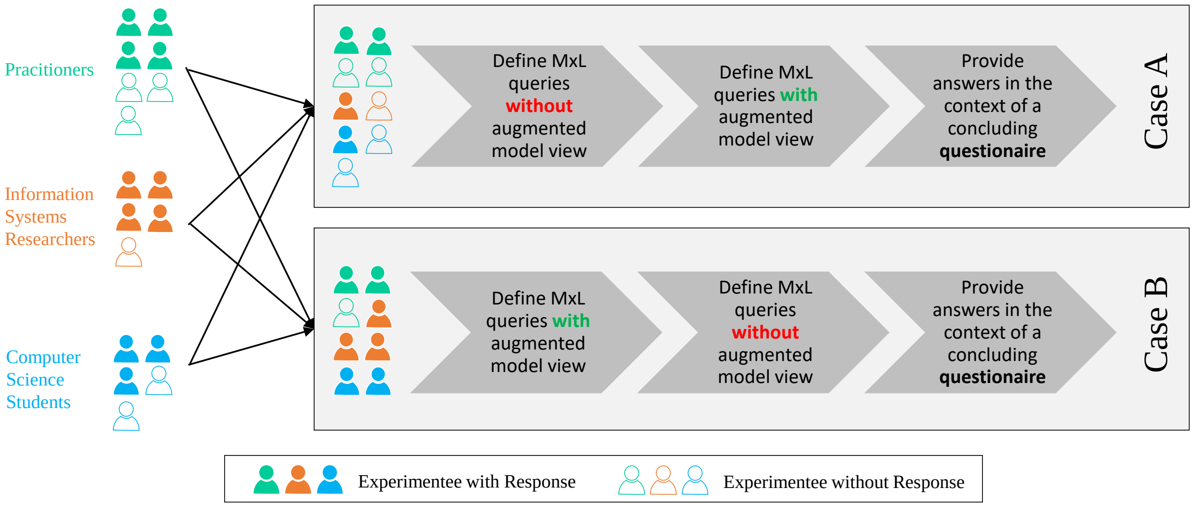

Pracitioners
Information Systems Researchers
Computer Science Students
Define MxL queries without augmented model view
Define MxL queries with augmented model view
Provide answers in the context of a concluding questionaire
Case A
Define MxL queries with augmented model view
Define MxL queries without augmented model view
Provide answers in the context of a concluding questionaire
Case B
Experimentee with Response
Experimentee without Response

Figure 6.3.: An illustration of the allocation of experimentees to two different experiment cases. The cases differ in the order of steps one and two.

Figure 6.3 illustrates the setting of the experiment. The allocation of the experimentees to the cases was manually done based on the type (practitioner, researcher, student) and our assessment of the experimentee's knowledge level. By this, we tried to reach a balanced allocation with each group having a comparable profile of experimentees. However, due to different response rates in groups A and B, we got more responses for the latter (seven) than for the former (four) group.

6.1.2. Key Findings of the Experiment

After the scoping and planning stages of the experiment (Wohlin et al., 2012), we sent out emails to the selected people including comprehensive instructions as well as the allocation to the respective experiment case (A or B). Following a time period of three months, we collected all received responses. As illustrated in Figure 6.3, 11 persons out of 17 contacted ones completed the tasks and provided an answer to the questionnaire.

By consolidating and interpreting the individual responses, we were able to extract key findings.

In the following, we summarize them and add direct quotes from experimentees, whereas we refer to them by their code as defined in Table 6.1:

The augmented model view is helpful for formulating queries All but one respondents explicitly stated that the formulation of queries is significantly easier and/or faster with the augmented model. They also mentioned that it is particularly helpful in situations in which users are not familiar with the underlying data model, the data model is subject to frequent changes, or the data model is not easily accessible.

The aspect of the augmented model view which was found to be the most useful one is that it helps users to retain the orientation when navigating through the data model. For example, experimentees stated that "you can see if your expression navigates through the data model like you expect it" (P2), "the model view helps to understand the current position within the model and makes it simpler to query without knowing the model in detail." (R3), and that formulating queries with the augmented model view is "a lot easier, since you can see the correct attribute names and reference names" (S1).

P1 argued that "In this case, the difficulty was the same" when comparing the formulation of queries with and without the augmented model view. However, he added that he "could imagine that it is more helpful when data models are bigger, e.g. more than ten classes and relationships.".

Showing only a local view has to be reconsidered The current implementation of the augmented model view only shows a local excerpt of the data model which only includes directly referenced and adjacent model elements. The intention of this feature was to reduce the visual complexity of the model view and to emphasize navigable data model elements (cf. Section 5.2.3).

However, six respondents noted that showing the full data model would be more helpful than showing only a local excerpt. Experimentee R3 argues that the local view makes it "difficult to keep track of the big picture". However, they also admitted that in case of larger data models, showing only a local perspective on the data model might be useful to hide irrelevant parts. Experimentee R2 suggests to make a compromise and to make this behavior configurable by the user.

The augmented model view should be better integrated with the textual code editor The augmented model view as of today is automatically generated based on the expression in a connected MxL code editor, i.e., changing the input of the code editor automatically updates the augmented model view.

However, three experimentees would like to have an even better integration of those two components. For example, experimentee S3 suggests to highlight elements within the augmented model view which are marked within the code editor, or which are currently focused in the list auto-completion hints. According to S3, this would "make it easier for users to navigate data models and get feedback for their actions".

Furthermore, P1 and P4 argue for implementing a bi-directional integration, i.e., that not only the code editor serves as input for the generation of the augmented model view, but also vice versa. For example, P1 "could imagine that [he] writes average and then simply clicks within the model to select the attribute which should be used".

The augmented model view should be more interactive In addition to a deeper integration of the code editor and the augmented model view, two experimentees also ask for more interactivity of the augmented model view component: For example, P4 suggests to enable end-users to rearrange model elements within the model view by an intuitive Drag&Drop mechanism. Furthermore, S3 would like to expand and collapse types and their attributes manually by double-clicking them. This would empower end-users to manually expand the local view.

The augmented model view might suffer from scaling issues On a critical note, three experimentees mention that they assume scaling issues in case of larger data models. Indeed, experimentee S2 already had problems with the example models, i.e., "the model view was too small. It was not possible to read all the multiplicities and names for relationships". Obviously, larger data models as they appear in practice, e.g., in EAM (Störrle, 2011), would only aggravate this issue.

Users still have to know the syntax of MxL The main purpose of the augmented model view is the support for ad-hoc familiarization with the underlying data model. Naturally, it does not help users to learn the query language MxL and its syntax and semantics. However, this was still mentioned negatively by four respondents. Their consensus is that MxL queries and their definition with a textual query editor is "a little bit too complex for a not-trained / not-skilled person" (P3). Specifically, P1 stated that "the problem [when formulating queries] was a combination of how to navigate and how to combine the functions". Together, the code editor and augmented model view component have to provide a more extensive documentation. For example, experimentee P4 proposes an improvement of the auto-completion mechanism and to implement a filter facility to enable the selective search for specific functions.

In summary, the results of the experiments suggest that formulating queries within a textual editor is easier and/or faster with the augmented model view (as it is today) than without it. Nevertheless, most responses from experimentees include recommendations to reconsider certain design decisions (e.g., showing a local view instead of the full data model), or to implement further UI features to improve the usability and utility of the code editor and the augmented model view.

Furthermore, the scalability of the augmented model view with respect to the size of the data model is an open issue which has to be addressed in future work. First, further evaluation has to identify the concrete pain points when trying to formulate queries on large data models. Based on those findings, further research can propose concrete solution approaches addressing those issues.

Finally, while the augmented model view was assessed to be helpful by nearly all experimentees, the technical and textual nature of defining queries with MxL is considered to be too complex for casual end-users. For non-technical end-users, they suggest to implement a visual query editor which provides more guidance by the implementing software. Nevertheless, the experimentees still argue that the textual query editor would be the preferred UI for the formulation of complex queries by expert users.

6.2. Evaluation of the Social Information Flow Graph

In a collaborative environment, identifying the right person to interact and coordinate with is one of the most challenging tasks (Cataldo et al., 2006). Addressing this issue, we developed the Social Information Flow Graph (SIFG), whose conceptual design is described in Section 4.4, and whose prototypical implementation as part of the SC Visualizer was outlined in Section 5.3.2.

The purpose of the SIFG is to provide a means to end-users empowering them to interactively explore which elements and co-workers are affected by one's interaction with the EUA system. For this, we implemented several UI features in order to improve the usability of our prototype and to facilitate the explorability of the SIFG.

In this section, we outline the highlights of the evaluation of the SIFG through demonstrating the prototype to multiple practitioners followed by a semi-structured interview. The goal of the evaluation is two-fold: On the one hand, we gathered feedback on weaknesses and strength of the presented version of the prototype as well as potential improvements for future development. On the other hand, we also wanted to provide a proof of concept and to find out concrete use-cases which are addressable by the SIFG in the interviewee's opinion.

Highlights of the evaluation's findings were already published in a preliminary work (Reschenhofer et al., 2016b), while details of the evaluation are described in the master's thesis of Bürgin (2015).

6.2.1. Setting of the Interview Series

The methodology applied for the evaluation of the SIFG is an exploratory case study including three multiple cases (Yin, 2014). Figure 6.4 summarizes and illustrates the general structure and process of the experiment.

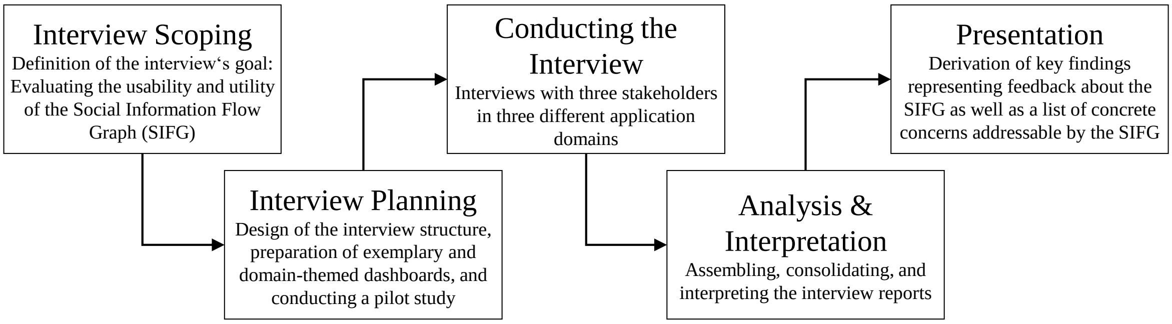

Interview Scoping
Definition of the interview‘s goal: Evaluating the usability and utility of the Social Information Flow Graph (SIFG)
Interview Planning
Design of the interview structure, preparation of exemplary and domain-themed dashboards, and conducting a pilot study
Conducting the Interview
Interviews with three stakeholders in three different application domains
Analysis & Interpretation
Assembling, consolidating, and interpreting the interview reports
Presentation
Derivation of key findings representing feedback about the SIFG as well as a list of concrete concerns addressable by the SIFG

Figure 6.4.: An illustration of the case study's structure.

As a preliminary pilot study, we first evaluated the prototype with researchers of the sebis chair. In this context, the subjects of the study used the prototype while being monitored by us, and subsequently provided respective feedback, e.g., regarding its usability.

The main study was done in cooperation with three of our industry partners. Thereby, we identified three different application domains in three different companies. In each of those cases, respective knowledge workers already use a BI or EUA tool for collaboratively analyzing and visualizing domain-specific data in an end-user-driven way. Therefore, creating awareness of who is doing what is an issue in those cases.

In order to get feedback about how the SIFG could support knowledge workers in those domains, we conducted interviews with users which are responsible for the respective cases. The application domains and corresponding contact persons are:

Enterprise Architecture Management (EAM) The EAM domain was already introduced in Chapter 1. As a brief summary, EAM is a discipline to plan, develop, and control an Enterprise Architecture (EA). It involves multiple stakeholders to gather, model, and visualize EA data.

The interviewee in this case was an enterprise architect with over nine years of professional experience in his role. He currently manages the internal architecture of a German IT services provider (5,000 - 10,000 employees). Currently, his team uses a mixture of prevalent EA tools (Matthes et al., 2008) as well as prevalent spreadsheet software.

Financial Risk Management In the domain of financial services, risk management companies generate reports about financial scenarios including a plethora of parameters, e.g., interest rates and stock prices. Through the analysis and assessment of those scenarios, financial risk management seeks to minimize various risks when making decisions in the financial market by automatically generating recommendations.

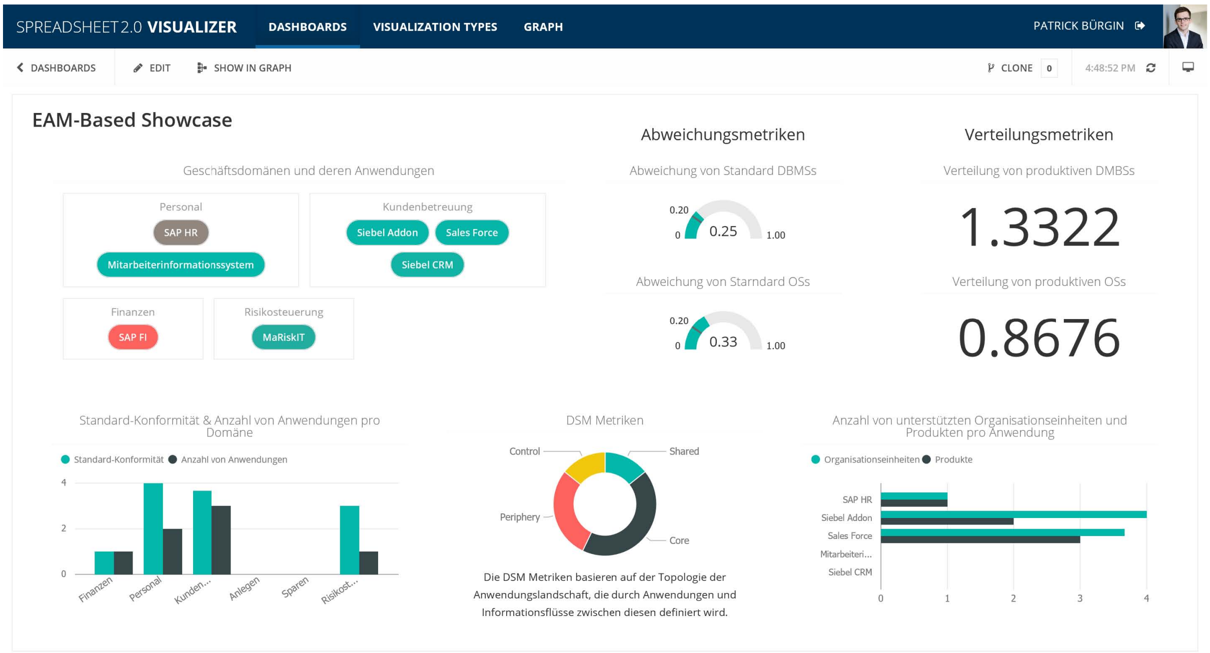
In this domain, we interviewed the IT Infrastructure Manager of a German investment company (10,001+ employees). In his team, co-workers primarily use prevalent spreadsheet software as well as numerical computing environments like MATLAB[1] in order to collaboratively define and generate financial reports.

Data Quality Management (DQM) This domain is about roles, responsibilities, and processes regarding the acquisition and maintenance of data. Data quality typically not only refers to the completeness of data, but also to its accuracy and timeliness. The purpose of DQM is to ensure a certain degree of data quality and thus to reduce the risk of wrong and costly decisions which are made based on wrong or incomplete data (Shankaranarayanan and Cai, 2006).

The interview partner in this case was the data quality manager of a German logistics company (10,000+ employees). Currently, her team uses primarily prevalent spreadsheet software to acquire and maintain roles and processes for different kinds of data (sources).

Before conducting each actual interview, we prepared an exemplary dashboard which implements a domain-themed scenario. Consequently, we developed an exemplary EAM dashboard,

[1] `https://www.mathworks.com/products/matlab/index.html`, last accessed on: 04.10.2016



Die DSM Metriken basieren auf der Topologie der Anwendungslandschaft, die durch Anwendungen und Informationsflüsse zwischen diesen definiert wird.

Figure 6.5.: An exemplary dashboard implementing an EAM use case which was demonstrated to an industry partner (Bürgin, 2015).

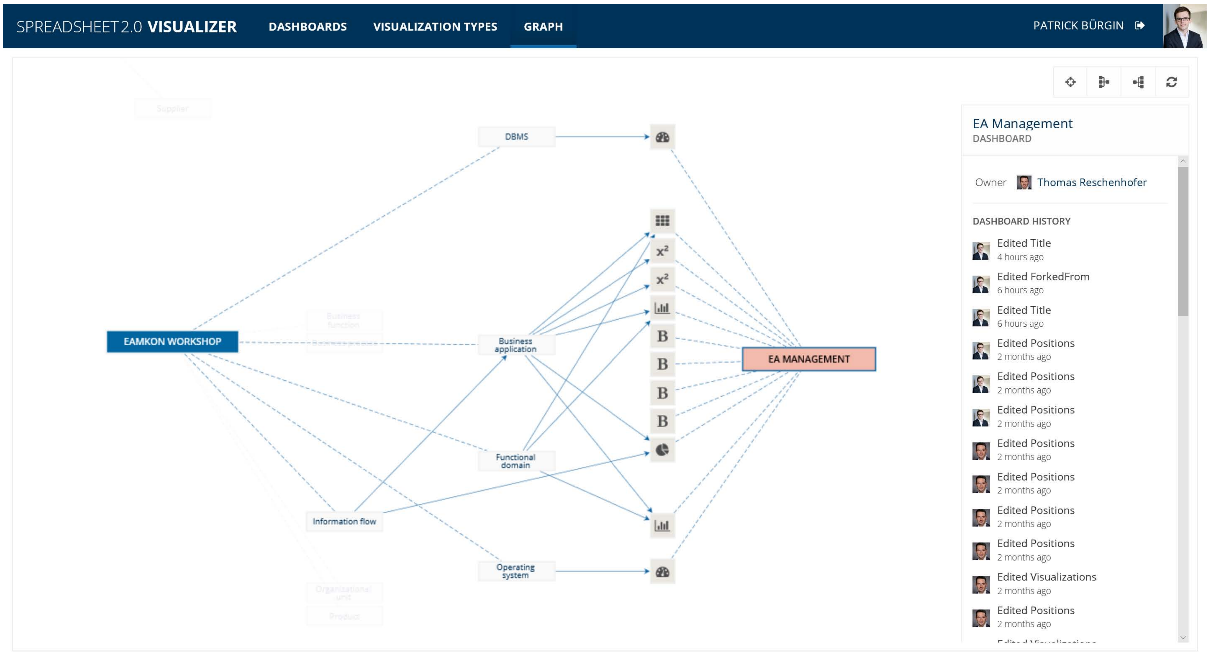


Figure 6.6.: The SIFG for the exemplary dashboard in Figure 6.5 (Bürgin, 2015).

an exemplary financial services dashboard, and an exemplary DQM dashboard. For example, Figure 6.5 shows the EAM dashboard, and Figure 6.6 the derived SIFG view.

It is worth noting that the SIFG presented to the interviewees is an early version compared to the prototype described in Section 5.3.2. For example, the version presented in the context of this case study did not support composite nodes, i.e., sub-element relationships were represented by a certain kind of edge. Furthermore, analytical abstractions were not shown in the previous version of the SIFG. Only based on the feedback gained through the case study as described in the current section, we were able to improve the SIFG and to come up with the version as described in Section 5.3.2.

The interview itself is based on the qualitative and responsive style as defined by Rubin and Rubin (2011). This means that the interview primarily consisted of open-ended queries to ensure flexibility for both the interviewee and the interviewer. Nevertheless, all three interviews followed a common structure consisting, i.a., of the following steps (Bürgin, 2015):

1. General questions, e.g., addressing the role of data analysis and visualizations in the interviewee's organization and application domain.
2. Brief introduction to and explanation of the dashboard and visualization type concepts as implemented by the respective version of the SC Visualizer. The introduction and explanation is based on the domain-specific example dashboard which was prepared beforehand.
3. Brief introduction to and explanation of the Social Information Flow Graph (SIFG) as implemented by the respective version of the SC Visualizer. Again, the introduction and explanation is based on the domain-specific example dashboard and the SIFG view derived from it.
4. Extensive discussion of the SIFG, its strength and weaknesses, desirable improvements, as well as concrete concerns which would be addressable by applying it in the respective application domain.

6.2.2. Technical Feedback Gained from the Interview Series

The key findings of the interview series are two-fold: One the one hand, we got feedback regarding the technical features of the SIFG view. On the other hand, the interviewees described concerns they typically face in their application domain and which are potentially addressable by the SIFG.

The former kind of key findings represents rather technical feedback and serves suggestions for improvements of the prototype. Based on the consolidation of the reports of all three interviews, we can present the following set of key findings referring to technical aspects of the SIFG:

Different kinds of edges are distracting As shown in Figure 6.6, the initial version of the SIFG view implemented part-of relationships by dashed edges. For example, the entity type *Information flow* is part of the workspace *EAMKON Workshop*, which is represented by a dashed line between them. However, when trying to explore information flows, those additional edges tend to distract users, although edges representing information flows are designed slightly differently. For this reason, we opted for an alternative implementation of part-of relationships, namely as composite nodes (cf. Section 5.3.2).

Collapsing and expanding nodes is a useful feature Although not yet implemented as composite nodes, elements representing parent or container nodes of other elements were already collapsible and expandable in the presented version of the SIFG. This feature was explicitly labeled as "very helpful" by two interviewees.

More details about the data model elements have to be visible in the SIFG The presented version of the SIFG view only visualized workspaces and entity types as data model elements, but did not include attribute definitions. The interviewees considered this reduced view as helpful to obtain an overview over the general data structures. However, in a next step, the interviewees would expect to be able to drill down into specific data structures in order to make it explorable by the end-user. As a consequence, we included attribute definitions as sub-elements of entity types in the current version of the SIFG view (cf. Section 5.3.2).

Analytical abstractions have to be part of the SIFG Similar to the previous point, interviewees asked for more details about how the data elements represented by entity types are bound to the visualizations. More specifically, they "would have liked to see the data transformations" between the actual data model elements and the visualizations. For this reason, we improved the SIFG and integrated analytical abstractions, i.e., derived attribute definitions are shown as sub-elements of respective entity types, while custom functions are displayed as sub-nodes of a workspace container (cf. Section 5.3.2).

The visibility of information within the SIFG has to be controllable One interviewee voiced his concern about privacy issues, since the SIFG and the underlying system tracks all user's interactions with the system's elements and makes them visible to co-workers. He suggested that "the system should provide sufficient access control capabilities" in order to specify which interactions of which users should be tracked by the system and displayed within the SIFG.

The SIFG has to integrate more information about users Contrary to the privacy concern mentioned by the aforementioned interviewee, another one asked for even more user- and interaction-related information. In the SC Visualizer's version as presented to the interviewees, the SIFG showed owners of elements as well as version histories also capturing users. However, further user-related meta-information should be potentially displayable in the SIFG, e.g., users of the elements, or users which are potentially interested in a certain element.

The SIFG view should offer extensive configuration mechanisms On a general note, the interviewees stated that the SIFG is potentially able to address many concerns (as described later on in this section). However, the SIFG "view should depend on the question at hand", i.e., different concerns require users to focus on different aspects of the SIFG. For example, one interviewee mentioned that filtering the set of shown nodes and edges by different aspects would allow to configure the SIFG and to tailor it to specific use cases.

Based on the technical and UI-related feedback gained through the interviews, we improved the UI design and prototypical implementation of the SIFG and its integration into the SC

Visualizer. For the case studies described in Section 6.3, we were already able to provide the updated version as described in Section 5.3.2.

6.2.3. Identified Concerns Addressable by the SIFG

While one goal of the interview series was to evaluate the SIFG and its prototypical implementation in order to lay the foundation for its refinement, the second goal was to identify concrete concerns which typically occur in the interviewee's application domain, and which are potentially addressable by the SIFG. In this context, the interviewees should assume that the aforementioned suggestions for improvements are implemented. Therefore, the discussions with the interviewees were hypothetical ones. Nevertheless, the results of the discussions and the identified concerns which are addressable by such an improved SIFG are still of interest, since they can drive the further development of the SIFG by tailoring it to certain use cases.

Analogous to the suggestions for improvement, we consolidated the identified concerns from each individual interview (Bürgin, 2015). Based on this, we derived the following set of general concerns which are potentially addressable by the SIFG (Reschenhofer et al., 2016b):

Usage Analysis In all three studied application domains, the interviewees agreed that "the collection of data induces the highest cost in our daily business". As a specific example, one of them explains that "the more [he] can learn about who uses certain visualizations [etc], the more [he] can infer about the importance of model elements".

Although the current prototype does not yet support the tracking of users of visualizations or other elements and thus the SIFG is not yet able to display this information, it already provides means to explore information flows between elements. Adding usage information to nodes would instantly enable the tool-supported derivation of the importance of maintaining certain data elements. As a consequence, the SIFG would enable the optimization of the data maintenance process.

Stakeholder Identification The interviewees stated that the SIFG empowers end-users to identify stakeholders which are related to certain data model element or analytical abstraction. This would enable them to proactively contact users which explicitly expressed their interest in related elements (e.g., by a "follow/subscribe mechanism") and to ask them "whether they want to participate" or "discuss possible changes".

Again, the current SIFG prototype does not support such a follow/subscribe mechanism. Nevertheless, it does provide facilities to facilitate the exploration of potential stakeholders if such a feature would be implemented.

Impact Analysis The enterprise architect among the interviewees described situation in which users ask him to change data model elements since they are "afraid to break something". The problem is that users just do not know which parts of the system are affected by changing a specific element. The SIFG promotes transparency with respect to which elements are semantically related to each other. In this way, the SIFG "would certainly help to make the models and relations more transparent", as stated by one of the interviewees.

Support for Data Provenance Sharing data among colleagues is "part of the daily business" of the interviewees. As a consequence, it is "interesting to know what the story behind a given information asset is". Specific questions related to the provenance of data are "Where does the data come from?", "Who worked on it?", and "When was the last change and by whom?". Providing an environment to visually explore the SIFG may reduce the "significant operative cost" induced by struggles when identifying co-workers that faced similar problems in the past.

Addressing Compliance Demands Driven by compliance demands, organizations particularly in the financial domain have to be able to reveal the full calculation process behind a given measure or visualization. The drivers are both of an internal (avoid reputational risk) and external (financial regulations) nature. For example, even prevalent spreadsheets are subject to legal regulations if used for financial reporting (Panko and Ordway, 2005). The SIFG can support responsible analysts and auditors to shed light on how metrics and visualizations are computed and generated.

On a related note, the increased level of transparency obtained by using the SIFG can also be used to introduce new hires to the environment.

Support for Data Consolidation The interviewee of the DQM domain stated that the SIFG would be helpful to initiate the consolidations of dashboards and visualizations which consume overlapping information, i.e., information which is visualized in different visualizations and dashboards. Furthermore, the same information can be used to automatically calculate "similar" dashboards and to recommend them to end-users.

In summary, we were able to identify a set of concerns which typically occur in practice, and for which the interviewed practitioners are still seeking respective tool-support. However, while those concerns are potentially addressable by an improved version of the SIFG, this still has to be tested in a separate evaluation.

Furthermore, there are general open issues: First and foremost, the interviewees stated that data models and the set of analytical abstractions are usually larger than implemented in the exemplary dashboards. However, scalability and performance of tools are important aspects which have to be met by them, and which are not addressed by the SC Visualizer.

Another issue is related to privacy: All interviewees explicitly mentioned that their work councils "would have major worries if users can explore all activities of their co-workers". This confirms critical observations of related studies in the field of Social Network Analysis (SNA) which also outline ethical and privacy issues (Borgatti and Molina, 2003).

6.3. Evaluation of the Prototype in Practical Environments

In Sections 6.1 and 6.2, we evaluated the usability and utility of specific components of our prototype and its design. In contrast, the purpose of the case study as described in the following section is to observe how the prototype as a whole performs in a practical environment. In this sense, the prototype is applied in different companies as a tool supporting ad-hoc EA analysis. Thereby, practitioners define analytical abstractions, visualizations, and dashboards based on organization-specific data, and subsequently report on their experiences.

6.3.1. Case Study Design and Setting

The methodology applied for the evaluation of the prototype in a practical environment is case study research following a multiple-case holistic design (Yin, 2014). This means that there is a single unit of analysis which is studied in different cases.

In order to identify potential candidates for the practical contexts for the evaluation of our prototype, we established the *SocioCortex Community* with a corresponding workshop series starting in December 2015 at the Technical University of Munich (TUM). Thereby, we invited enterprise architects and IT specialists from more than 20 Austrian, German, and Swiss companies from different sectors, e.g., finance, health care, and IT services. In average, about 15 community members attended to the SocioCortex Community workshops which were organized on a quarterly basis.

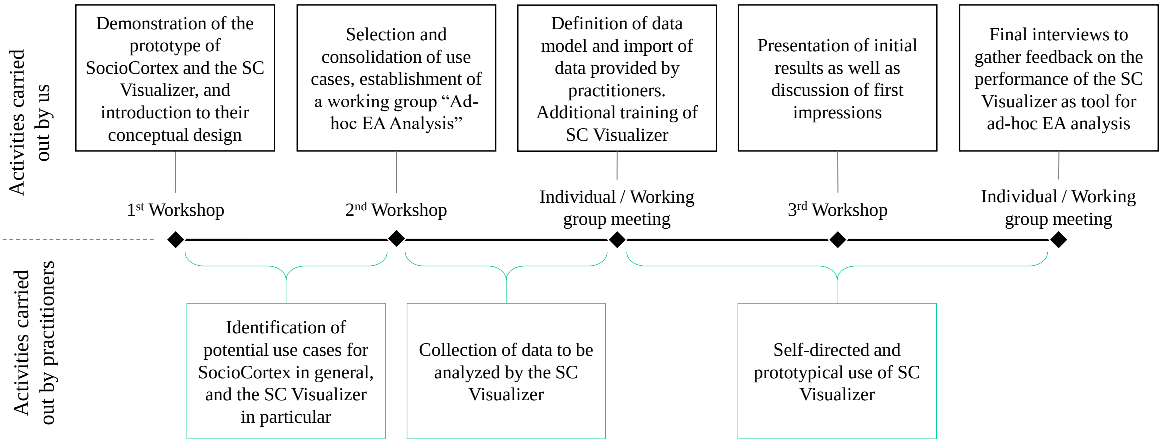

Activities carried out by us
Demonstration of the prototype of SocioCortex and the SC Visualizer, and introduction to their conceptual design
Selection and consolidation of use cases, establishment of a working group "Ad-hoc EA Analysis"
Definition of data model and import of data provided by practitioners. Additional training of SC Visualizer
Presentation of initial results as well as discussion of first impressions
Final interviews to gather feedback on the performance of the SC Visualizer as tool for ad-hoc EA analysis
1st Workshop
2nd Workshop
Individual / Working group meeting
3rd Workshop
Individual / Working group meeting
Activities carried out by practitioners
Identification of potential use cases for SocioCortex in general, and the SC Visualizer in particular
Collection of data to be analyzed by the SC Visualizer
Self-directed and prototypical use of SC Visualizer

Figure 6.7.: An illustration of how the case study was prepared and operated through a series of workshops and individual meetings.

Figure 6.7 illustrates the sequence of organized workshops and intermediary meetings. In the kick-off workshop, we introduced to the workshop in general and described its intention. In addition to the identification of concrete application areas for our prototype, another purpose of the workshop is to empower partners from industry and research to contribute to its development. Subsequently, we elaborated on the technical platform and the current state of the prototype including the SC Visualizer, and demonstrated it in form of a live-demo. We concluded the kick-off workshop by asking the participants to ponder about potential use cases for SocioCortex and the SC Visualizer which could be discussed in the next meeting.

In the second workshop, we discussed and assembled use cases for our prototype which were identified and proposed by the workshop participants. In a next step, we consolidated the use cases and prioritized them based on the number of participants which are interested in it and willing to contribute it in the form of a specialized working group. As a result of this approach, we identified and defined the working group "Ad-hoc EA Analysis" three industry partners as its primary contributors. The goal of this working group was to prototypically implement the

SC Visualizer as a tool empowering end-users to analyze EA models in an ad-hoc manner. For this purpose, we provided the SC Visualizer as an online service.

To assist the members of the "Ad-hoc EA Analysis working group, we provided online tutorials and documentations, and conducted several web conferences and phone calls either with the whole working group or individual members of it. Thereby, we explained them in detail how to use the SC Visualizer, and how they can import their organization-specific EA models including anonymized test data set into SocioCortex in order to make it analyzable by the SC Visualizer. Furthermore, we served as a technical support, i.e., we provided answers and assistance in case of technical problems, and fixed possible bugs of the prototype.

In the third SocioCortex Community workshop, we presented intermediary results of the working groups. This included demonstrations of the dashboards and visualizations created by the working group members as well as an elaboration and discussion on first impressions of the SC Visualizer.

After a further time period of three months, we conducted final interviews in order to conclude the case study. Thereby, we specifically asked for strengths and weaknesses of the SC Visualizer, particularly when applying in practice and real-world scenarios. Furthermore, the interviewees provided feedback on technical features they were missing and which they consider as essential for an ad-hoc EA analysis tool.

In the following sections, we elaborate on three selected cases, whereas each of those cases represent the application of the SC Visualizer as "Ad-hoc EA Analysis" tool in a different industry partner and member of the working group. Thereby, we not only present the highlights of the interview protocols, but also present data models and dashboards as defined by the respective industry partner as additional sources of evidence for the case study (Patton, 2002; Yin, 2014). It is noteworthy that the set of industry partners of this case study is distinct to the set of industry partners of the case study described in Section 6.2.

6.3.2. Case 1: Ad-hoc EA Analysis in a German Logistics Company

This case's industry partner is a German Company in the logistics sector with 10,001+ employees. The interviews were conducted with two IT specialists having leading roles in the company's EAM department.

Currently, they use a traditional EA tool (Matthes et al., 2008) to manage and analyze an EA model. For the prototypical application of the SC Visualizer as ad-hoc EA analysis tool, an excerpt of the traditional EA tool's model and data was exported to a Microsoft Excel file in a first step. Subsequently, the data was reduced and anonymized to a test data set. Finally, we imported the Excel sheet to SocioCortex and consolidated the data model which was automatically inferred by SocioCortex from the Excel sheet's internal structure.

The final data model of this case is depicted in Figure 6.8. For the sake of clarity and particularly for confidentiality reasons, we do not show details of the types, and slightly renamed classes and relations. However, the data model shown in Figure 6.8 is still semantically equivalent to the imported data model. Its central elements are business applications which support specific

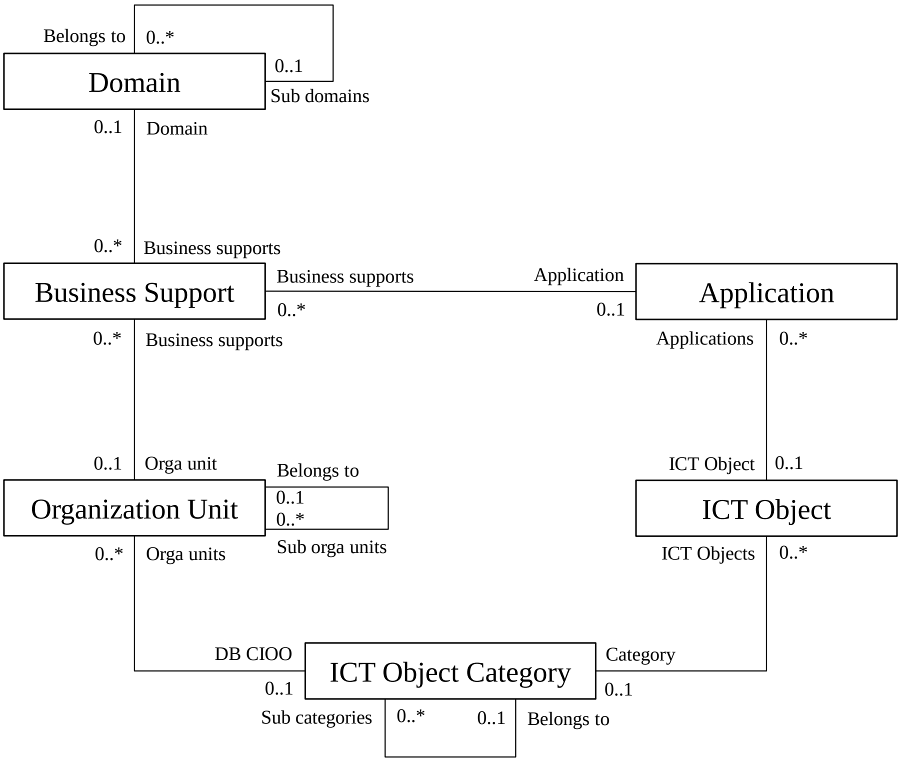


Figure 6.8.: The data model created in case 1. Attributes are hidden for the sake of clarity and for confidentiality reasons.

domains in specific organization units via the *Business Support* class. Furthermore, this excerpt of the EA model captures ICT objects which are assigned to respective categories. Business domains, organization units, and ICT object categories are organized hierarchically and thus have a reflective relation.

Based on the imported data model, enterprise architects of the industry partner created the exemplary dashboard in Figure 6.9 in an unassisted and iterative way. At the time of the final interview, the dashboard contained multiple visualizations to quantify and visualize different aspects of the EA model. They used two different visualization types, namely the "Simple Number" and "Bar Chart" in order to show the overall number of applications and domains on the one hand, and different quantified aspects for individual domains on the other hand.

In the concluding interview with this industry partner, the we asked specifically for strength and weaknesses of the SC Visualizer which were experienced during its application for ad-hoc EA analysis. Thereby, they named the following positive key aspects of the SC Visualizer client:

Automated analysis and transparency of the analysis model's dependencies The Social Information Flow Graph (SIFG) and its view can serve as foundation for an extensive meta-analysis of the analysis model, e.g., to identify stakeholders (cf. concerns in Section 6.2.3). According to the industry partner, this feature sets it apart from analysis capabilities of comparable tools in the field of EAM and even Self-Service BI.

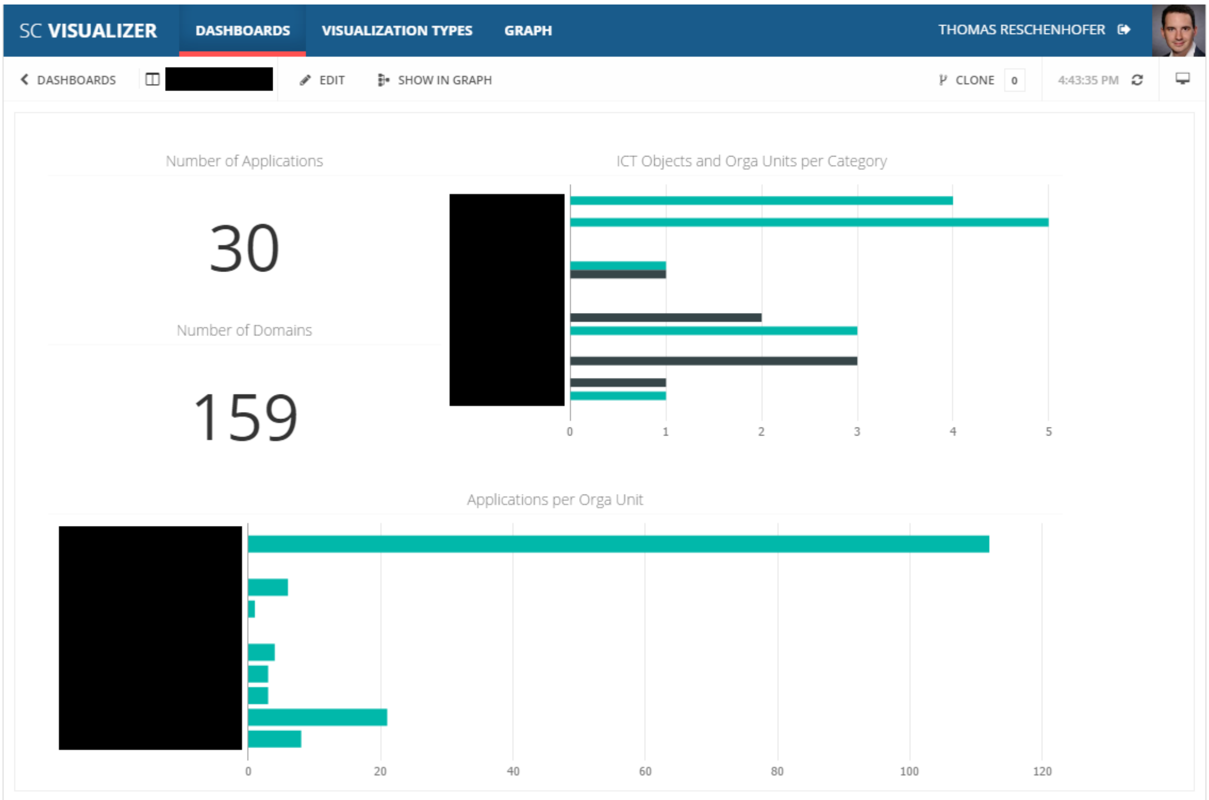


Figure 6.9.: An exemplary dashboard created by the industry partner in case 1.

Intuitive usability The industry partner highlighted that the familiarization process with the basic facilities of the SC Visualizer was fostered by its intuitive usability. This included specifically the management of dashboards and their visualizations as well as operating the SIFG view.

Grid system of the dashboard The industry partner specifically emphasized the usability of the grid system of the dashboard (cf. Section 5.3.2). It enables a "smart and dynamic arrangement of elements on the dashboard" by end-users.

Immediate feedback and clear error messages As a defining feature of EUA software (cf. Section 2.2.1), the industry partner appreciated the direct feedback and clear error messages when defining data bindings for visualizations. In this way, he was immediately able to validate the input and if the visualization is defined as intended.

Visual pleasing design One of the main differences of the SC Visualizer compared to other tools in the field of EAM was mentioned to be its visual pleasing design. Although the praise for the design of visualizations has to be forwarded to the integrated visualization libraries and their developers, the design decision to separate visualization types and bind them to the underlying data model through a well-defined data input interface (cf. Section 4.3.3) is essential for being able to integrate common visualization libraries.

While the interviewee outlined several strength of the SC Visualizer, particularly when comparing it to traditional EA tools, the interview also revealed two major weaknesses of the current prototype:

MxL is too complex for causal end-users Despite our endeavor to empower end-users to formulate of complex queries based on evolving complex linked data through implementing several UI features in our query editor interface (cf. Section 5.2.3), the query formulation UI is still considered as far too complex for casual end-users which are not tech-savvy. One of the interviewed enterprise architects mentioned that "after several weeks of not dealing with MxL, [he] was not able to remember the syntax and needed help from MxL tutorials and documentation". According to him, "MxL is not intuitive enough for the casual end-user". He proposes to implement an alternative UI for casual end-users enabling the definition of queries by "more WYSIWYG and Drag&Drop".

However, at the same time the interviewees agree that formulating queries with MxL is "much easier than with other [textual] query languages because of the auto-completion feature and augmented model view".

Configurability of visualization types only possible for experts On a similar note, the interviewees also consider the adaption of visualization types as too complex. Changing supposedly minor visual properties of visualizations already requires extensive web development skills.

In this regard, it has to be noted that our conceptual framework for collaborative EUA as described in Section 3.1 provides that the development and adaption of visualization types is indeed carried out by professional web developers. Nevertheless, the fact that this point was raised indicates that the tool should foster communication between the different involved user roles (in this case: analytics end-users and web developers). For example, this could be achieved by an integrated tool-support for creating and implementing visualization-related change requests.

Therefore, the interviewees designate the SC Visualizer as a tool for IT and domain experts rather than an EUA tool for casual end-users. In the context of EAM, they state that enterprise architects as tech-savvy end-users could act as experts and pre-configure dashboards and visualizations. However, this would not represent "real self-reporting for end-users", as added by one of the interviewed enterprise architects.

In order to move the SC Visualizer towards such a "real self-reporting tool", the interviewees suggest the implementation of additional features:

Visual query language Although the interviewed enterprise architects appreciate MxL's flexibility and expressiveness, they consider it as not suitable for casual end-users. Therefore, they would like to have an additional visual query editor which enables the definition of simple queries and provides more guidance for end-users. In this sense, the EUA tool should combine two data binding mechanisms: On the one hand, certain use cases still require the definition of complex queries to configure complex visualizations. On the other hand, the configuration of simple visualizations could be done by a more user-friendly UI, or (semi-) automated, e.g., through structural pattern matching (Schaub et al., 2012; Hauder et al., 2012; Roth et al., 2013).

End-user-configurable view filters In the current prototype of the SC Visualizer, visualization types provide two different kinds of parameters: Data bindings to define the data input interface of the visualization type, and visual settings to make visual properties of the visualization type configurable. In the context of the case study, the interviewees asked for an additional parametrization of visualization types to define visualization-specific view filters. In contrast to data bindings and visual settings which are only definable when configuring a visualization, a view filter would be displayed beside the actual visualization and rendered as an input field. In this way, end-users could simply pass filter parameters to the visualization without having to adapt the actual visualization. For example, defining a respective view filter in the "Directed graph" visualization type as described in Section 4.3.2 would empower end-users to filter the displayed set of nodes or edges.

6.3.3. Case 2: Ad-hoc EA Analysis in a German Automotive Supplier

The industry partner in this case is a German automotive component supplier (10,001+ employees). The meetings and interviews were conducted with an enterprise architect of the internal EAM department.

Currently, they use a traditional EA tool (Matthes et al., 2008) to manage an EA model. Furthermore, at the time of the case study, they are doing research about the integration of prevalent Self-Service BI tools with their current EA tool. Therefore, they are actively searching for solutions enabling end-user-oriented ad-hoc EA analysis.

Analogous to the first case, the industry partner created an export of the traditional EA tool's model and data in the form of a Microsoft Excel file. The exported data was reduced and anonymized in order to generate a test data set. In the same fashion as in the first case, we imported the test data through SocioCortex's integrated import mechanism for Microsoft Excel sheets.

The imported data model for the second case is illustrated in Figure 6.8. Again, we maintain confidentiality by renaming classes and relations, and omitting attributes. Nevertheless, the data model still has the structural properties of the data model as provided by the industry partner.

The EA model in this case captures business applications as well as hierarchically organized functional domains they are used in. Furthermore, they are assigned to an ICT Object by which they are governed, and to organizational units, e.g., acting as owners. Organizational units are defined hierarchically too. In this sense, the data model is similar to the one in case 1. However, it also defines additional concepts, e.g., components, business objects, and devices.

Based on the imported data model and corresponding test data set, the enterprise architect and one of his colleagues evaluated the SC Visualizer and the underlying SocioCortex platform by adapting the data and its model, and by creating a simple dashboard. Figure 6.11 displays the resulting dashboard containing a visualization of type bar chart, which shows the number of applications per organizational unit. The simplicity of the dashboard is owed to the perceived cumbersomeness of familiarizing with MxL and thus difficulties in defining respective data bindings.

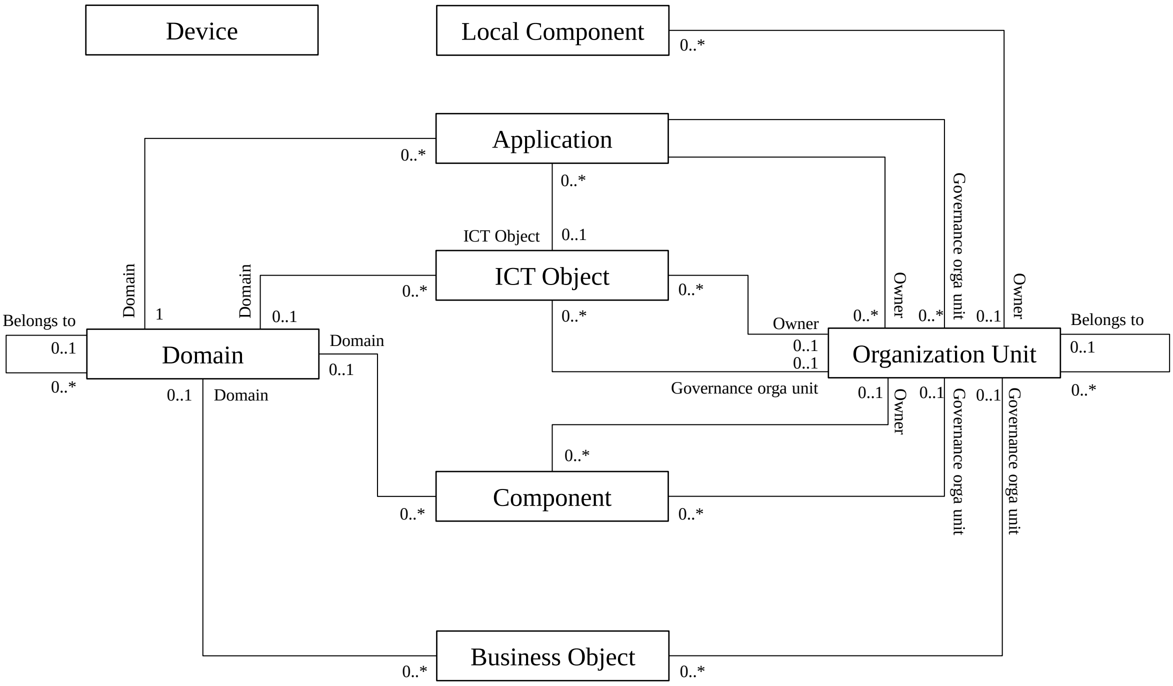


Figure 6.10.: The data model created in case 2. Attributes are hidden for the sake of clarity and for confidentiality reasons.

To conclude the case study with this case's industry partner, we interviewed the enterprise architect and asked him about his experiences when using the SC Visualizer and the underlying SocioCortex platform. He named and described the following positive key features:

Flexibility regarding data modeling As a first point, the interviewee raised the flexibility of the data modeling approach. The flexible data modeling is empowered by the Hybrid Wiki concepts, and was particularly experienced during the data consolidation phase.

Flexibility of dashboards and visualizations The enterprise architect also emphasized the flexibility when designing dashboards and configuring their visualizations. This regards specifically the grid system of the dashboard and thus the flexibility in specifying the location and size of multiple visualizations. The statement also includes flexibility in binding the underlying data to visualizations through complex queries. The interviewee explicitly states that the prototype's flexibility is "a big difference compared to [the EA tools] of other vendors".

Extendability of visualization type catalog Another feature which—according to the interviewee– sets the SC Visualizer apart from comparable tools is the extendable visualization type catalog. Thereby, web developers can implement new visualization types and add them to the SC Visualizer at run-time. The interviewed enterprise architect notes that this enables a "marketplace for visualizations", whereas "one could think of the development of visualization types through external service providers".

Device-independence As a consequence of building the prototype of the SC Visualizer as web application, it is a platform-independent application. Furthermore, it does not even re-

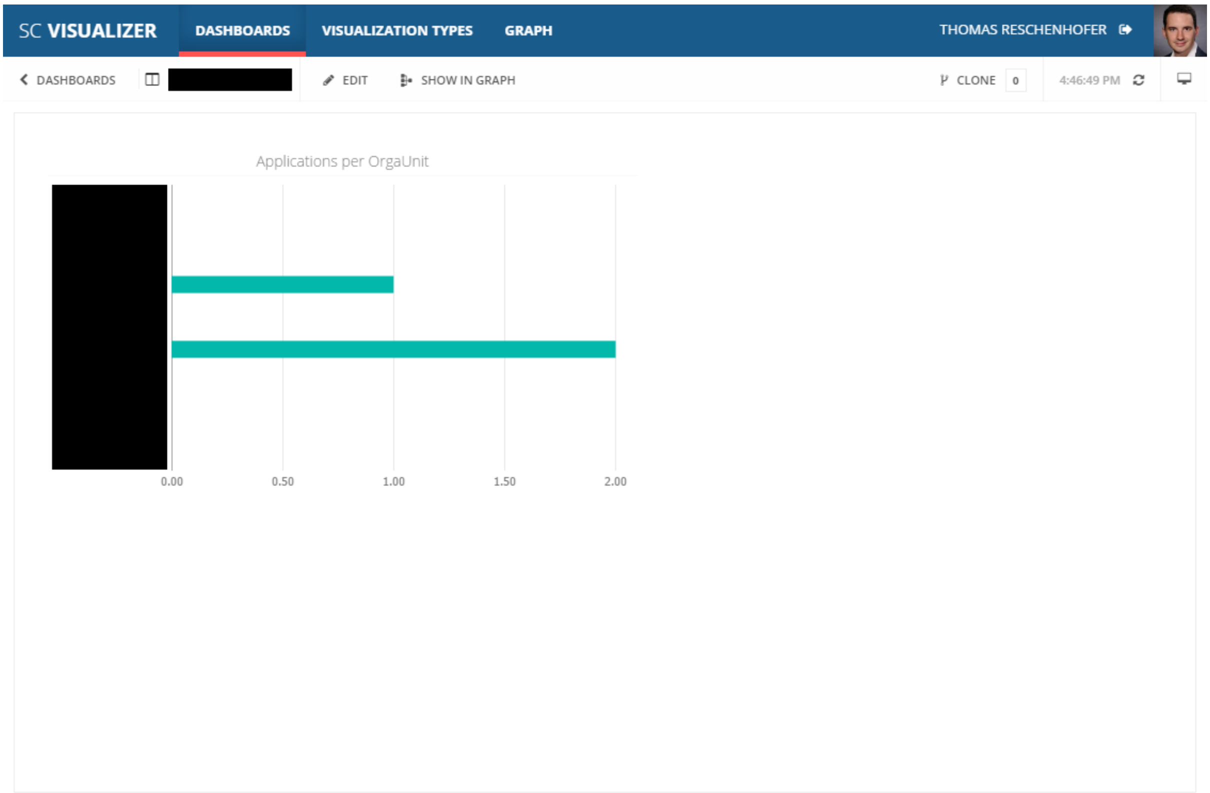


Figure 6.11.: An exemplary dashboard created by the enterprise architect in case 2.

quire end-users to install a Desktop application. This device-independence was explicitly mentioned as a technical strength of the SC Visualizer.

Complementary to the strength of the SC Visualizer's prototype, we also discussed its weaknesses with the enterprise architect of this case's industry partner. Thereby, he came up with the following two key issues:

Configuring visualizations requires technical expertise The main conclusion of the second case of the study is the same as in the first one: Binding data to visualizations is considered as "not suited for casual end-users". It requires specific technical expertise and knowledge in defining functional queries with MxL.

Therefore, defining data bindings with MxL is again mentioned as a strength and weakness of the prototype at the same time. In this sense, the flexibility and expressiveness of the functional and textual query language MxL competes with the user-friendliness and user-guidance provided by comparable visual languages (Störrle, 2011; Haag et al., 2015; Soylu et al., 2015).

Applying the SIFG requires commitment On a related note, the interviewee expressed his reservation about the SIFG. Although he agrees that it can serve as foundation for different meta-analysis use-cases, it "requires commitment from co-workers to maintain the meta data" of all elements within the SIFG. For this, "they have to expect direct benefits in exchange for the additional effort [when manually maintaining meta-data]".

As a conclusion, the interviewee stated the fact that he would certainly consider in applying the SC Visualizer as an "expert tool for a central EAM team". However, it still lacks UI facilities to make it "configurable for casual end-users". In this sense, he comes to the same conclusion as the interviewees of the first case which further underpins the SC Visualizer's nature as expert tool rather than an end-user tool.

Analogous to the first case, the interviewee provided his suggestions for improvements and features he would like to have in order to use the SC Visualizer as a tool for end-users:

Visual query language Basically repeating the suggestion of the interviewees of the first case, the enterprise architect of the second case asks for an alternative query formulation UI for end-users. He outlines that a "query wizard" should enable casual end-users to configure simple visualizations, while an "expert interface" would enable expert users to configure more complex visualizations.

Sharing of visualizations and dashboards As a fundamental feature of a collaborative EUA tool, the enterprise architect names a "complex sharing mechanism for dashboards and visualizations". As a prerequisite, the SC Visualizer has to provide facilities to define access rights on the level of visualizations in order to define which users are allowed to view and adapt which visualizations.

As described in Section 5.3.2, dashboards and visualizations are mapped to SocioCortex entities, which in turn are already subject to SocioCortex's access control mechanism. Therefore, the platform already provides the technical capabilities to allow the definition of access rights through the SC Visualizer. However, the current prototype of the SC Visualizer does not yet implement a respective UI for managing access rights. Then again, implementing this feature only requires an adaption of the SC Visualizer front-end application.

6.3.4. Case 3: Ad-hoc EA Analysis in a German Bank

In the third case, we cooperated with the enterprise architect of a German Bank (5,000 - 10,000 employees).

For managing the holistic EA model of the organization, the EA department of the industry partner uses traditional EA tools. However, in the past the industry partner already took part in evaluations of tools implementing an agile approach to EAM. For example, the industry partner was already part of the *Wiki4EAM* (Matthes and Neubert, 2011) and thus already assessed the Hybrid Wikis system as a tool for EA modeling.

In contrast to the other cases described in the previous sections, the industry partner did not provide a test data set as Microsoft Excel export. They manually created an exemplary data model for the evaluation of the SC Visualizer. Figure 6.12 illustrates the data model created in this case. Analogous to the other cases described in this thesis, we renamed classes and relations and omitted attributes for the sake of confidentiality.

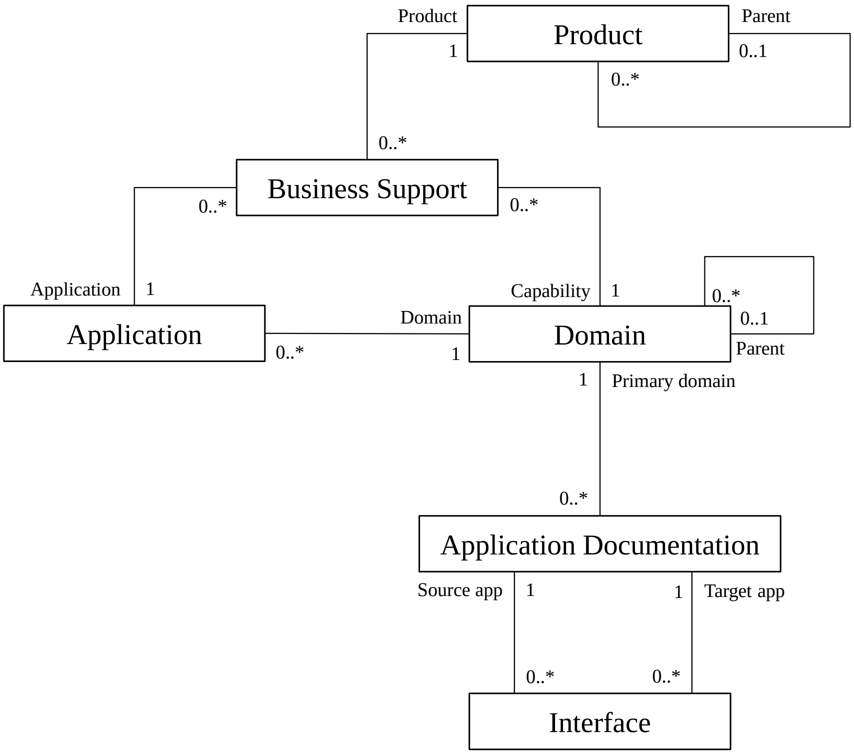
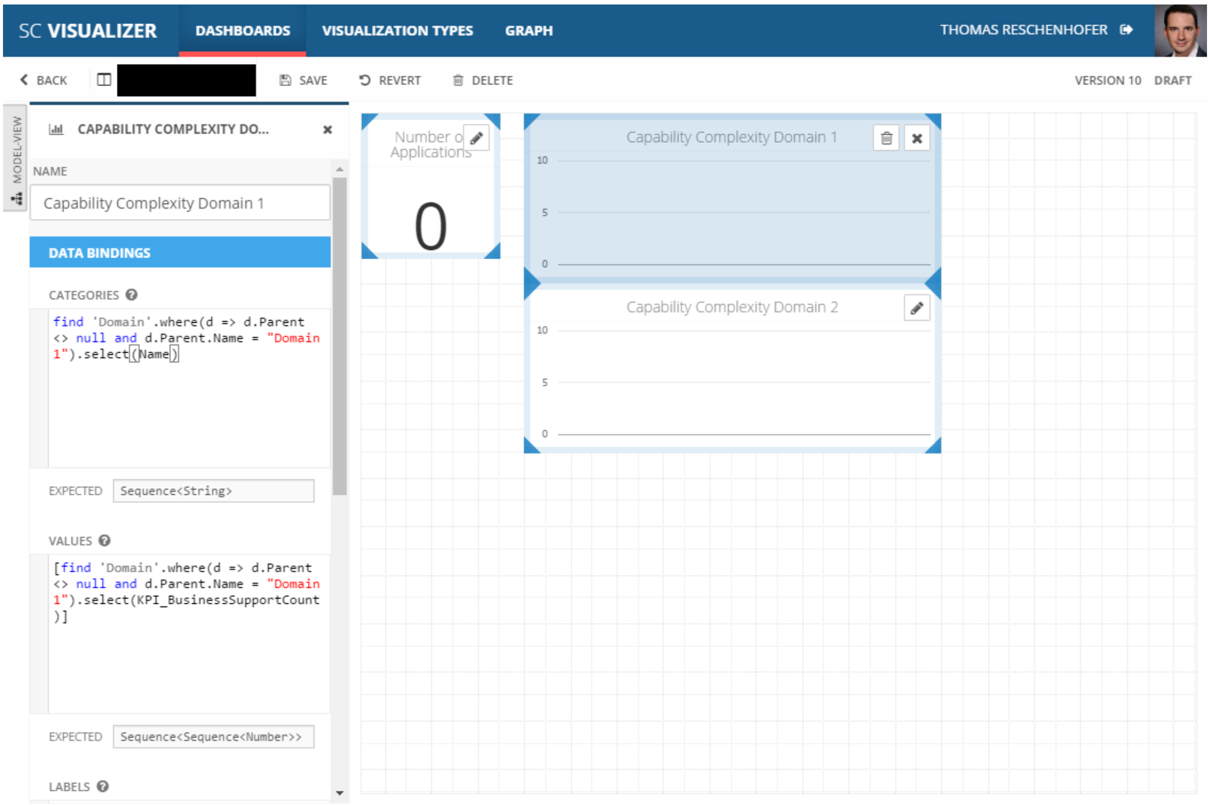


Figure 6.12.: The data model created in case 3. Attributes are hidden for the sake of clarity and for confidentiality reasons.

The designed data model essentially captures the organization's application landscape, i.e., it documents applications and which products they support in which functional domain. Furthermore, the EA model includes interfaces between applications which compare to information flows as described in our exemplary data model in Figure 4.7. In addition to data model elements, the enterprise architect also created analytical abstractions in the form of derived attributes. For example, he defined a derived attribute for functional domains calculating the number of business supports which are assigned to it via the capability relation.

Figure 6.13 shows the dashboard as result of the enterprise architect's endeavor in defining visualizations based on the underlying data model. Since he created the data model manually and did not enter any test data, the rendered views of the visualizations are rather irrelevant. However, switching the dashboard to the edit mode reveals that the enterprise architect was able to define rather complex data bindings. As shown in Figure 6.13, he configured a visualization of type "Bar chart" to show the number the value of a self-defined KPI for each functional domain which have the same parent domain. The corresponding SIFG view as displayed in Figure 6.14 shows that the enterprise architect was capable of defining a remarkably complex analysis model consisting of multiple visualizations and chains of analytical abstractions.

Again, the enterprise architect provided his impressions of the SC Visualizer as feedback, e.g., during individual meetings and the third SocioCortex workshop as illustrated in Figure 6.7.



NAME
Capability Complexity Domain 1

CATEGORIES

```
find 'Domain'.where(d => d.Parent
<> null and d.Parent.Name = "Domain
1").select(Name)
```

EXPECTED Sequence<String>

VALUES

```
[find 'Domain'.where(d => d.Parent
<> null and d.Parent.Name = "Domain
1").select(KPI_BusinessSupportCount
)]
```

EXPECTED Sequence<Sequence<Number>>

LABELS

Figure 6.13.: An exemplary dashboard created in case 3.

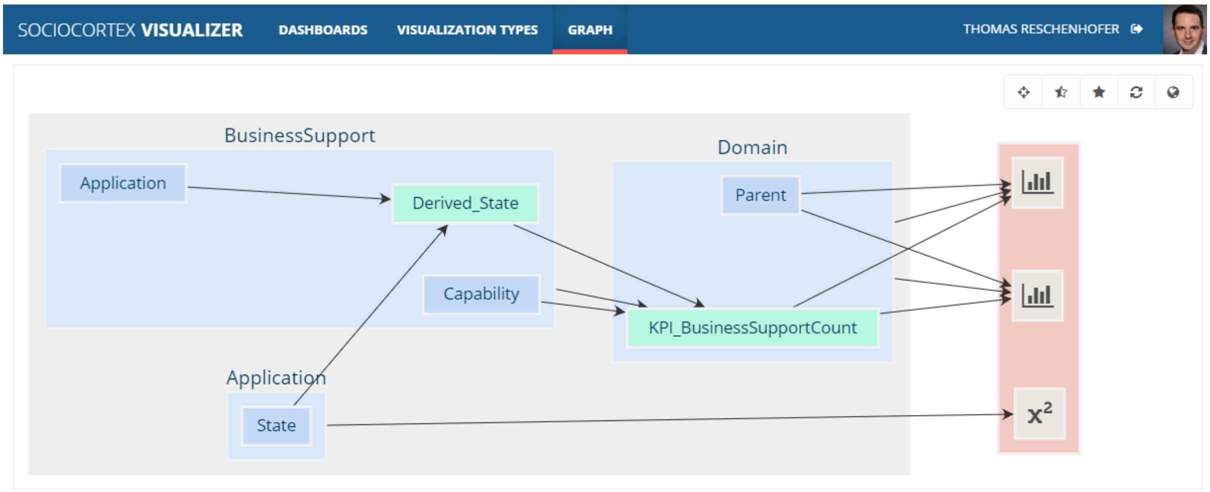


Figure 6.14.: An excerpt of the SIFG illustrating the semantic dependencies for the dashboard in Figure 6.13.

The feedback includes both concrete strengths of the current prototype as well as suggestions for improvements. The strengths were named as follows:

Flexibility of the data model A strength of the prototype which he already experienced during his evaluation of SocioCortex's predecessor (Matthes and Neubert, 2011; Neubert, 2012) is the flexibility of the data modeling approach. This feature was again mentioned as a "big difference to traditional EA tools".

Intuitive usability of SC Visualizer Another feature which sets the SC Visualizer apart from comparable tools is its intuitive usability. This specifically refers to the definition of dashboards and visualizations through the implemented grid system. The precise arrangement of visualizations seems to be an important feature of dashboards, particularly when considering that the same point was raised in other cases.

Expressiveness of MxL The enterprise architect stated multiple times that he considers MxL as "a great tool to define complex EA metrics". In contrast to the feedback gained from the other cases described in previous sections, he would prefer to define MxL queries through a textual query editor instead of a visual query interface, particularly since the MxL query editor provides "useful features like auto-completion and the augmented model view". He considers MxL as the main feature of the prototype and its defining feature which sets it apart from other tools. When asked about the complexity of textual MxL queries he responded that "even in [Microsoft] Excel you'll find complex formulas".

In addition to strengths of the prototype, the enterprise architect also expressed a particular suggestion for improvement, namely to "make visual properties of visualizations easier configurable". As a concrete example, he named the scaling of dimensions in bar charts. For example, he intended to position two bar charts side by side. However, since both charts are rendered independently, the scales of their y-axes did not match to each other. On the one hand, this could indicate that the web developers defining the visualization types should make the scaling of axes and other aspects configurable. On the other hand, this could also be interpreted as a lack of interconnectivity between visualizations, i.e., by connecting different visualizations they could synchronize certain parameters, e.g., data or visual parameters.

6.4. Synthesis of Evaluation Results

In this section, we briefly summarize the findings of the three evaluation strategies described in Sections 6.1, 6.2, and 6.3 and derive a common set of lessons learned.

In general, implementing the prototype as a web application based on modern JavaScript technologies and frameworks enabled us to build a device-independent software solution and to benefits from the huge JavaScript ecosystem, e.g., modern and visual appealing visualization libraries. This was unanimously received by evaluation partners as a positive technical decision. Furthermore, it also facilitated the operation of the evaluation since evaluation partners did not have the need to install any software, but were able to simply access a web application which was hosted at the sebis chair.

On another note, the combination of well-defined data modes, analytical abstractions, and views paired with a flexible, collaborative, and iterative modeling approach was perceived as a defining feature which sets it apart from other EUA and particularly EA tools. While prevalent EUA tools (e.g., spreadsheets) lack an explicit design (cf. Section 2.2.4), traditional BI and EA tools lack flexibility with respect to adaptability of data models, analytical abstractions, and view. Our prototype positions itself in between, i.e., data models, analytical abstractions, and views are explicitly defined, but at the same time adaptable at run-time through end-users.

Further positive remarks address the UI features implemented in our prototype. For example, the auto-completion feature of the MxL code editor as well as the augmented model view were received well by participants of both the online experiment and case studies as described in Section 6.1 and 6.3 respectively. Furthermore, the evaluation revealed that enabling end-users to arrange and size visualizations freely on the grid of dashboards seems to be a helpful feature which surpasses the visual modeling capabilities of comparable Self-Service BI systems (Walter, 2015). In this context the case study partners also confirmed that the prototype provides immediate visual feedback when configuring visualizations and thus implemented one of the central features of EUA tools (cf. sec:eua:spreadsheetFeatures).

The primary lesson learned of the evaluation is that the prototype still lacks UI features to consider it as a tool for analytics end-users. The unanimous tenor is that the tool is still "too technical, but the general approach is good". More specifically, the evaluation partners agree that it definitely provides means to enable IT and domain experts to define even complex data models, analytical abstractions, and views. However, it is not applicable for casual end-users. This is mainly due to the technical and functional nature of MxL and its query formulation UI. The evaluation partners' unanimous opinion is that the prototype should provide a visual query editor which empowers casual end-users to configure at least simple visualizations, while the textual query editor as presented in this thesis would act as an expert interface which would still allow the definition of complex queries and the configuration of complex visualizations. In this sense, it should be considered to combine our approach focusing complex queries with the end-user-friendly structural model matching approach as defined by Roth et al. (2013).

The next issue relates to the SIFG view: As shown in Section 6.2, the SIFG view of the SC Visualizer provides transparency of analysis artifacts and makes them explorable by end-users, which in turn hypothetically enables a couple of interesting use-cases. When asked about its application in practice, the evaluation partners raised a major concern: The SIFG might be subject to privacy and surveillance issues. In its current implementation, users can explore which co-workers are editing or owning certain analysis artifacts. However, showing also the users of analysis artifacts would be only a minor technical challenge. While this would enable a whole lot more of use-cases, it also raises the question of what users are allowed to know from co-workers, or what they should know from them, i.e., which level of transparency or surveillance is appropriate? In the end, the answer to this question will depend on multiple factors, e.g., the actual use-case, or the organization context. Therefore, we cannot and don't want to give a general answer. For a general discussion about privacy and surveillance in the context of BI and SNA, we refer to related literature (Borgatti and Molina, 2003; Gürses and Diaz, 2013; Lyon, 2003; Varian, 2014; Zuboff, 2015).

CHAPTER 7

Conclusion

In this chapter, we summarize this thesis and conclude on its contribution. Thereby, we specifically recapitulate the research questions raised in Chapter 1 as well as the requirements defined in Chapter 3. In Section 7.2, we critically reflect on the findings and methodology as described in this thesis, and outline known limitations. Finally, in Section 1.5 we propose potential future research topics which are enabled by this thesis' contribution.

7.1. Summary

The thesis at hand starts with a motivation of how an increasingly turbulent business environment can be tackled by meta-model-based information systems which empower end-users to dynamically define and adapt data models. However, the problem description in Section 1.1 highlights that the flexible data modeling approach imposes various challenges to the analysis of complex data structures in collaborative environments. This particularly includes keeping analysis artifacts consistent in light of data model changes, supporting complex linked data structures according to the definition in Chapter 1, and enabling multiple users to contribute to the analysis approach in order to harness their collective intelligence. Based on the problem description, Section 1.2 elaborates on the formulation of concrete research questions, while Section 1.3 discusses the adoption of the design science research methodology to the context of this thesis. Sections 1.4 and 1.5 summarize the thesis' core contributions and structure respectively.

Chapter 2 introduce Adaptive Information System (AIS) and elaborates on the Hybrid Wiki approach as the foundation for the conceptual design of our solution (cf. Section 2.1). Section 2.2 summarizes characterizing features of EUA software, e.g., analytical functions as well as immediate visual feedback. It subsumes with the results of our state-of-the-art analysis of how spreadsheets as de facto standard EUA tool are applied in practice, and from which shortcomings they suffer. In Section 2.3, we provide a brief overview over related work in the field of spreadsheets, model-based analytics and visualizations, and Self-Service and collaborative BI.

In Chapter 3, we describe how we derived the requirements for the conceptual and technical design of our approach to collaborative EUA on evolving complex linked data. To this end, in Section 3.1 we first define a respective conceptual framework which captures three different conceptual layers, namely the data model layer, the analytical abstraction layer, and the view layer. We further identify different roles of users interacting with the artifacts of the layers. Based on our conceptual framework as well as on related work on collaborative technology, we describe the systematic derivation of 22 requirements categorized along the layers of the conceptual framework and including a cross-cutting category for meta-analysis-related requirements.

Based on the derived requirements, we discuss the conceptual design of our approach in Chapter 4. Again, we separate design decisions along the layers of our conceptual framework. Consequently, Section 4.1 starts with a description of how we extended the original Hybrid Wiki meta-model in order to be able to support complex linked data. Subsequently, we elaborate on the design of concepts for analytical abstractions including the design of the query language MxL (cf. Section 4.2). In Section 4.3, we explain the meta-model for the view layer including a concept to specify a data input interface through which end-users can enter data into instantiated view templates. For the concepts of all layers, we discuss the design decisions and assess their consequences. Section 4.4 addresses the meta-analysis of the inter-connected data model elements, analytical abstractions, and view elements and discusses the conceptual design of the Social Information Flow Graph (SIFG) as means to ensure transparency of the analysis model and to make it explorable by end-users.

Chapter 5 outlines the highlights of the implementation of the prototype empowering end-users to collaboratively analyze evolving complex linked data. As described in Section 5.1, the prototype is based on the SocioCortex platform which implements the data model and analytical abstraction concepts of the meta-model derived in Chapter 4. Furthermore, we elaborate on an AngularJS-based web framework which serves as a wrapper for the SocioCortex back-end's REST API, and describe further reusable UI components in Section 5.2. Specifically, we present two UI components for enabling end-users to define complex queries, namely the MxL code editor and the augmented model view. In the subsequent Section (5.3), we briefly describe three web clients as part of the SocioCortex front-end, whereas the SC Visualizer as the central EUA application occupies the most important role in the context of this thesis. It also implements dashboards, visualizations, and visualization types which correlate to the view concepts as defined in Chapter 4. Furthermore, in this section we also elaborate on conceptual and UI design decisions regarding the SIFG and its prototypical implementation and integration into the SC Visualizer.

The prototypical implementation serves as a proof of concept on the one hand, and enables the evaluation of the developed concepts as described in Chapter 6 on the other hand. Thereby, we applied different evaluation strategies to validate different aspects of the prototype. In Section 6.1, we elaborate on the setting and key findings of an online experiment for the evaluation of the usability of the query formulation interface. Section 6.2 outlines the setting and key findings of the evaluation of the Social Information Flow Graph by interviews with three different industry partners. And Section 6.3 summarizes how we prepared and operated a case study including three different cases in three companies which enabled us to apply the prototype in practice for a specific EUA use-case, namely ad-hoc EA analysis.

After summarizing the thesis and its chapters, we assess its results with respect to the research questions raised in Section 1.2. Therefore, we provide a brief answer and refer to specific sections of the thesis which provide more details.

> ***RQ 1:*** What is the state-of-the-art of tool support in EUA, and what are shortcomings of those tools—particularly spreadsheets—in the context of knowledge-intensive team work?

In order to answer the first research questions, we performed a state-of-the-art analysis of how spreadsheets as the de facto EUA tool are currently applied in practice. First, we observed which kind of data is typically maintained in spreadsheets and derived a respective semantic meta-model capturing typical structural patterns occurring in spreadsheets (cf. Section 2.2.2). Second, we also conducted case studies in order to investigate HOW spreadsheets are applied in practice. To this end, we studied different usage scenarios of spreadsheets in practice and derived respective usage patterns which are described in Section 2.2.3. Third, to identify shortcomings of today's EUA tools, we conducted an empirical observation in two German companies (cf. Section 2.2.4).

In addition to our own empirical studies, we also performed a literature review in the fields of spreadsheets, EUA in general, Self-Service and collaborative BI, and model-based analytics and visualizations. On the one hand, the literature review revealed the characterizing features and success factors of EUA tools. On the other hand, it also provides an overview over related approaches as well as a differentiation to this thesis' approach.

> ***RQ 2:*** What are requirements for empowering end-users to collaboratively analyze evolving complex linked data structures?

With respect to RQ 2, Chapter 3 not only lists identified requirements, but also describes a conceptual framework which served as foundation for the systematic derivation of them (cf. Section 3.1). Thereby, the conceptual framework is based on related work on end-user-driven data visualizations (Chi and Riedl, 1998), visual information analysis (Isenberg et al., 2008), end-user-driven model analysis (Roth et al., 2013), and general EUA (Tamm et al., 2013). It defines three logical layers building on the top of each other, namely a data model, analytical abstraction, and view layer. By adopting generic requirements for collaborative technology as defined by de Hertogh et al. (2011) to the artifacts of those layers as well as by addressing specific shortcomings identified in our state-of-the-art analysis of EUA software, we were able to systematically derive 22 requirements for a tool empowering end-users to collaboratively analyze evolving complex linked data. The requirements are categorized and summarized in Section 3.2.

> ***RQ 3:*** How can a system design for a tool empowering end-users to define data models, analytical abstractions, and visualizations look like?

The answer to this research questions is primarily described in Chapter 4: The conceptual system design is formally defined by a meta-model which captures the artifacts which were already defined in our conceptual framework in Section 3.1. The data model concepts are primarily derived from the Hybrid Wiki meta-model, and extended by additional concepts in order to support complex linked data structures. Furthermore, in Sections 4.2.1 and 4.3.1 we describe concepts of the analytical abstraction layer (e.g., custom functions) and the view layer (e.g., visualizations). The holistic system design for a tool empowering end-users to define data models, analytical abstractions, and visualizations is illustrated by the meta-model in Figure 4.1.

The technical system design of our prototype is outlined in Chapter 5. Figure 5.1 illustrates how the prototype is separated into different components and assembled to a holistic EUA tool. It consists of three different web-based clients for managing the content, modeling data structures and analytical abstractions, and defining visualizations.

> ***RQ 4:*** What are features and properties a language for defining queries and calculations on evolving complex linked data structures must have?

In Section 4.2.2, we elaborate on the key features and properties of the Model-based Expression Language (MxL) which serves as a functional, object-oriented, sequence-oriented, and statically type-safe query language based on the Hybrid Wiki data model. Thereby, we provide a brief justification for each of those properties, and specifically describe MxL's type system which complies to the data model concepts of the underlying Hybrid Wiki meta-model. Furthermore, Section 4.2.2.2 summarizes basic MxL functions which implement different kinds of operations, e.g., common query operations (projection, selection, ...), or aggregation operations.

Section 5.1.2 highlights specific aspects of MxL's implementation, and illustrates the interpretation process of MxL expressions including a scanning, parsing, and type checking phase. Moreover, this section also elaborates on how the MxL properties enable the prototype to be adaptive, i.e., to automatically adapt MxL queries on changes of the underlying data model in order to keep them consistent.

> ***RQ 5:*** How can end-users be supported in defining complex queries, calculations, and visualizations on evolving complex linked data structures?

This question is primarily answered in Section 5.2.3 which describes the implementation of the MxL query formulation UI. Its central component is a textual query editor for MxL providing a number of features which were confirmed to be helpful during its evaluation (cf. Section 6.1). For example, it provides MxL-specific syntax highlighting, error localization, and auto-completion support. Furthermore, an augmented model view illustrating the underlying data model as UML class diagram supports users in navigating through potentially complex linked data structures.

With respect to the definition of complex visualizations, Section 5.3.2 outlines the highlights of the implementation of the SC Visualizer in general, and the UI for defining dashboards and configuring visualizations in particular. Thereby, we adopted characterizing features of EUA as identified in Section 2.2.1s. For example, when configuring visualizations, immediate visual feedback enables end-users to check if the provided input leads to the intended visualization without having to commit its configuration.

> ***RQ 6:*** How can end-users be supported in performing a meta-analysis on collaboratively defined data models, analytical abstractions, and visualizations?

Our answer to this research question is the Social Information Flow Graph (SIFG). The SIFG is a means to ensure transparency of which analysis artifacts are connected in which way, and awareness of which users are interacting with those artifacts. In this context, analysis artifacts refer to data model elements, analytical abstractions, and view elements. The SIFG empowers end-users to interactively explore the analysis model which is formed by those artifacts and the semantic dependencies between them. The conceptual model of the SIFG is outlined in Section 4.4.

Based on its conceptual design, we present the SIFG's visual design in Section 5.3.2. We specifically elaborate on UI features which facilitate the SIFG's usability and explorability by end-users, e.g, highlighting paths through a particular node of the graph, or collapsing composite nodes in order to reduce the visual complexity of the graph view.

> ***RQ 7:*** What is the experience of users of the proposed solution? What are further challenges of Web 2.0-based EUA?

Chapter 6 summarizes the settings and key findings of three different evaluation strategies, namely an online experiment to validate the usability of the query formulation UI (cf. Section 6.1), an interview series with practitioners of different domains to get feedback on the SIFG (cf. Section 6.2), and a case study including three cases of ad-hoc EA analysis to assess the prototype's practicability. Section 6.4 provides a synthesis of the key findings identified with the different evaluation strategies.

For further challenges of collaborative EUA we refer to Section 7.3 in which we summarize potential future research opportunities based on the findings of the thesis at hand.

We conclude our summary with a recapitulation of the requirements identified in Section 3.2. Table 7.1 lists the requirements and provides both a brief description of how we addressed them and a reference to the respective section of this thesis.

7.2. Critical Reflection and Known Limitations

While the thesis addresses each requirement listed in Table 7.1, we are aware that not all of them are fully met. Furthermore, the evaluation revealed further limitations of the prototype which were already discussed in Section 6.4. In the following sections, we recapitulate on known limitations and critically reflect both this thesis' contribution and its evaluation.

7.2.1. Functional Limitations of the Prototype

Section 6.4 already discussed limitations revealed by the different evaluation strategies. However, there are additional noteworthy limitations of the conceptual design and prototypical implementation of the approach as presented in this thesis. In the following, we systematically discuss various technical limitations along the layers of our conceptual EUA framework (cf. Section3.1)

Req	**Brief Description**	**Sections**
Data Model Requirements		
Req 1	Extended Hybrid Wiki data model with complex attribute types	4.1.1
Req 2	Hybrid Wiki data modeling approach, SC data table	2.1.2, 5.3.1, 5.2.2
Req 3	Hybrid Wiki data modeling approach, SC data table	2.1.2, 5.3.1, 5.2.2
Req 4	Workspace overview and augmented model view	5.2.3, 5.3.1
Req 5	Hybrid Wiki default feature	2.1.2
Analytical Abstraction Requirements		
Req 6	MxL and its query formulation UI	4.2.2, 5.2.3
Req 7	Extendable set of basic functions	4.2.2.2 , 5.1.2
Req 8	Commentability and shareability of analytical abstractions	4.2.1, 5.3.1
Req 9	Creatability and adaptability of analytical abstractions at run-time	4.2.1, 5.3.1
Req 10	Automated refactoring of analytical abstractions on changes	5.1.2
Req 11	Overviews of custom functions and derived attributes, auto-completion hints in MxL code editor	5.3.1, 5.2.3
Req 12	Derived from Hybrid Wiki concepts	4.2
View and View Template Requirements		
Req 13	Online implementation, separation into logic, template, and style	5.3.2.3
Req 14	Online code-editor enables adaption at run-time	5.3.2.3
Req 15	Forking mechanism for visualization types	5.3.2.3
Req 16	Visualization types are mapped to access-controlled entities	5.3.2.1
Req 17	Dashboards and visualizations are editable at run-time	4.3.1,5.3.2
Req 18	Reusability of visualizations in different dashboards	4.3.1, 5.3.2.1
Req 19	Dashboards, visualizations are mapped to access-controlled entities	5.3.2.1
Meta-Analysis Requirements		
Req 20	Extendable set of meta-attributes for information assets	4.4
Req 21	Static type-safety of MxL, MxL dependency management	4.2.2, 5.1.2
Req 22	Social Information Flow Graph	4.4, 5.3.2.4

Table 7.1.: Requirements identified in Section 3.2 and how we addressed them.

Limitation on the data layer primarily refer to original Hybrid Wiki concepts and its prototype. For example, Hybrid Wikis support a data-first approach to data modeling, i.e., users can add initially undefined structure to information objects, based on which a modeler expert can define a respective data model. Consequently, the system potentially allows states in which information objects are inconsistent with respect to the explicitly define data model. It depends on the actual use-case, which level of strictness is appropriate, i.e., if users should be enforced to stick to the defined data model in order to keep consistency, or if they should be allowed to deviate from the data model for the sake of flexibility. A more detailed discussion can be found in preliminary work (Reschenhofer et al., 2016a) as well as in the PhD thesis of Neubert (2012).

A technical limitation which is related to the extension of the Hybrid Wiki meta-model is its support for complex linked data as described in Section 4.1.1. According to the definition in Chapter 1, complex linked data is defined as a set of entities which potentially have arbitrarily nested attributes, attributes with multiple values, and multiple relations to other data entities. While the original Hybrid Wiki meta-model (Matthes et al., 2011) already supports multi-valued attributes and relations, we extended the set of possible attribute types by a *ComplexConstraint* in order to address nested attributes. However, the *CompositeType* as defined in 4.2 does neither support nested relations nor relations to nested objects. This naturally impairs the expressiveness of the meta-model with respect to its support for complex linked data.

The potential inconsistencies of the data with respect to its model also has consequences for the analytical abstractions: MxL expressions are based on the explicitly defined data model and expect that the data complies to it. If this is not the case, the evaluation will potentially fail, except the inconsistent value can be parsed successfully into the expected type. The same is true for empty or non-existent attributes which are represented by *null* values. Furthermore, MxL expressions cannot access free attributes of entities since they are not yet captured by the data model. Therefore, although the Hybrid Wiki meta-model provides great flexibility with respect to the data modeling approach, MxL does not benefit from them since it is based on the explicitly defined data model. However, it is noteworthy that MxL supports the evolution of the data model and ensures consistency of its expressions (cf. Section 5.1.2).

On a different note, the current prototype only implements basic features for the collaborative creation of analytical abstractions. For example, users can annotate MxL expressions and parts of them with comments and thus provide informal descriptions for oneself and for others. Furthermore, analytical abstractions can be separated and shared by defining reusable custom functions. However, the prototype as of today does neither provide an integrated synchronous messaging facility (*chat*) to foster communication between the system's users, nor does it support real-time collaboration enabling multiple users to define one MxL expression collaboratively. Therefore, the prototype still lacks sophisticated collaboration facilities and thus only partially fulfills Requirement Req 8.

Furthermore, in Section 4.2.1 we describe that analytical abstraction concepts are not defined as *AccessControlled* artifacts. Instead, the infer their access rights from the workspace they are assigned to. This behavior is inspired by data model elements (e.g., entity types), which also only infer the access rights from its workspaces instead of allowing users to define explicit ones. However, we think that there are use-cases which require a more fine-grained authentication concept where users should be enabled to define specific access control rules for analytical abstractions. Since the current prototype does not implement such a differentiating authentication concept, it suffers from this kind of technical limitation, and thus only partially fulfills Requirement Req 12.

The evaluation described in Chapter 6 revealed multiple limitations related to the view layer of our conceptual framework and related to the SC Visualizer front-end. The most obvious one which was already discussed in Section 6.4 is the lack of a query formulation UI for casual end-users. The current prototype only provides a textual query editor which targets technical experts and addresses the formulation of complex queries. However, missing a visual query editor for end-user is a major limitation of the prototype as a EUA tool.

Furthermore, although dashboards, visualizations, and visualization types are technically mapped to SocioCortex entities and thus inherently implement access control lists, the current SC Visualizer prototype does not provide UI mechanisms to access and manipulate those properties. However, this is considered as a purely technical shortcoming whose implementation would be a minor challenge, particularly since the SC REST API and thus the sc-angular framework already supports the manipulation of an entity's access rights.

With respect to the Social Information Flow Graph (SIFG), the prototype only implements an exemplary set of meta attributes for each kind of analysis artifact. As of now, it only displays owners and editors of artifacts and thus only a limited set of user-related attributes. As a consequence, it hardly allows the identification of new social relationships and thus new SNA use cases as claimed in Section 4.4. However, particularly the meta attributes of analysis artifacts including data model elements, analytical abstractions, and view elements are easily extendable by further ones. In the end, the set of required meta attributes depends on an actual use case, which was also named as a main conclusion by an interview partner as described in Section 6.2.

7.2.2. Discussion about the Practicability of the Prototype

In addition to concrete functional and technical limitations of the approach and prototype developed in this thesis, there are further issues related to the practicability.

In Section 2.2.1, we identify interoperability as one of the characterizing features and success factors of EUA. The EUA tool's integration into an existing application landscape and the integration of other data sources are critical aspects when applying it in practice. This was also confirmed by evaluation partners in our case studies as described in Section 6.3. For example, the enterprise architect of the second case (cf. Section 6.3.3) intends to use an EUA tool in conjunction with traditional EA tools in order to benefit from the strength of both kind of tools. At the same time, one of the main tasks of enterprise architects is the prevention of the uncontrolled growth of an application landscape and to manage its diversity (Schneider, 2015). Therefore, implementing for means for a seamless integration of a collaborative EUA approach is a major challenge.

Furthermore, our approach and prototype still has to be examined with respect to its scalability. In this context, scalability not only refers to the tool's ability to handle large data sets, but also to handle a huge number of analytical abstractions and view elements. On the one hand, scalability refers to certain technical arrangements, e.g., a scalable database. On the other hand, it also implies the need for novel conceptual means, e.g., to allow end-users to effectively explore the SIFG or the augmented model view even in the light of thousands of nodes and edges. In the context of this thesis, we already provide related suggestions of how to address the scalability issue, e.g., showing only a local perspective in the augmented model view, or enabling users to collapse certain nodes in the SIFG. However, those features were not evaluated with respect to scalability.

In two of the three interviews as described in Section 6.2, the interviewee stated that their work council would have worries when applying the SIFG. In general, this issue refers to the topic of "surveillance vs. privacy" which is also regular subject to public debate. In the concrete context of Business Intelligence and Business Analytics, the discussion about surveillance can be narrowed down to the tension between its holistic economic utility for an organization and each individual's privacy claims. In fact, Varian (2014) claims that people are willing to make certain concessions regarding their privacy as long as they "get something back" (Zuboff, 2015). With respect to the Social Information Flow Graph as introduced in this thesis, this would imply that its acceptance depends on its utility for the organization and each individual. This, in turn, depends on the actual use case the SIFG is applied in, and the actual user-related meta-information which would be necessary to capture and make explorable by co-workers. As a result, we cannot form a general opinion about the SIFG's practicability, but only discuss its potential benefits and drawbacks.

7.2.3. Critical Reflection on the Evaluation

In addition to the actual contribution of this thesis, we also want to critically reflect on methodological aspects. This particularly refers to the evaluation part of the thesis and thus the validity of the conclusion.

Different aspects of the approach and the prototype were validated with different evaluation strategies, namely an online experiment with 11 responses to validate the usability of the query formulation UI, an interview series with three practitioners of different domains and different companies to assess the utility of the SIFG (cf. Section 6.2), and a case study including three EUA cases in the domain of EAM to evaluate the prototype's practicability (cf. Section 6.3).

For the online experiment and the validation of the query formulation UI's usability we have chosen only individuals which already know MxL in order to minimize the familiarization effort with the experiment's subject. However, at the same time the specific selection of experimentees represents a potential bias and threat to the experiment's validity. Further experiments with other individuals are inevitable in order to validate the experiment's results, although this would imply a more extensive experiment including a comprehensive introduction phase.

For the second evaluation strategy, we implemented domain-themed dashboards and demonstrated the resulting SIFG to three different interviewees. Based on this demonstration, we discussed the potential utility of the SIFG if applied in the respective application domain. However, the discussions were of a hypothetical nature, since the interviewees did not use the SIFG by themselves (although they were able to ask questions about it) on the one hand, and the

SIFG did not implement all features as desired by the interviewees on the other hand. This fact represents another threat to the evaluation's validity.

In the context of the case studies, three different industry partners applied the prototype for adhoc EA analysis. Each of them created a custom dashboard based on an organization-specific data model. However, only one of them also defined reusable analytical abstractions in the form of custom functions (cf. Section 6.3.4). The others bound the visualizations directly to the data model elements and thus skipped an essential part of the prototypical solution. As a consequence, the case studies can only be considered as an evaluation of the front-end part of the prototype, namely the SC Visualizer application.

Considering all three applied strategies, the evaluation in general is primarily of a qualitative nature. Although the individual evaluation strategies—particularly the case studies—provide helpful and in-depth insights into concrete applications of the prototype in specific contexts, the evaluation's empirical foundation implies that its results are not generalizable. Furthermore, most of the evaluation partners are either enterprise architects or IT experts in the domain of EAM. Therefore, the evaluation is potentially biased towards this specific domain. Its applicability for EUA use-cases in other domains still has to be studied and evaluated.

7.2.4. Critical Reflection on the Research Methodology

As described in Section 1.3, we adopted the design science research framework (Hevner et al., 2004) in order to systematically develop the approach and prototypical implementation as described in the main part of this thesis.

While this framework and its guidelines are widely used and adopted in design science research (Gregor and Hevner, 2013), they are also subject to critics. For example, Frank (2006) discusses four flaws and misconceptions they are suffering from, namely a lack of accounting for possible future worlds, insufficient conception of a scientific foundation, a mechanistic world view, and a lack of appropriate concepts for describing the IT artifact.

For example, Hevner et al. (2004) suggests to focus on problems which are motivated by business needs as of today. Business needs which might potentially emerge in future are not captured by the framework. Regarding this, the motivation of this thesis as described in Chapter 1 is indeed based on current problems, experiences, and business needs, and thus is certainly suffering from a lack of accounting for possible future worlds. On another note, we derive the requirements for our IT artifact based on related literature and our perception of collaborative EUA (cf. Section 3.1). As a consequence, the derived requirements are certainly contingent. This fact is described by Frank (2006) as one of the key issues of the design science research framework's mechanistic world view.

Therefore, it is important to note that the results and conclusions of this thesis have to be interpreted in the light of those flaws.

7.3. Further Research

In the final section of this thesis, we outline future research opportunities which are enabled by this thesis' contribution and findings. Thereby, we also highlight potential challenges which might be faced when addressing those opportunities. Natural candidates for future research opportunities are the limitations described in Section 7.2.1. However, in the following we focus on research opportunities which are not direct implications from those technical limitations.

7.3.1. Bottom-up Approach for Defining Analytical Abstractions

The Hybrid Wiki system supports both a top-down and a bottom-up approach to data modeling (Reschenhofer et al., 2016a). The former approach means that the data model is defined by modeling experts in a first step, which in turn imposes restrictions to the subsequent creation of data objects. On the other hand, bottom-up means that first individual data objects are defined and enriched by attributes, which in turn can be used to infer a data model.

Further research could adopt this principle to analytical abstractions, e.g., derived attributes. For example, users could be enabled to define derived attributes for individual data instances, based on which a modeling expert could derive respective measures for the modeling of analytical abstractions. Specific research challenges are, e.g., merge conflicts in case of diverse definitions of the same derived attribute, and how those can be resolved in an end-user-friendly way.

7.3.2. Impersonation for Evaluation Analytical Abstractions

In the current prototypical implementation, analytical abstractions are evaluated under the identity of a currently authenticated user. This ensures that users can only access entities to which they have read access to, even by querying it with MxL. As a consequence, different users might get different results when evaluating the same MxL query.

As an alternative, future research activities could be about studying alternatives regarding the identity under which queries are evaluated in a collaborative environment in which data and data model elements are subject to authorization. Conceivable approaches would be to define an explicit identity for an analytical abstraction under which the corresponding MxL query is evaluated (e.g., the owner of the analytical abstraction), to provide a general system identity under which all queries are evaluated, or a (configurable) combination of them. In this context, performing a specific action under a different identity is called *impersonation*.

7.3.3. Alternative Evaluation Strategies for Analytical Abstractions

As of today, analytical abstractions are evaluated on demand, i.e., each time a user requests the value of an analytical abstraction, its MxL expression is evaluated.

In contrast, spreadsheets typically implement a different approach: The values of cells are persisted and (by default) only recalculated in case of changes of input cells. The static type-safety of MxL already enables the automated determination of elements which serve as input for an expression. In this sense, it already provides the foundation for a spreadsheet-like reactive evaluation strategy of analytical abstractions.

However, in our prototype the evaluation of MxL expressions is dependent on the evaluating user's identity and respective access right profile. This means that for a reactive evaluation strategy, the system does not only have to hold one value for an analytical abstraction, but potentially one for each user of the system. The number of values to be persisted for each analytical abstraction can be reduced by assigning them not to individual users, but to access right profiles which can be shared by multiple users. A user's access right profile encodes to which information artifacts a user has access to. Obviously, this strategy leads to further complexity and thus to further challenges which can be tackled in future. This research opportunity might be even more challenging if combining the implementation of alternative evaluation strategy with the aforementioned impersonation approach.

7.3.4. Further Research on End-user-driven and Model-based Data Visualizations

In this thesis, we present concepts for model-based analytics and visualizations as well as UI features allowing end-users to interact with respective implementations. Based on those foundations, future research could be about improvements or additional concepts and UI facilities for model-based analytics and visualizations.

For example, one evaluation partner explicitly asked for end-user-configurable view filters. Related challenges are, e.g., about integrating view filter concepts into our meta-model and providing UI facilities to empower users to configure different types of view filters. Furthermore, the aspect data model evaluation could be focused, e.g., by annotating recent changes to the augmented model view. This feature was a concrete suggestion by one of the reviewers during the submission process of a preliminary work (Reschenhofer and Matthes, 2016b) and creates awareness of how the data model is actually changing.

7.3.5. Further Usage Scenarios for the SIFG

In Sections 4.4 and 5.3.2.4 we describe the conceptual and technical design of the Social Information Flow Graph (SIFG) respectively. Based on its prototypical implementation on integration into the SC Visualizer, we identified potential use-cases and concerns addressable by the SIFG (cf. Section 6.2), e.g., stakeholder analysis, impact analysis, and usage analysis.

Those concerns and addressing them by a SIFG-like approach represent multiple future research opportunities. Targeting specific concerns and adapting the SIFG adequately would enable an in-depth evaluation of the SIFG. For example, Maier (2014) developed an alternative SIFG view in his master's thesis with a specific focus on impact analysis.

7.3.6. Market Place for Analytical Abstractions and Visualizations

The approach presented in this thesis enables the end-user-driven definition of reusable analytical abstractions and visualizations. Thereby, different analytics artifacts can be shared with users of the same system.

Inspired by a suggestion from an evaluation partner, we outline the idea of a market place for analytical abstractions and visualizations. Thereby, the market place represents an inter-organizational catalog of reusable components, which in turn can be deployed and downloaded by specific user groups. Future research could deal with challenges like the conceptual design of such a market place, corresponding on- and off-boarding approaches, its integration with the concepts presented in this thesis, or the socio-technical challenges and consequences of such a market place.

Bibliography

Robin Abraham and Martin Erwig. Goal-Directed Debugging of Spreadsheets. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 37–44, 2005.

Robin Abraham, Martin Erwig, Steve Kollmansberger, and Seifert Ethan. Visual Specifications of Correct Spreadsheets. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 189–196, 2005.

Robin Abraham, Margaret Burnett, and Martin Erwig. Spreadsheet Programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

Frederik Ahlemann, Eric Stettiner, Marcus Messerschmidt, and Christine Legner. *Strategic Enterprise Architecture Management*. Springer, 2012. ISBN 978-3642242229.

Pouya Aleatrati Khosroshahi, Matheus Hauder, Alexander W. Schneider, and Florian Matthes. *Enterprise Architecture Management Pattern Catalog (Version 2.0, November 2015)*. Technical Report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2015.

Yirsaw Ayalew and Roland Mittermeir. Spreadsheet Debugging. *Proceedings of the European Spreadsheet Risks Information Group Conference*, pages 67–79, 2003.

Laura Beckwith, Jácome Cunha, João Paulo Fernandes, and João Saraiva. An Empirical Study on End-Users Productivity Using Model-Based Spreadsheets. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2011.

Henrike Berthold, Philipp Rösch, Stefan Zöller, Felix Wortmann, Alessio Carenini, Stuart Campbell, Pascal Bisson, and Frank Strohmaier. An Architecture for Ad-hoc and Collaborative Business Intelligence. *Proceedings of the EDBT/ICDT Workshops*, pages 13:1–13:6, 2010.

Manoj Bhat, Thomas Reschenhofer, and Florian Matthes. A Model-Based Approach for Retrospective Analysis of Enterprise Architecture Metrics. *Proceedings of the International Conference on Enterprise Information Systems*, pages 595–611, 2015.

Stephen P. Borgatti and José Luis Molina. Ethical and Strategic Issues in Organizational Social Network Analysis. *Journal of Applied Behavioral Science*, 39(3):337–349, 2003.

Leslie Bradley and Kevin McDaid. Using Bayesian Statistical Methods to Determine the Level of Error in Large Spreadsheets. *Proceedings of the International Conference on Software Engineering*, pages 351–354, 2009.

Peter J. Brockwell and Richard A. Davis. *Time Series: Theory and Methods*. Springer Science & Business Media, 2009. ISBN 978-1441903198.

Thomas Büchner. *Introspektive Modellgetriebene Softwareentwicklung*. PhD Thesis, Technical University of Munich, Munich, 2007.

Sabine Buckl, Florian Matthes, Christian Neubert, and Christian M. Schweda. A Wiki-based Approach to Enterprise Architecture Documentation and Analysis. *Proceedings of the European Conference on Information Systems*, 2009.

Sabine Buckl, Florian Matthes, Christian Neubert, and Christian M. Schweda. A Lightweight Approach to Enterprise Architecture Modeling and Documentation. In *Information Systems Evolution*, pages 136–149. Springer, 2010. ISBN 3642177212.

Peter Buneman and Robert E. Frankel. FQL: A Functional Query Language. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 52–58, 1979.

Peter Buneman, Shamim Naqvi, Val Tannen, and Limsson Wong. Principles of Programming with Complex Objects and Collection Types. *Theoretical Computer Science*, 149(1):3–48, 1995. ISSN 0304-3975.

Patrick Bürgin. *Design and Prototypical Implementation of a Dashboard System for Visualizing Semi-Structured Data in a Traceable Way*. Master's Thesis, Technical University of Munich, Munich, Germany, 2015.

Margaret Burnett and Brad A. Myers. Future of End-User Software Engineering: Beyond the Silos. *Proceedings of the International Conference on Software Engineering*, pages 201–211, 2014.

Margaret Burnett, John Atwood, Rebecca Walpole Djang, James Reichwein, Herkimer Gottfried, and Sherry Yang. Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *Journal of Functional Programming*, 11(02):155–206, 2001.

Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. *Proceedings of the Conference on Computer Supported Cooperative Work*, pages 353–362, 2006.

Jonathan P. Caulkins, Erica Layne Morrison, and Timothy Weidemann. Spreadsheet Errors and Decision Making: Evidence from Field Interviews. *Journal of Organizational and End User Computing*, 19(3):1–23, 2007.

Chris Chambers and Christopher Scaffidi. Struggling to Excel: A Field Study of Challenges Faced by Spreadsheet Users. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 187–194, 2010.

Ed Huai-hsin Chi. A Taxonomy of Visualization Techniques using the Data State Reference Model. *IEEE Symposium on Information Visualization*, pages 69–75, 2000.

Ed Huai-hsin Chi and John Riedl. An Operator Interaction Framework for Visualization Systems. *Proceedings of the IEEE Symposium on Information Visualization*, pages 63–70, 1998.

Mei C. Chuah and Steven F. Roth. On the Semantics of Interactive Visualizations. *Proceedings of the IEEE Symposium on Information Visualization*, pages 29–36, 1996.

Ravi Chugh, Patrick M. Rondon, and Ranjit Jhala. Nested Refinements: A Logic for Duck Typing. *ACM SIGPLAN Notices*, 47(1):231–244, 2012.

Markus Clermont. *A Scalable Approach to Spreadsheet Visualization*. PhD Thesis, Universität Klagenfurt, Klagenfurt, Austria, 2003.

Edgar F. Codd. Relational Completeness of Data Base Sublanguages. *Database Systems*, pages 65–98, 1972.

M. F. Costabile, D. Fogli, C. Letondal, P. Mussio, and A. Piccinno. Domain-Expert Users and their Needs of Software Development1. *Proceedings of the International Conference on Universal Access in Human-Computer Interaction*, pages 232–236, 2003.

Rob Cross, Stephen P. Borgatti, and Andrew Parker. Making invisible work visible: Using social network analysis to support strategic collaboration. *California management review*, 44 (2):25–46, 2002. ISSN 0008-1256.

Christian Crumlish and Erin Malone. *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience*. O'Reilly Media, 2009. ISBN 978-0596154929.

Jácome Cunha. *Model-based Spreadsheet Engineering*. PhD Thesis, Universidade do Minho, Portugal, Braga, 2011.

Jácome Cunha, Martin Erwig, and João Saraiva. Automatically Inferring Classsheet Models from Spreadsheets. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 93–100, 2010.

Jácome Cunha, Jorge Mendes, João Saraiva, and João P. Fernandes. Embedding and Evolution of Spreadsheet Models in Spreadsheet Systems. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 179–186, 2011a.

Jácome Cunha, Joost Visser, Tiago Alves, and João Saraiva. Type-Safe Evolution of Spreadsheets. *Fundamental Approaches to Software Engineering*, pages 186–201, 2011b.

Jácome Cunha, João P. Fernandes, Jorge Mendes, and João Saraiva. MDSheet: A Framework for Model-driven Spreadsheet Engineering. *Proceedings of the International Conference on Software Engineering*, pages 1395–1398, 2012a.

Jácome Cunha, João P. Fernandes, Jorge Mendes, and João Saraiva. A Bidirectional Model-Driven Spreadsheet Environment. *Proceedings of the International Conference on Software Engineering*, pages 1443–1444, 2012b.

Jácome Cunha, João P. Fernandes, Jorge Mendes, and João Saraiva. Extension and Implementation of ClassSheet Models. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 19–22, 2012c.

Jácome Cunha, Joao Paulo Fernandes, Christophe Peixoto, and João Saraiva. A Quality Model for Spreadsheets. *Proceedings of the International Conference on the Quality of Information and Communications Technology*, pages 231–236, 2012d.

Jácome Cunha, João P. Fernandes, Jorge Mendes, Rui Pereira, and João Saraiva. Querying Model-Driven Spreadsheets. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 83–86, 2013.

Jácome Cunha, Martin Erwig, Jorge Mendes, and João Saraiva. Model Inference for Spreadsheets. *Automated Software Engineering*, pages 1–32, 2014a.

Jácome Cunha, João P. Fernandes, Rui Pereira, and João Saraiva. Graphical Querying of Model-Driven Spreadsheets. *Proceedings of the International Conference on Human-Computer Interaction*, pages 419–430, 2014b.

Jácome Cunha, João Paulo Fernandes, Martins Pedro, Rui Pereira, and João Saraiva. Refactoring meets Model-Driven Spreadsheet Evolution. *Proceedings of the International Conference on the Quality of Information and Communications Technology*, 2014c.

Jácome Cunha, Jorge Mendes, João Saraiva, and Joost Visser. Model-based Programming Environments for Spreadsheets. *Science of Computer Programming*, 96:254–275, 2014d.

Carlo A. Curino, Hyun J. Moon, Myung Won Ham, and Carlo Zaniolo. The PRISM Workwench: Database Schema Evolution Without Tears. *Proceedings of the International Conference on Data Engineering*, pages 1523–1526, 2009.

Thomas H. Davenport. *Thinking for a Living: How to Get Better Performances and Results from Knowledge Workers*. Harvard Business Press, 2013. ISBN 978-1591394235.

Thomas H. Davenport and D. J. Patil. Data Scientist: The Sexiest Job of the 21st Century. *Harvard Business Review*, 90(10):70–76, 2012.

Umeshwar Dayal, Ravigopal Vennelakanti, Ratnesh Sharma, Malu Castellanos, Ming Hao, and Chandrakant Patel. Collaborative Business Intelligence: Enabling Collaborative Decision Making in Enterprises. *On the Move to Meaningful*, pages 8–25, 2008.

Steven de Hertogh, Stijn Viaene, and Guido Dedene. Governing Web 2.0. *Communications of the ACM*, 54(3):124–130, 2011.

Barbara Dinter, Peter Gluchowski, and Christian Schieder. A Stakeholder Lens on Metadata Management in Business Intelligence and Big Data–Results of an Empirical Investigation. *Proceedings of the Americas Conference on Information Systems*, 2015.

Christopher Durugbo, Ashutosh Tiwari, and Jeffrey R. Alcock. Modelling Information Flow for Organisations: A Review of Approaches and Future Challenges. *International journal of information management*, 33(3):597–610, 2013. ISSN 0268-4012.

Njeru Mwendi Edwin. Software Frameworks, Architectural and Design Patterns. *Journal of Software Engineering and Applications*, 7(8):670, 2014. ISSN 1945-3124.

Martin Erwig and Gregor Engels. ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications. *Proceedings of the International Conference on Automated Software Engineering*, pages 124–133, 2005.

Arlene Fink. *How to Conduct Surveys: A Step-By-Step Guide.* Sage Publications, 6 edition, 2016. ISBN 978-1483378480.

Marc Fisher, Mingming Cao, Gregg Rothermel, Curtis Cook, and Margaret Burnett. Automated Test Case Generation for Spreadsheets. *Proceedings of the International Conference on Software Engineering*, pages 141–151, 2002.

Asbjørn Følstad. Work-Domain Experts as Evaluators: Usability Inspection of Domain-Specific Work-Support Systems. *International Journal of Human-Computer Interaction*, 22(3):217–245, 2007. ISSN 1044-7318.

Ulrich Frank. *Towards a Pluralistic Conception of Research Methods in Information Systems Research.* Technical Report, University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB), 2006.

Ulrich Frank, David Heise, Heiko Kattenstroth, and Hanno Schauer. Designing and Utilising Business Indicator Systems within Enterprise Models – Outline of a Method. *Proceedings of the Conference on Modellierung betrieblicher Informationssysteme*, pages 89–105, 2008.

Ulrich Frank, David Heise, and Heiko Kattenstroth. Use of a Domain Specific Modeling Language for Realizing Versatile Dashboards. *Proceedings of the OOPSLA Workshop on Domain-Specific Modeling (DSM)*, 2009.

Linton Freeman. *The Development of Social Network Analysis.* Createspace, Vancouver, 2004. ISBN 978-1594577147.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994. ISBN 978-0201633610.

Gartner Inc. Magic Quadrant for Business Intelligence and Analytics Platforms, 2015. URL `https://www.gartner.com/doc/2989518/magic-quadrant-business-intelligence-analytics`.

Robert Geisler, Marcus Klar, and Claudia Pons. Dimensions and Dichotomy in Metamodeling. *Proceedings of the Conference on Northern Formal Methods*, 1998.

Athula Ginige, Luca Paolino, Monica Sebillo, Richa Shrodkar, and Giuliana Vitiello. User Requirements for a Web Based Spreadsheet-Mediated Collaboration. *Proceedings of the International Conference on Advanced Visual Interfaces*, pages 133–136, 2010a.

Athula Ginige, Luca Paolino, Monica Sebillo, Genoveffa Tortora, Marco Romano, and Giuliana Vitiello. A Collaborative Environment for Spreadsheet-Based Activities. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 273–274, 2010b.

Shirley Gregor and Alan R. Hevner. Positioning and Presenting Design Science Research for Maximum Impact. *Management Information Systems Quarterly*, 37(2):337–355, 2013.

Thomas A. Grossman, Vijay Mehrotra, and Özgür Özlük. Lessons from Mission-Critical Spreadsheets. *Communications of the Association for Information Systems*, 20(1):60, 2007.

Tom Grossman and Erik Burd. Towards Reusable Spreadsheet Code: Experiments to Create and Interchange Encapsulated Excel Modules. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2015.

Penny Grubb and Armstrong A. Takang. *Software Maintenance: Concepts and Practice*. World Scientific, 2003. ISBN 9814485616.

Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

Torsten Grust and Marc H. Scholl. How to Comprehend Queries Functionally. *Journal of Intelligent Information Systems*, 12(2-3):191–218, 1999.

Seda Gürses and Claudia Diaz. Two Tales of Privacy in Online Social Networks. *IEEE Security & Privacy*, 11(3):29–37, 2013. ISSN 1540-7993.

Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl. Visual Querying of Linked Data with QueryVOWL. *Joint Proceedings of SumPre*, pages 2014–2015, 2015.

Maria J. J. Hall. A Risk and Control-Oriented Study of the Practices of Spreadsheet Application Developers. *Proceedings of the Hawaii International Conference on System Sciences*, pages 364–373, 1996.

Matheus Hauder. *Empowering Users to Collaboratively Structure Knowledge-Intensive Processes*. PhD Thesis, Technical University of Munich, Munich, Germany, 2015.

Matheus Hauder, Florian Matthes, Sascha Roth, and Christopher Schulz. Generating Dynamic Cross–Organizational Process Visualizations through Abstract View Model Pattern Matching. *Proceedings of the International Conference on Interoperability for Enterprises Systems and Applications*, pages 95–101, 2012.

Matheus Hauder, Sascha Roth, Christopher Schulz, and Florian Matthes. Current Tool Support for Metrics in Enterprise Architecture Management. *Proceedings of the DASMA Software Metrik Kongress*, 2013.

Jeffrey Heer and Danah Boyd. Vizster: Visualizing Online Social Networks. *IEEE Symposium on Information Visualization (INFOVIS)*, pages 32–39, 2005.

Anders Heijlsberg and Mads Torgersen. Standard Query Operators Overview, 2013. URL `http://msdn.microsoft.com/en-us/library/bb397896.aspx`.

Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

Felienne Hermans. *Analyzing and Visualizing Spreadsheets.* PhD Thesis, Technische Universiteit Delft, 2012.

Felienne Hermans and Emerson Murphy-Hill. Enron's Spreadsheets and Related Emails: A Dataset and Analysis. *Proceedings of the International Conference on Software Engineering*, pages 7–16, 2015.

Felienne Hermans, Martin Pinzger, and Arie van Deursen. Automatically Extracting Class Diagrams from Spreadsheets. *Proceedings of the European Conference on Object-Oriented Programming*, pages 52–75, 2010.

Felienne Hermans, Martin Pinzger, and Arie van Deursen. Breviz: Visualizing Spreadsheets using Dataflow Diagrams. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2011a.

Felienne Hermans, Martin Pinzger, and Arie van Deursen. Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams. *Proceedings of the International Conference on Software Engineering*, pages 451–460, 2011b.

Felienne Hermans, Martin Pinzger, and Arie van Deursen. Measuring Spreadsheet Formula Understandability. *Proceedings of the European Spreadsheet Risks Information Group Conference*, pages 77–96, 2012.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *Management Information Systems Quarterly*, 28(1):75–105, 2004.

Karin Hodnigg and Roland T. Mittermeir. Metrics-Based Spreadsheet Visualization: Support for Focused Maintenance. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2008.

Volker Hoyer, Katarina Stanoesvka-Slabeva, Till Janner, and Christoph Schroth. Enterprise Mashups: Design Principles towards the Long Tail of User Needs. *Proceedings of the International Conference on Services Computing*, pages 601–602, 2008.

Kai M. Hüner, Boris Otto, and Hubert Österle. Collaborative Management of Business Metadata. *International journal of information management*, 31(4):366–373, 2011. ISSN 0268-4012.

International Organization for Standardization. ISO/IEC 42010:2007 Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems, 2007. URL `http://dx.doi.org/10.1109/ieeestd.2007.386501`.

Petra Isenberg, Anthony Tang, and Sheelagh Carpendale. An Exploratory Study of Visual Information Analysis. *Proceedings of the Conference on Human Factors in Computing Systems*, pages 1217–1226, 2008.

James Kalbach. *Designing Web Navigation: Optimizing the User Experience.* O'Reilly Media, 2007. ISBN 978-0596528102.

Bennett Kankuzi and Yirsaw Ayalew. An End-User Oriented Graph-Based Visualization for Spreadsheets. *Proceedings of the International Workshop on End-User Software Engineering*, pages 86–90, 2008.

Bennett Kankuzi and Jorma Sajaniemi. An Empirical Study of Spreadsheet Authors' Mental Models in Explaining and Debugging Tasks. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 15–18, 2013.

Florian Katenbrink. *Optimizing the User Experience of a Social Content Manager for Casual Users.* Bachelor's Thesis, Technical University of Munich, Munich, Germany, 2015.

Jens Kaufmann and Peter Chamoni. Structuring Collaborative Business Intelligence: A Literature Review. *Proceedings of the Hawaii International Conference on System Sciences*, pages 3738–3747, 2014.

Vijay Khatri and Carol V. Brown. Designing Data Governance. *Communications of the ACM*, 53(1):148–152, 2010.

Martin Kilduff and Daniel J. Brass. Organizational Social Network Research: Core Ideas and Key Debates. *The Academy of Management Annals*, 4(1):317–357, 2010. ISSN 1941-6520.

Won Kim, Elisa Bertino, and Jorge F. Garza. Composite Objects Revisited. *ACM SIGMOD Record*, 18(2):337–347, 1989.

Bryan Klimt and Yiming Yang. Introducing the Enron Corpus. *CEAS*, 2004.

Brian Knight, David Chadwick, and Kamalesen Rajalingham. A Structured Methodology for Spreadsheet Modelling. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2000.

Markus Krötzsch, Denny Vrandečić, and Max Völkel. Semantic Mediawiki. *Proceedings of the International Semantic Web Conference*, pages 935–942, 2006.

Steve Krug. *Don't Make Me Think Revisited: A Common Sense Approach to Web and Mobile Usability.* New Riders Publishing, 3 edition, 2014. ISBN 9780321965516.

Robert Lagerström, Carliss Young Baldwin, Alan D. Maccormack, and Stephan Aier. Visualizing and Measuring Enterprise Application Architecture: An Exploratory Telecom Case. *Proceedings of the Hawaii International Conference on System Sciences*, pages 3847–3856, 2014.

Barry R. Lawson, Kenneth R. Baker, Stephen G. Powell, and Lynn Foster-Johnson. A Comparison of Spreadsheet Users with Different Levels of Experience. *Omega*, 37(3):579–590, 2009.

Linda A. Leon, Dolphy M. Abraham, and Lawrence Kalbers. Beyond Regulatory Compliance for Spreadsheet Controls: A Tutorial to Assist Practitioners and a Call for Research. *Communications of the Association for Information Systems*, 27(28):541–560, 2010.

Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf, editors. *End User Development.* Human-Computer Interaction Series. Springer, 9 edition, 2006. ISBN 978-1-4020-5309-2.

Javier López, Fernando Bellas, Alberto Pan, and Paula Montoto. A Component-based Approach for Engineering Enterprise Mashups. *Proceedings of the International Conference on Web Engineering*, pages 30–44, 2009.

David Lyon. *Surveillance as social sorting: Privacy, risk, and digital discrimination.* Psychology Press, 2003. ISBN 0415278732.

Matti Maier. *A Concept for the Visual and Interactive Impact Analysis and Simulation of Data Changes to Enterprise Metrics.* Master's Thesis, Technical University of Munich, Munich, Germany, 2014.

David Martin and Ian Sommerville. Patterns of Cooperative Interaction: Linking Ethnomethodology and Design. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 11(1):59–89, 2004. ISSN 1073-0516.

Heiko Matheis. SmartNet Navigator and Application Guidelines. *Seventh Framework Programme*, 2013.

Florian Matthes and Christian Neubert. Wiki4EAM - Using Hybrid Wikis for Enterprise Architecture Management. *Proceedings of the International Symposium on Wikis and Open Collaboration*, page 226, 2011.

Florian Matthes, Sabine Buckl, Jana Leitel, and Christian M. Schweda. *Enterprise Architecture Management Tool Survey.* Technical Report, Technical University of Munich, Munich, Germany, 2008.

Florian Matthes, Christian Neubert, and Alexander Steinhoff. Hybrid Wikis: Empowering Users to Collaboratively Structure Information. *Proceedings of the International Conference on Software and Data Technologies*, pages 250–259, 2011.

Florian Matthes, Ivan Monahov, Alexander Schneider, and Christopher Schulz. *EAM KPI Catalog v1.0.* Technical Report, Technical University of Munich, Munich, Germany, 2012a. URL `http://wwwmatthes.in.tum.de/pages/19kw70p0u5vwv/EAM-KPI-Catalog`.

Florian Matthes, Ivan Monahov, Alexander W. Schneider, and Christian Schulz. Towards a Unified and Configurable Structure for EA Management KPIs. *Proceedings of the Trends in Enterprise Architecture Research Workshop*, 2012b.

Simon McGinnes and Evangelos Kapros. Conceptual Independence: A Design Principle for the Construction of Adaptive Information Systems. *Information Systems*, 47:33–50, 2015.

Matthias Mertens and Tobias Krahn. Knowledge Based Business Intelligence for Business User Information Self-Service. In Stefan Bruggemann, editor, *Collaboration and the Semantic Web*, pages 271–296. Information Science Reference, 2012.

Björn Michelsen. *Implementing a Web Client for Social Content and Task Management.* Master's Thesis, Technical University of Munich, Munich, Germany, 2016.

Ivan Monahov. *Integrated Software Support for Quantitative Models in the Domain of Enterprise Architecture Management.* PhD Thesis, Technical University of Munich, Munich, 2014.

Ivan Monahov, Thomas Reschenhofer, and Florian Matthes. Design and Prototypical Implementation of a Language Empowering Business Users to Define Key Performance Indicators for Enterprise Architecture Management. *Proceedings of the Trends in Enterprise Architecture Research Workshop*, 2013.

Anders I. Mørch, Gunnar Stevens, Markus Won, Markus Klann, Yvonne Dittrich, and Volker Wulf. Component-based Technologies for End-user Development. *Communications of the ACM*, 47(9):59–62, 2004.

Renate Motschnig-Pitrik and Jens Kaasboll. Part-Whole Relationship Categories and Their Application in Object-Oriented Analysis. *IEEE Transactions on Knowledge and Data Engineering*, 11(5):779–797, 1999. ISSN 1041-4347.

Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, 1993. ISBN 0262140535.

Bonnie A. Nardi and James R. Miller. An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development. *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 197–208, 1990.

Christian Neubert. *Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms*. PhD Thesis, Technical University of Munich, Munich, 2012.

Object Management Group. Meta Object Facility (MOF) v2.4.2, 2014a. URL `http://www.omg.org/spec/MOF/2.4.2/`.

Object Management Group. Object Constraint Language (OCL), 2014b. URL `http://www.omg.org/spec/OCL/2.4`.

Object Management Group. Unified Modeling Language (UML), 2015. URL `http://www.omg.org/spec/UML/2.5`.

Immanuel Pahlke, Roman Beck, and Wolf Martin. Enterprise Mashup Systems as Platform for Situational Applications. *Business & Information Systems Engineering*, 2(5):305–315, 2010.

Raymond R. Panko. Applying Code Inspection to Spreadsheet Testing. *Journal of Management Information Systems*, pages 159–176, 1999.

Raymond R. Panko. Facing the Problem of Spreadsheet Errors. *Decision Line*, 37(5):8–10, 2006.

Raymond R. Panko. What We Don't Know About Spreadsheet Errors Today. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2015.

Raymond R. Panko and Richard P. Halverson Jr. Individual and Group Spreadsheet Design: Patterns of Errors. *Proceedings of the Hawaii International Conference on System Sciences*, pages 4–10, 1994.

Raymond R. Panko and Richard P. Halverson Jr. Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks. *Proceedings of the Hawaii International Conference on System Sciences*, pages 326–335, 1996.

Raymond R. Panko and Richard P. Halverson Jr. Are Two Heads Better than One? (At Reducing Errors in Spreadsheet Modeling). *Office Systems Research Journal*, 15(1):21–32, 1997. ISSN 0737-8998.

Raymond R. Panko and Richard P. Halverson Jr. An Experiment in Collaborative Spreadsheet Development. *Journal of the Association for Information Systems*, 2(1):4, 2001.

Raymond R. Panko and Nicholas Ordway. Sarbanes-Oxley: What About all the Spreadsheets? *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2005.

Raymond R. Panko and Daniel N. Port. End User Computing: The Dark Matter (and Dark Energy) of Corporate IT. *Journal of Organizational and End User Computing*, pages 4603–4612, 2012.

Michael Quinn Patton. *Qualitative Evaluation and Research Methods.* Sage Publications, 3 edition, 2002. ISBN 978-0761919711.

Paul A. Pavlou and Omar A. El Sawy. Understanding the Elusive Black Box of Dynamic Capabilities. *Decision Sciences*, 42(1):239–273, 2011. ISSN 1540-5915.

Jon D. Pemberton and Andrew J. Robson. Spreadsheets in Business. *Industrial Management & Data Systems*, 100(8):379–388, 2000.

Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, Cambridge, Mass, 2002. ISBN 978-0262162098.

William Pinnington, Ben Light, and Elaine Ferneley. Too Much of a Good Thing? A Field Study of Challenges in Business Intelligence Enabled Enterprise System Environments. *Proceedings of the European Conference on Information Systems*, pages 1941–1952, 2007.

Stephen G. Powell, Kenneth R. Baker, and Barry Lawson. A Critical Review of the Literature on Spreadsheet Errors. *Decision Support Systems*, 46(1):128–138, 2008.

Stephen G. Powell, Kenneth R. Baker, and Barry Lawson. Impact of Errors in Operational Spreadsheets. *Decision Support Systems*, 47(2):126–132, 2009.

Kamalasen Rajalingham, David Chadwick, Brian Knight, and Dilwyn Edwards. Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development. *Proceedings of the Hawaii International Conference on System Sciences*, pages 10–18, 2000.

Sven Rehm, Thomas Reschenhofer, and Klym Shumaiev. IS Design Principles for Empowering Domain Experts in Innovation: Findings From Three Case Studies. *Proceedings of the International Conference on Information Systems*, 2014.

Thomas Reschenhofer. *Design and Prototypical Implementation of a Model-based Structure for the Definition and Calculation of Enterprise Architecture Key Performance Indicators.* Master's Thesis, Technical University of Munich, Munich, Germany, 2013.

Thomas Reschenhofer and Florian Matthes. An Empirical Study on Spreadsheet Shortcomings from an Information Systems Perspective. *Proceedings of the International Conference on Business Information Systems*, pages 50–61, 2015a.

Thomas Reschenhofer and Florian Matthes. A Framework for the Identification of Spreadsheet Usage Patterns. *Proceedings of the European Conference on Information Systems*, 2015b.

Thomas Reschenhofer and Florian Matthes. Empowering End-users to Collaboratively Manage and Analyze Evolving Data Models. *Proceedings of the Americas Conference on Information Systems*, 2016a.

Thomas Reschenhofer and Florian Matthes. Supporting End-Users in Defining Complex Queries on Evolving and Domain-Specific Data Models. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 96–100, 2016b.

Thomas Reschenhofer, Ivan Monahov, and Florian Matthes. Application of a Domain-Specific Language to Support the User-Oriented Definition of Visualizations in the Context of Collaborative Product Development. *Proceedings of the International Conference on Interoperability for Enterprises Systems and Applications*, pages 164–169, 2014a.

Thomas Reschenhofer, Ivan Monahov, and Florian Matthes. Type-Safety in EA Model Analysis. *Proceedings of the International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, pages 87–94, 2014b.

Thomas Reschenhofer, Manoj Bhat, Adrian Hernandez-Mendez, and Florian Matthes. Lessons Learned in Aligning Data and Model Evolution in Collaborative Information Systems. *Proceedings of the International Conference on Software Engineering*, pages 132–141, 2016a.

Thomas Reschenhofer, Patrick Bürgin, and Florian Matthes. A Social Information Flow Graph - Design and Prototypical Implementation. *Proceedings of the International Conference on Advanced Information Systems Engineering Forum*, pages 137–144, 2016b.

Thomas Reschenhofer, Sirma Gjorgievska, Bernhard Waltl, and Florian Matthes. A Semantic Meta Model of Spreadsheets. *Proceedings of the European Conference on Information Systems*, 2016c.

Thomas Reschenhofer, Bernhard Waltl, Klym Shumaiev, and Florian Matthes. A Conceptual Model for Measuring the Complexity of Spreadsheets. *Proceedings of the European Spreadsheet Risks Information Group Conference*, 2016d.

Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

Stefano Rizzi. Collaborative Business Intelligence. In *Business Intelligence*, pages 186–205. Springer, 2012. ISBN 3642273572.

Andrew Robson and Jonathan Pemberton. Spreadsheet Selection: Good Luck or Good Management? *Logistics Information Management*, 9(3):49–56, 1996. ISSN 0957-6053.

Boaz Ronen, Michael A. Palley, and Lucas Jr, Henry C. Spreadsheet Analysis and Design. *Communications of the ACM*, 32(1):84–93, 1989.

Sascha Roth. *Federated Enterprise Architecture Model Management: Conceptual Foundations, Collaborative Model Integration, and Software Support.* PhD Thesis, Technical University of Munich, Munich, Germany, 2014.

Sascha Roth, Matheus Hauder, Marin Zec, Alexej Utz, and Florian Matthes. Empowering Business Users to Analyze Enterprise Architectures: Structural Model Matching to Configure Visualizations. *Proceedings of the International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, 2013.

Sascha Roth, Marin Zec, and Florian Matthes. *Enterprise Architecture Visualization Tool Survey 2014*. epubli GmbH, Berlin, 2014. ISBN 978-3844289381.

Karen Rothermel, Curtis Cook, Margaret Burnett, Justin Schonfeld, Green, Thomas R. G., and Gregg Rothermel. WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation. *Proceedings of the International Conference on Software Engineering*, pages 230–239, 2000.

Herbert J. Rubin and Irene S. Rubin. *Qualitative Interviewing: The Art of Hearing Data*. Sage Publications, 2011. ISBN 978-1412978378.

Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the Numbers of End Users and End User Programmers. *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, pages 207–214, 2005.

Michael Schaub, Florian Matthes, and Sascha Roth. Towards a Conceptual Framework for Interactive Enterprise Architecture Management Visualizations. *Modellierung*, 2012.

Alexander W. Schneider. *Decision Support for Application Landscape Diversity Management*. PhD Thesis, Technical University of Munich, Munich, Germany, 2015.

Alexander W. Schneider, Thomas Reschenhofer, Alexander Schütz, and Florian Matthes. Empirical Results for Application Landscape Complexity. *Proceedings of the Hawaii International Conference on System Sciences*, pages 4079–4088, 2015.

Tobias Schrade. *Implementing a Web Client for Integrated Data, Role, Function, and Task Modelling*. Master's Thesis, Technical University of Munich, Munich, Germany, 2016.

Alexander Schuetz, Thomas Widjaja, and Jasmin Kaiser. Complexity in Enterprise Architectures - Conceptualization And Introduction of a Measure from a System Theoretic Perspective. *Proceedings of the European Conference on Information Systems*, 2013.

Bill Scott and Theresa Neil. *Designing Web Interfaces: Principles and Patterns for Rich Interactions*. O'Reilly Media, 2009. ISBN 978-0-596-51625-3.

Roger S. Scowen. Extended BNF – A Generic Base Standard. *Proceedings of the International Symposium on Software Engineering Standards*, 1993.

Denilson Sell, Liliana Cabral, Enrico Motta, John Domingue, Farshad Hakimpour, and Roberto Pacheco. A Semantic Web based Architecture for Analytical Tools. *Proceedings of the International Conference on E-Commerce Technology*, pages 347–354, 2005.

James A. Senn. *Information Technology: Principles, Practices, and Opportunities*. Pearson Prentice Hall, Upper Saddle River, N.J., 3rd ed. edition, 2004. ISBN 9780131246812.

Peter Sestoft. *Spreadsheet Technology*. Technical Report, IT University of Copenhagen, Copenhagen, Denmark, 2011.

GANESAN Shankaranarayanan and Yu Cai. Supporting Data Quality Management in Decision-Making. *Decision Support Systems*, 42(1):302–317, 2006.

Claude Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, pages 379–423, 1948.

HongHai Shen and Prasun Dewan. Access Control for Collaborative Environments. *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1992:51–58, 1992.

Hidekazu Shiozawa, Ken-ichi Okada, and Yutaka Matsushita. 3D Interactive Visualization for Inter-Cell Dependencies of Spreadsheets. *Proceedings of the IEEE Symposium on Information Visualization*, pages 79–82, 1999.

Forrest Shull, Jeffrey Carver, and Guilherme H. Travassos. An Empirical Methodology for Introducing Software Processes. *ACM SIGSOFT Software Engineering Notes*, 26(5):288–296, 2001.

Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance in e-Science. *ACM SIGMOD Record*, 34(3):31–36, 2005.

Richard Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems (TODS)*, 12(2):247–298, 1987. ISSN 0362-5915.

Javier Soriano, David Lizcano, Miguel A. Cañas, Marcos Reyes, and Juan J. Hierro. Fostering Innovation in a Mashup-Oriented Enterprise 2.0 Collaboration Environment. *Proceedings of the International Conference on Adaptive Business Systems*, 24(2007):62–68, 2007.

Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jimenez-Ruiz, Martin Giese, and Ian Horrocks. Ontology-Based Visual Query Formulation: An Industry Experience. In *Advances in Visual Computing*, pages 842–854. Springer, 2015. ISBN 3319278568.

Michael Spahn, Christian Dörner, and Volker Wulf. End User Development of Information Artefacts: A Design Challenge for Enterprise Systems. *Proceedings of the European Conference on Information Systems*, pages 482–493, 2008a.

Michael Spahn, Hoachim Kleb, Stephan Grimm, and Stefan Scheidl. Supporting Business Intelligence by Providing Ontology-based End-user Information Self-service. *Proceedings of the International Workshop on Ontology-Supported Business intelligence*, 2008:10:1–10:12, 2008b.

Harald Störrle. VMQL: A Visual Language for Ad-Hoc Model Querying. *Journal of Visual Languages & Computing*, 22(1):3–29, 2011.

Harald Störrle. Improving the Usability of OCL as an Ad-hoc Model Querying Language. *Proceedings of the International Workshop on OCL, Model Constraint and Query Languages*, pages 83–92, 2013.

Gunnvald B. Svendsen. The Influence of Interface Style on Problem Solving. *International Journal of Man-Machine Studies*, 35(3):379–397, 1991. ISSN 0020-7373.

Toomas Tamm, Peter Seddon, and Graeme Shanks. Pathways to Value from Business Analytics. *Proceedings of the International Conference on Information Systems*, 2013.

Steven L. Tanimoto. VIVA: A Visual Language for Image Processing. *Journal of Visual Languages & Computing*, 1(2):127–139, 1990.

David J. Teece, Gary Pisano, and Amy Shuen. Dynamic Capabilities and Strategic Management. *Strategic management journal*, pages 509–533, 1997. ISSN 0143-2095.

Anca Vaduva and Thomas Vetterli. Metadata Management for Data Warehousing: An Overview. *Journal of Cooperative Information Systems*, 10(3):273–298, 2001.

Rafael Valencia-García, Francisco García-Sánchez, Dagoberto Castellanos-Nieves, and Jesualdo Tomás Fernández-Breis. OWLPath: An OWL Ontology-guided Query Editor. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(1):121–136, 2011. ISSN 1083-4427.

Marcel van Oosterhout, Eric Waarts, and Jos van Hillegersberg. Change Factors Requiring Agility and Implications for IT. *European Journal of Information Systems*, 15(2):132–145, 2006.

Peter van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming.* MIT Press, 2004. ISBN 0262220695.

Hal R. Varian. Beyond Big Data. *Business Economics*, 49(1):27–31, 2014. ISSN 0007-666X.

Peter Velten. *Avoiding Redundancy in the Management of Technical Documentation and Models: Requirements Analysis and Prototypical Implementation for Enterprise Architecture Management.* Master's Thesis, Technical University of Munich, Munich, Germany, 2016.

Padmal Vitharana. Risks and Challenges of Component-based Software Development. *Communications of the ACM*, 46(8):67–72, 2003.

Holger Wache, Thomas. Vögele, Ubbo. Visser, Heiner Stuckenschmidt, G. Schuster, H.. Neumann, and S. Hübner. Ontology-Based Integration of Information: A Survey of Existing Approaches. *Proceedings of the IJCAI Workshop on Ontologies and Information Sharing*, pages 108–117, 2001.

Valérianne Walter. *Analyzing State-of-the-art Self-Service BI Tools.* Bachelor's Thesis, Technical University of Munich, Munich, Germany, 2015.

Bernhard Waltl, Thomas Reschenhofer, and Florian Matthes. Data Governance on EA Information Assets: Logical Reasoning for Derived Data. *Proceedings of the Trends in Enterprise Architecture Research Workshop*, 2015.

Bernhard Waltl, Florian Matthes, Tobias Waltl, and Thomas Grass. LEXIA - A Data Science Environment for Semantic Analysis of German Legal Texts. *Proceedings of the Internationales Rechtsinformatik Symposium*, 2016.

Eric M. Wilcox, John Atwood, Margaret Burnett, Jonathan Cadiz, and Curtis Cook. Does Continuous Visual Feedback Aid Debugging in Direct-Manipulation Programming Systems? *Proceedings of the Conference on Human Factors in Computing Systems*, pages 258–265, 1997.

Ryan Wisnesky. *Functional Query Languages with Categorical Types.* PhD Thesis, Harvard University, Cambridge, Massachusetts, 2014.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering.* Springer Science & Business Media, 2012. ISBN 978-3642290435.

Volker Wulf and Matthias Jarke. The Economics of End-User Development. *Communications of the ACM*, 47(9):41–42, 2004.

Volker Wulf and Markus Rohde. Towards an Integrated Organization and Technology Development. *Proceedings of the Conference on Designing Interactive Systems*, pages 55–64, 1995.

Robert K. Yin. *Case Study Research: Design and Methods.* Sage Publications, 5 edition, 2014. ISBN 1483302008.

Shoshana Zuboff. Big Other: Surveillance Capitalism and the Prospects of an Information Civilization. *Journal of Information Technology*, 30(1):75–89, 2015.

Fritz Zwicky. *Discovery, Invention, Research - Through the Morphological Approach.* The Macmillian Company, Toronto, 1969.

Abbreviations

ACM	Adaptive Case Management
AIS	Adaptive Information System
ALM	Application Lifecycle Management
API	Application Programming Interface
AST	Abstract Syntax Tree
BA	Business Analytics
BI	Business Intelligence
CRUD	Create-Read-Update-Delete
CS	Computer Science
CSS	Cascading Style Sheets
DAO	Data Access Object
DEC	Domain Expert Configuration
DSML	Domain-Specific Modeling Language

DQM	Data Quality Management
EA	Enterprise Architecture
EAM	Enterprise Architecture Management
EBNF	Extended Backus-Naur Form
EUA	End-User Analytics
EUD	End-User Development
EM	Enterprise Mashup
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ICT	Information and Communication Technology
IS	Information System
IT	Information Technology
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
KPI	Key Performance Indicator
LALR	Look-Ahead, Left to Right
LINQ	Language Integrated Queries
MOF	Meta Object Facility
MVC	Model View Controller
MxL	Model-based Expression Language

NPD	New Product Development
OCL	Object Constraint Language
PDF	Portable Document Format
PMIS	Performance Management Information System
REST	Representational State Transfer
RSA	Rivest-Shamir-Adleman
SC	SocioCortex
SE	Software Engineering
sebis	Chair of Software Engineering of Business Information Systems
SIFG	Social Information Flow Graph
SNA	Social Network Analysis
TUM	Technical University of Munich
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
ViTSL	Visual Template Specification Language
WYSIWYG	"What You See Is What You Get"
WYSIWYT	"What You See Is What You Test"

APPENDIX A

Appendix

Basic MxL Functions

By default, MxL supports a couple of basic functions which are described in the following tables. They are derived from Microsoft's Standard Query Operators (Heijlsberg and Torgersen, 2013) on the one hand, and from four years of applying the system for different kinds of analysis tasks (particularly EA analysis, cf. Section 4.2.3).

Name and Description	**Parameters**	**Returns**
toString		
Returns a string representation of the current object.		String
transitive		
Performs a depth-first traversion over all descendants of the current object, whereas the descendants are determined by the given map-function.	map : Function<T, Sequence<T> >	Sequence<T>

Table A.1.: Basic **object** functions. In this table, all functions are applied on an object of type T, whereas T is an arbitrary *MxlType*.

Name and Description	**Parameters**	**Returns**
date		
Parses the given string to a date.	dateAsString : date	Date
exp		
Returns Euler's number e raised to the power of the given value.	n : Number	Number
sequence		
Returns a sequence of numbers ranging from the *from* value (inclusive) to the *to* value (inclusive), whereas the optional *interval* defines the step size (default is 1).	from : Number to : Number interval? : Number	Sequence<Number>
ln		
Returns the natural logarithm of the given number.	n : Number	Number
sqrt		
Returns the positive square root of the given number.	n : Number	Number

Table A.2.: Basic **static** functions. In this table, all functions are static, i.e., they are not applied to any object.

Name and Description	**Parameters**	**Returns**
format		
Formats the current number based on a formatting string (e.g. "#.##").	formatStr : String	String
intdiv		
Integer division of the current number through the given one.	operand : Number	Number
mod		
Modulo of the integer division of the current number through the given one.	operand : Number	Number

Table A.3.: Basic **number** functions. In this table, all functions are applied on an object of type *Number*.

Name and Description	Parameters	Returns
length		
Returns the number of characters in the current string.		Number
match		
Returns true, if the current string matches the given regular exrpression, and otherwise false.	regex : String	Boolean
split		
Splits this string around matches of the given delimiter.	delimiter : String	Sequence<String>
substring		
Returns a new string that is a substring of the current one string. The substring begins at the specified start.	start : Number end? : Number	String

Table A.4.: Basic **string** functions. In this table, all functions are applied on an object of type *String*.

Name and Description	Returns
asSequence	
Returns all key-value-pairs of the map as a sequence of structures.	Sequence<Structure<key : K, value : V> >

Table A.5.: Basic **map** functions. In this table, all functions are applied on an object of type *Map<K,V>*, whereas *K* and *V* are arbitrary *MxlTypes*.

Name and Description	Returns
asSequence	
Returns a sequence of structures containing the name and the value of each of the source structure's attributes	Sequence<Structure<key:String,value:Object> >

Table A.6.: Basic **structure** functions. In this table, all functions are applied on an object of type *Structure*.

Name and Description	**Parameters**	**Returns**
select		
Applies the map-function to each element of the source sequence and returns a sequence containing the results of each individual application.	map : Function<T, U>	Sequence<U>
selectMany		
Similar to the select-function, however, in selectMany, the map-function returns a sequence for each element. The concatenation of all sequences forms the result of the selectMany-function.	map : Function<T, Sequence<U> >	Sequence<U>
where		
Filters the source sequence by the given predicate, i.e., all elements fulfilling the predicate remain in the sequence.	pred : Function<T, Boolean>	Sequence<T>
orderby		
Sorts the source sequence by the (optional) keySel-function, whereas a natural order will be applied. The (optional) descending parameter determines, if the elements should be ordered ascending (default) or descending. The default value for the keySel-parameter is the identity function.	keySel? : Function<T, Object> descending? : Boolean	Sequence<T>
groupby		
Groups the elements of the source list by the keySel-Function and applies the (optional) map-function on the elements of each single group. The default value for the keySel- and map-parameter is the identity function.	keySel? : Function<T, K> map? : Function<Sequence<T>,V>	Map<K,V>

Table A.7.: **Common query** functions. In this table, all functions are applied on an object of type *Sequence<T>*, whereas *T*, *U*, *K*, and *V* are arbitrary *MxlTypes*.

Name and Description	**Parameters**	**Returns**
any		
Returns *true*, if at least one element of the source sequence fulfills the given predicate, otherwise *false*.	pred : Function<T, Boolean>	Boolean
all		
Returns *true*, if each element of the source sequence fulfills the given predicate, otherwise *false*.	pred : Function<T, Boolean>	Boolean
none		
Returns *true*, if no element of the source sequence fulfills the given predicate, otherwise *false*.	pred : Function<T, Boolean>	Boolean
contains		
Returns *true*, if the given element is contained in the source sequence, otherwise *false*.	element : T	Boolean
isEmpty		
Returns *true*, if the source sequence has no elements, otherwise *false*.		Boolean
isNotEmpty		
Returns *true*, if the source sequence has at least one element, otherwise *false*.		Boolean

Table A.8.: **Quantifier** functions mapping a sequence to a boolean value. In this table, all functions are applied on an object of type *Sequence<T>*, whereas *T* is an arbitrary *MxlType*.

Name and Description	**Parameters**	**Returns**
distinct		
Removes all duplicates of the source sequence.		Sequence<T>
except		
Returns a sequence with all elements contained in the source sequence, but not in the other one.	other : Sequence<T>	Sequence<T>
intersect		
Returns a sequence with all elements contained in the source sequence and in the other one.	other : Sequence<T>	Sequence<T>
concat		
Concatenates the source sequence with the other one, i.e., the resulting sequence contains all elements of the source sequence, followed by all elements of the other one.	other : Sequence<T>	Sequence<T>

Table A.9.: **Set** functions based on the presence or absence of elements in the same or another sequence. In this table, all functions are applied on an object of type *Sequence<T>*, whereas *T* is an arbitrary *MxlType*.

Name and Description	**Parameters**	**Returns**
rest		
Returns the source sequence without the first element.		Sequence<T>
take		
Returns a sequence with the first *n* elements of the source sequence.	n : Number	Sequence<T>
takeWhile		
Returns all elements of the source sequence until an element does not satisfy the predicate.	pred : Function<T,Boolean>	Sequence<T>
skip		
Returns a sequence without the first *n* elements of the source sequence.	n : Number	Sequence<T>
skipWhile		
Skips all elements of the source sequence as long as these elements satisfy the predicate, and returns the rest.	pred : Function<T,Boolean>	Sequence<T>

Table A.10.: **Partitioning** functions returning a subsequence. In this table, all functions are applied on an object of type *Sequence<T>*, whereas *T* is an arbitrary *MxlType*.

Name and Description	**Parameters**	**Returns**
first		
Returns the first element of the source sequence (or the first element satisfying the predicate). If there is not such an element, this function throws an exception.	pred? : Function<T,Boolean>	T
last		
Returns the last element of the source sequence (or the last element satisfying the predicate). If there is not such an element, this function throws an exception.	pred? : Function<T,Boolean>	T
single		
Returns the only element of the source sequence (or the only element satisfying the predicate). If there is not such an element, or if there is more than one element, this function throws an exception.	pred? : Function<T,Boolean>	T
argMax		
Determines the element with the maximum value determined by the map-function (using a natural order).	map : Function<T,Object>	T
argMin		
Determines the element with the minimum value determined by the map-function (using a natural order).	map : Function<T,Object>	T
argMedian		
Determines the element with the minimum value determined by the map-function (using a natural order).	map : Function<T,Object>	T

Table A.11.: **Sequence element** functions choosing a certain element of the sizrce sequence. In this table, all functions are applied on an object of type *Sequence<T>*, whereas *T* is an arbitrary *MxlType*.

Name and Description	**Parameters**	**Returns**
count		
Counts all elements of the source sequence (or counts the elements satisfying the predicate).	pred? : Function<T,Boolean>	Number
ratio		
Returns the ratio of elements fulfilling the predicate as a number between 0 and 1.	pred : Function<T,Boolean>	Number
sum		
Sums up all numbers determined by the optional map-function (default is the identity function).	map? : Function<T,Number>	Number
average		
Computes the average of all numbers determined by the optional map-function (default is the identity function).	map? : Function<T,Number>	Number
max		
Determines the maximal value determined by the optional map-function (using a natural order, default is the identity function).	map? : Function<T,U>	U
min		
Determines the minimal value determined by the optional map-function (using a natural order, default is the identity function).	map? : Function<T,U>	U
median		
Determines the median value determined by the optional map-function (using a natural order, default is the identity function).	map? : Function<T,U>	U
aggregate		
The func-function is invoked for the result of its previous invocation and each of the source sequence's elements. For the first iteration of the func-function, the seed value is used. The result of the last invocation of the func-function is the final result.	fun : Function<T, T, T> seed : T>	T

Table A.12.: **Aggregation** functions folding up all elements to a single value. In this table, all functions are applied on an object of type *Sequence<T>*, whereas *T* and *U* are arbitrary *MxlTypes*.