DEPARTMENT OF INFORMATICS

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Avoiding Redundancy in the Management of Technical Documentation and Models: Requirements Analysis and Prototypical Implementation for Enterprise Architecture Management

Peter Velten

DEPARTMENT OF INFORMATICS

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Avoiding Redundancy in the Management of Technical Documentation and Models: Requirements Analysis and Prototypical Implementation for Enterprise Architecture Management

Vermeidung von Redundanzen in der Pflege von technischen Dokumentationen und Datenmodellen: Anforderungserhebung und prototypische Implementierung für das Enterprise Architecture Management

Author:	Peter Velten
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Thomas Reschenhofer, M. Sc.
Submission Date:	September 15, 2016

Abstract

To make well-founded decisions, managers require consistent access to adequate and expedient information. Enterprise Architecture Management approaches this challenge by aligning information technology and business. However, as enterprises grow and evolve, frequently the problem arises, that information is utilized in various artifacts (e.g. documentations and reports), causing inconsistencies in content and form. Thus their content partially overlap and redundancies occur. This factor, combined with extensive information volume, rapidly changing requirements, and manually processing leads to 3 major problems in the documentation of the Enterprise Architecture. It is considered as time consuming, cost extensive, and error prone.

Addressing these problems, companies demand an applicable report-generating tool to prevent these redundancies, whereby consistent artifacts have to be created. Accordingly, specific requirements arise.

In this thesis practical relevant requirements for such a reporting tool are elaborated. Hereupon a concept is devised to accomplish and prototypically implement those requirements. As technical environment SocioCortex is in use. SocioCortex is an information system to organize semi-structured data within Enterprise Architecture Management, employing a dynamic and collaborative Wiki-based approach. Thus data is structured and presented as interconnected Wiki-pages. Additionally attributes and tasks may be denoted to incrementally enhance the data's structure. By applying this system as data-provider for a report-generator the confinement of properly collected data can be attenuated.

Via implementing this prototype the potentialities and capabilities of a reporting function within SocioCortex shall be demonstrated. A final review evaluates the implementation and its integration within the system. Moreover additional requirements, for the deployment in real enterprises, are discussed.

Contents

List of Abbreviations

API	Application Programming Interface
CSS	Cascading Style Sheets
EA	Enterprise Architecture
EAM	Enterprise Architecture Management
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IS	Information Systems
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MxL	Model-based expression Language
OMG	Object Management Group
PDF	Portable Document Format
REST	Representational State Transfer
SEBIS	Software Engineering for Business Information Systems
SQL	Structured Query Language
TUM	Technische Universität München
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

List of Figures

List of Tables

List of Listings

Thesis Outline

Chapter 1: Introduction

Chapter 1 motivates this thesis by emphasizing the importance of Enterprise Architecture Documentation in practice as well as in academia. In particular the focus is on the approaches of creating artifacts. Thereupon resultant problem areas and challenges in enterprises are pointed out. Subsequently the thesis objective is stated and the followed research paradigm is outlined.

Chapter 2: Foundations

Chapter 2 elucidates the domain background of this thesis. Thus Enterprise Architecture Management and its inherent tasks are defined. Furthermore the concept of Hybrid-Wikis is explicated and the instance *SocioCortex* is introduced.

Chapter 3: Requirements Analysis

For the purpose of constructing a report-generating application, functional requirements from real enterprises are collected and analyzed. In addition, non-functional requirements for internet and corporate applications are derived from literature.

Chapter 4: Modeling and Architecture

Prior to concrete implementation, models of the structure and behavior of the application are delineated. The notation and diagrams of the Unified Modeling Language (UML) is utilized for this purpose.

Chapter 5: Prototypical Implementation

Chapter 5 expounds the implementation of the application. At first the external resources, used to facilitate the implementation process, are explained. Subsequently the client- and server-side implementation is outlined by providing illustrations and code listings.

Chapter 6: Evaluation

The prototypical application is descriptively evaluated according to the design science paradigm. Thus feedback, to predefined use case scenarios, is solicited from representatives of enterprises to assess the functional requirements. The non-functional requirements are evaluated by reasoning with critical and factual arguments.

Chapter 7: Conclusion

In the conclusion the done work is recapitulated in a summary. Furthermore limitations of the application are identified. Hereto also the approach of this thesis is critically examined. Ultimately, prospective areas of the application and ideas for future developments are suggested.

1. Introduction

Not only high-tech firms, also brick-and-mortar enterprises use information technology platforms to support executive decisions to advance business performance (Weill and Ross 2009, pp. 1-9). Hereto up-to-date information in form of documentations and reports of the enterprise organization is imperative. Addressing this challenge section 1.1 motivates this thesis by expounding the importance of documentation in organizations in the context of Enterprise Architecture Management. Section 1.2 identifies the arising problems in practice by elaborating interview results and by examination of theoretical background. On the basis of these problems section 1.3 elucidates the objective of this thesis and outlines the followed research approach.

1.1. Motivation

Due to changing market requirements, an ongoing globalization, the emergence of IT to an enabling driver, and other environmental influences (e.g. technical or regulatory amendments), the complexity of IT-landscapes in organizations increases continuously (Buckl, Dierl, Matthes, and Schweda 2010, pp. 1-2). This fact bears major challenges especially for medium to large enterprises. EAM approaches this challenges by aligning information technology and business to gain a strategic advantage by realizing cost savings, enhancing availability, and increasing failure tolerance (Roth, Hauder, Farwick, et al. 2013, pp. 1-4) (Buckl, Matthes, Neubert, and Schweda 2009, pp. 1-2). The importance of IT-business alignment is assessed by studies and research (Chan 2002) (Chan, Huff, Barclay, and Copeland 1997) and by surveying IT-executives, who ranked IT-business alignment as top issue in their organization (Luftman, Kempaiah, and Nash 2006).
The rising interest and application of EAM cannot only be observed in practice, but also in academia. In the paper "Enterprise Architecture: What Aspects is Current Research Targeting" Langenberg and Wegmann analyzed research activities in EAM by evaluating papers referencing the term "Enterprise Architecture". They concluded, due to the number of publications, a rising interest in research. (Langenberg and Wegmann 2004)
A significant part of EAM is the documentation of the Enterprise Architecture. While this includes information for maintaining and development, a documentation consists of current information as well as information about future states of the architecture.

An Enterprise Architecture is comprised of infrastructure components, business applications, business processes, and their relationships to each other. The scope of documenting these artifacts can be illustrated by considering the pure amount of business applications utilized in enterprises. From experience, as well as referencing the papers of Sabine Buckl or Sascha Roth, several thousand applications are in use in large enterprises. Additionally they are not operating independently; instead these applications are highly interconnected. Furthermore the long lifespan of applications amplifies the need of proper documentations, since descriptions of functionality, architecture, and connections to other applications should be available over the long term. (Roth, Hauder, Farwick, et al. 2013, pp. 1-4) (Buckl, Dierl, Matthes, Ramacher, et al. 2008, pp. 1-3)
Further practical evidence for the application of Enterprise Architecture Documentation can be derived from the results of a survey, provided by a cooperation of the University of Innsbruck and the Technische Universität München. In this survey 179 participants answered a questionnaire about key problems in EA Documentation. An overview of the usage of EA Documentation, based on this survey, is illustrated in Figure 1.1. (Farwick, Breu, Hauder, et al. 2013)

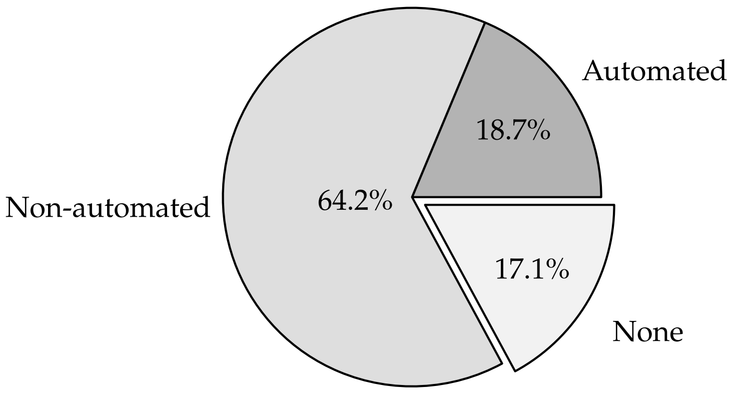


Figure 1.1.: Application of Enterprise Architecture Documentation in practice (Farwick, Breu, Hauder, et al. 2013)

As the pie-chart reveals, 18.7% of the participants already implemented automated EA Documentation mechanisms, 64.2% using EA Documentation, albeit not automated, and only 17.1% do not know or do not use any EA tools at all (Farwick, Breu, Hauder, et al. 2013, p. 4). Totaled up, 82.9% of the enterprises uses documentation techniques and tools, evincing that these are subjects in enterprises to a vast extent. The applied tools can cover various areas, ranging from the collection of data, via the creation of artifacts (e.g. documentations and reports), through to comprehensive EAM tools.

An overview and categorization of commonly used tools can be examined in the "Enterprise Architecture Management Tool Survey 2008" provided by Matthes et al. (Matthes, Buckl, Leitel, and Schweda 2008).
Alongside practical usage, also in theory Enterprise Architecture Documentation is considered as crucial for Enterprise Architecture Management. While there is no consensus over a concrete proceeding in EAM, several books and papers coincide to the significance of the usage of EA Documentation. Subsequently an overview of literature, covering the usage of EA Documentation, will be provided. The selection entails the book of Scott A. Bernard, the EA Cycle of Niemann, and the IT planing process of Hanger. For a further summary of relevant EAM literature the written report "On the State-of-the-Art in Enterprise Architecture Management in Literature" of Sabine Buckl and Christina M. Schweda is suggested (Buckl and Schweda 2011).
Scott A. Bernard describes in an implementation methodology how to implement and document an Enterprise Architecture, wherein he declares documentation as one of the keys to success. The implementation methodology is splitted up in 20 steps, clustered into four phases, as Figure 1.2 reveals (Bernard 2012, pp. 85-99).

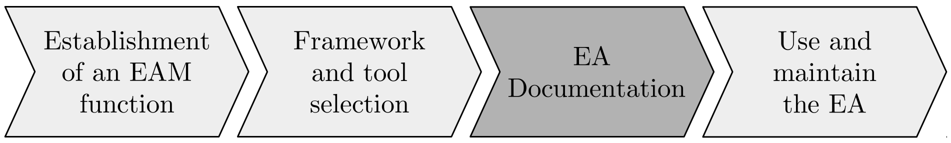

Establishment of an EAM function
Framework and tool selection
EA Documentation
Use and maintain the EA

Figure 1.2.: Phases of an Enterprise Architecture implementation methodology for the creation of an EA program according to Scott A. Bernard (Bernard 2012, pp. 85-99)

While this section motivates the application of EA documentation, phase three is highlighted. At this stage the current as well as future scenarios should be analyzed. This includes strategy, business, information, services, and infrastructure. In this phase artifacts are generated and organized. They include reports, documentations, overviews, analysis, and studies. For this purpose, several specialty tools may be required. Examples are general document-, spreadsheet-, and graphics applications. Bernard suggests to store the created artifacts in an appropriate EA repository. It should consist of a database and a file directory to archive all gathered information. As the other phases are out of the scope of this thesis, an extensive explanation is outlined in his book "An Introduction to Enterprise Architecture". While Bernard only abstractly describes the generation and storage of artifacts, concrete implementation recommendations is left up to the reader. (Bernard 2012, pp. 85-99)
However, it becomes clear, that supporting tools are essential, as the purely manual execution of these tasks leads to severe problems. Still many companies (see Figure 1.1) are not using appropriate tools.

Klaus D. Niemann describes the preceding in EA Management as a cycle of four steps with a central controlling unit. Thereby he concentrates on the usage of EAM in the context of an IT governance program. In his book "From Enterprise Architecture to IT Governance" Niemann explains every of the steps in detail in chapter 4 to 7. Figure 1.3 depicts the phases, while highlighting the essential part for this thesis. (Niemann 2006, p. 37)

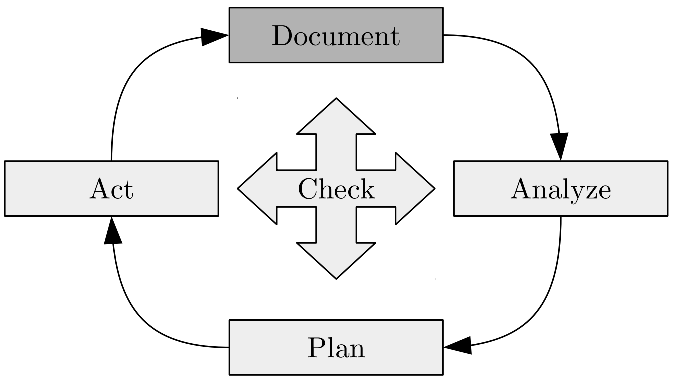


Figure 1.3.: EA-Cycle to develop an Enterprise Architecture in the context of an IT governance program based on Klaus D. Niemann (Niemann 2006, p. 37)

The documentation is the first step and constitutes how an Enterprise Architecture should be structured and documented. In this process the IT assets and relationships are documented in the required format. Thus plans, references, analysis, and documents are provided to organize the IT. Subsequently the analyze step uses and evaluates these artifacts. In the planning step target scenarios are developed, whereupon an implementation of scenarios take place in the act step. The central check unit controls and secures the process. Niemann concentrates in the documentation step on the different layers of an Enterprise Architecture, namely business architecture, software architecture, and systems architecture. Also he declares to document the current and future state, whereby the future state is not developed in the document phase, but rather in the planning phase. Niemann is more concrete, considering the implementation of such a program, as he provides application scenarios. While these scenarios provide different viewpoints, concrete implementations are absent. As with Bernard, Niemann also suggests the creation of documents and other artifacts, whereby the format and the creation-procedure remains unclear. (Niemann 2006, pp. 28,37,51,75-122) (Buckl and Schweda 2011, pp. 99-101)

Inge Hanschke provides a practical toolkit in her book "Strategic IT Management - A Toolkit for Enterprise Architecture Management". In this book she tries to close the gap between literature approaches and practice by providing practice-proven prescriptions, guidelines, and best practices in the context of EA Management. Following this practical approach, she amplifies her preceding by presenting the EA Management tool "iteraplan" (iteratec GmbH 2016).
She portrays methods for the management subjective "Business Landscape", "IT Landscape", and "Technology Landscape", whereby strong focus lays on the "IT Landscape". IT Landscape management links business and IT-structure and creates a transparent overview of the current and future status. While this is more abstract, she defines the task of Enterprise Architecture as the procedure of *"...pulling together disparate information from business and IT and create associations between elements such as business processes (from the business) and applications (from IT)."* (Hanschke 2009, p. 114). For this subjective she differentiates between four processes, which are delighted in Figure 1.4. (Hanschke 2009, pp. 138-139) (Buckl and Schweda 2011, p. 108)

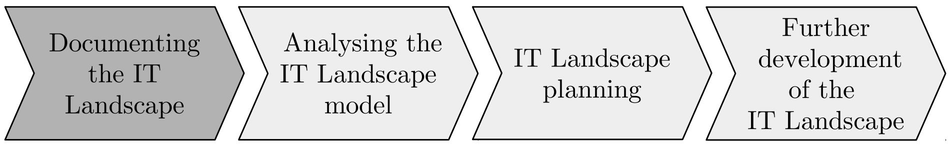

Documenting the IT Landscape
Analysing the IT Landscape model
IT Landscape planning
Further development of the IT Landscape

Figure 1.4.: Four processes of IT-Landscape Management in the context of Enterprise Architecture Management according to Hanschke (Hanschke 2009, pp. 138-139)

In the highlighted process "Documenting the IT Landscape" an overview of the applications and their relationships are provided. A relationship can exist between the applications themselves or between the the application and the business landscape respectively the infrastructure landscape. Hereby she describes a life-cycle of the documentation by first inventorying what exists and then try to maintain this information permanently. This is the so called "maintenance concept". Hanschke does not go into detail, while describing concrete formats or creation techniques for artifacts. However, she identified the problem, that data, collected from various data sources, is often not up-to-date or incomplete. (Hanschke 2009, pp. 138-142)

As they are different viewpoints and procedures in EAM, exact characterizations and definitions applicable for this thesis will be provided in chapter 2.
All these approaches in different contexts of EAM have the major challenge of documentation in common, including the issue of creating appropriate artifacts. The artifacts, including documentations and reports, are frequently created manually, whereby failures are provoked. Also the topic of changing data, resulting in outdated information,

was thematised. As the degree of documentation, alone by regarding the amount of application specific documentations, is enormous, a lot of information is utilized redundantly in different artifacts. By this redundancy, intertwined with manually proceeding, the occurring of failures is inevitable. Also the workload is tremendous if an information changes and all artifacts have to be adapted. Although these problems were recognized in literature and practice, solutions often lack of concrete approaches or implementations. Only Hanschke provided a properly EAM tool, albeit the question remains how concrete documents for documentation-reasons are created.

Summarizing, there is a strong need for documentation in the context of EAM. This need is recognize both in practice, e.g. in firms and enterprises, and in academia. In firms new division and departments have been established. Also tools for supporting this task with automatic mechanisms were introduced, even when the amount of enterprises, utilizing these tools, is outnumbered. In academia a rising amount of published paper in this topic can be observed. Although many recommendations are provided of handling this topic, different problems occur by applying proper EA Documentation in enterprises. Especially the creation of documents and other artifacts is problematic. Oftentimes the reason is due to abstract proposals, whereby concrete implementations are missing, which are oriented on concrete requirements from practice. To grasp the topic, the next section will formalize and elucidate these problems in more detail.

1.2. Problem Statement

As already mentioned, large enterprises have several thousand of applications running. The properly documentation of these applications and their relationships, but also of the organizational structures and processes is a crucial part of Enterprise Architecture Management.
Often this documentation procedure, which includes the creation of technical documentations, reports, and other artifacts, is done manually. Thus information from different sources are combined and final reports are generated. Many a time an information is used in several artifacts simultaneously. This fact, combined with extensive information volume and rapidly changing requirements, leads to 3 major problems in Enterprise Architecture Documentation. It is considered as time consuming, cost extensive, and error prone. (Hauder, Matthes, and Roth 2012, pp. 1-2)
These problems are not only occurring in the creation of documentations, also the gathering and collection of data is identified in academia and practice. Although this is a major topic, this is not covered in this thesis. There are different approaches available. For example Farwick et al. proposes tools to integrate information out of a cloud infrastructure into an EA view automatically. Additionally he provides a prototypical implementation for this approach (M. Farwick, B. Agreiter, R. Breu, et al. 2010). Buschle and Holm follow a similar approach by illustrating how to utilize a vulnerability scanner to create EA models automatically (Buschle, Holm, Sommestad, et al. 2012).
In this thesis a system with a proper data basis is taken for granted. By choosing a Hybrid-Wiki platform, as described in section 2.2, this restrictive presumption can be mitigated.
More concrete reasons for the aforementioned problems can be explicated by determining the operational proceeding in enterprises. First, often spreadsheets and text (e.g. Microsoft Office) is used by companies for documentation (Buckl, Matthes, Neubert, and Schweda 2009, p. 4). From experience some companies also use presentations as documentation format. While these tools have a good capability of structuring and styling documents, in combination with the complexity of Enterprise Architecture they have the problem of not scaling properly. Furthermore it is difficult to maintain these documents consistent, also because to the amount of frequent changes. (Buckl, Matthes, Neubert, and Schweda 2009, p. 4) (Farwick, Breu, Hauder, et al. 2013, p. 1)
By conducting interviews with several Enterprise Architects, these problem could be concretized. Thus different problem areas were reported, in which they struggle to produce consistent artifacts.

- As abstractly mentioned, the participants reported the need for properly Microsoft Word and Microsoft PowerPoint documentations. Up till now they are using many templates, while there is a lot of freedom to edit these documents. Therefore different layouts emerged. Furthermore no clear repository for these documents is available. As consequence these documents are distributed over the company and therefore it is challenging to find the right information in a short space of time.
- The problem of reusing text-fragments in different documents. This redundancy causes huge time effort in the case of changing source data, since all artifacts have to be changed properly. Of course in doing so, errors are provoked.
- Related to the aforementioned problem, often information in these documents are copied from information systems (EAM tools). While these information should be kept synchronized, there is no present way to link the text documents with the fragments in the EA tools.
- Current solutions of exporting information of EA-tools lacking in the power of office tools. Formatting therefore becomes very difficult. The resulting documents were contorted and a lot of work has to be done afterward to officially publish them, as often this documents have to be printed out and archived.
- Since the documents were distributed and no central EA repository exist it is not possible to create reports about the meta information. As a consequence it is impossible to state how many documents exist or which ones are missing. Another use case might be the identification of application without proper documentation.

In total it can be stated, that the creation of artifacts within EA Documentation has many problem areas involved. To narrow the scope of this thesis, only defined problems can be addressed. It gets clear from the concrete problems, that the application of powerful text-formatting tools are desired. Also because of the knowledge of the employees to handle these tools. However, the purely application of these tools is not expedient, as information cannot be linked with data sources and inconsistencies occur.
One citation of an enterprise architect states the demand of enterprises in one sentence:

> *"There is no parent solution how to create, edit, and mange technical documentations in a Hybrid-Wiki based system, with the functionality of standard desktop-based tools."* *(Enterprise Architect)*

1.3. Objectives and Approach

There are only few approaches considering concrete documentation, including the generation of artifacts of the Enterprise Architecture (Grunow, Matthes, and Roth 2013, p. 2). In present literature authors mostly follow a more principle approach, while they leave concrete instances of the problem open for the reader. Resulting in an error-prone, time-consuming, cost-extensive approach to EA Documentation.
While there is a notable demand of tool support in EA Documentation, different tasks has to be accomplished. Beginning by the collection of data and information and ending with the creation of artifacts (e.g. documentations and reports). This thesis focuses on the creation of artifacts. Therefore an application should be implemented, to create documentations and reports in a suitable manner to avoid one of the causes for these problems: The redundancy of information used in different documents.
For this purpose concrete requirements in literature are nonexistent. Therefore requirements from real enterprises should be collected and analyzed. With the aid of modeling languages a suitable solution, addressing this requirements, should be produced. The generated solution should be transferred to a prototypical implementation. While there is the observation for a growing interest for web 2.0 tools in enterprises, a modern web application is targeted (Büchner, Matthes, and Neubert 2009, p. 1). Ultimately an evaluation assesses the functionality of the application.

This approach follows the design science paradigm of Hevner et al. It seeks to extend the boundaries of human and organizational capabilities by creating new and innovation artifacts. By providing intellectual and computational tools, this practical approach is considered as a problem solving paradigm. This artifacts should address fundamental problems of the productive application of IT. (Hevner, March, Park, and Ram 2004)
Figure 1.5 summarizes the approach in a comprehensive exposition.
The problem can be derived from the environment, by identifying business needs. It consists of people, business organizations, and their used technology. In this thesis the needs are gained by raising concrete requirements from real enterprises. Based on these business needs the actual information systems research take place, subdivided into two phases: The build phase and the evaluation phase. The build phase has to goal of creating utility. In this thesis this is accomplished by building a functioning prototype. Finally the tool will be evaluated to assess the utility, quality and effectiveness. While Hevner proposes different methods of evaluation, in this thesis the evaluation is done via a descriptive analysis. The knowledge base provides the foundations and methodologies from academia. From this base suitable modeling-instruments are utilized. Furthermore non-functional requirements are derived from literature. By

passing through this process, additional knowledge is generated, which can be added to the knowledge base. Therefore the gathered requirements or the evaluation results might be helpful for other researchers. (Hevner, March, Park, and Ram 2004, pp. 79-81)

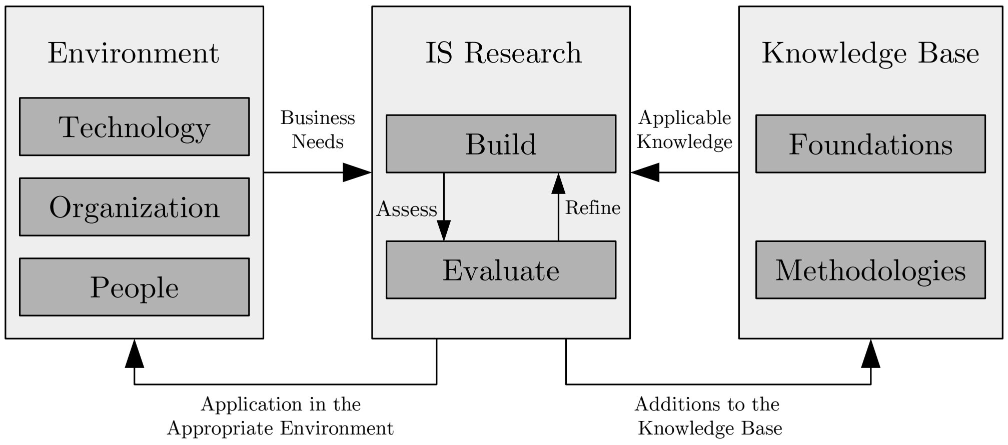


Figure 1.5.: Design science paradigm in information systems research to create utility by building and evaluating artifacts according to Hevner (Hevner, March, Park, and Ram 2004, p. 80)

2. Foundations

Enterprise Architecture Management substantiates this thesis by providing action fields to keep enterprises competitively viable. However, in academia no commonly accepted definition exist. Nevertheless this chapter defines EAM, applied throughout this thesis, by clarifying its methodology, procedure, and goals. As mentioned in chapter 1 the focus in this thesis is on the creation of documentations and reports, whereas the data collection and storage is not being targeted. To water down this confinement, the concept of Hybrid-Wikis will be outlined, while introducing an instance, namely "SocioCortex" as data provider for the application. In this approach, content is generated dynamically by a collaborative user base. The SocioCortex system will play a crucial role in the course of this thesis, as it not only provides the information for generating reports, but it turns out, that it is also a perfect environment to store created artifacts, and is therefore also used as EA repository, as suggested by Bernard.

2.1. Enterprise Architecture Management

Enterprise Architecture Management covers business- as well as IT aspects. Even no generally valid definition exist, there is a consensus, that EAM supplies a holistic view of the enterprise's organization, covering stakeholders, locations, and business processes. The goal is to provide relevant information to support a decision-making process. (Roth, Hauder, Farwick, et al. 2013, pp. 1-2).
Enterprise Architecture is defined by the International Organization for Standardization in no. 42010 as follows ("ISO/IEC/IEEE Draft Standard for Systems and Software Engineering – Architectural Description" 2010):

> *"Enterprise architecture (EA) is the fundamental conception of the enterprise in its environment, embodied in its elements, their relationships to each other and to its environment, and the principles guiding its design and evolution."*

By taking an perspective of an overall Enterprise Architecture, the management addresses the alignment of IT and business through a common vision, alignment, and standardization. Hereby the business drives EAM, whereby IT enables business strategies and goals. (Buckl, Dierl, Matthes, and Schweda 2010) (Matthes, Buckl, Leitel, and Schweda 2008, p. 24)

As the name implies, EAM is a management discipline. Therefore the typical management process, depicted in Figure 2.1, can be applied. More information about this cycle, also known as *Deming Cycle* can be found on the website of the institute of W. Edwards Deming (The W. Edwards Deming Institute 2016).

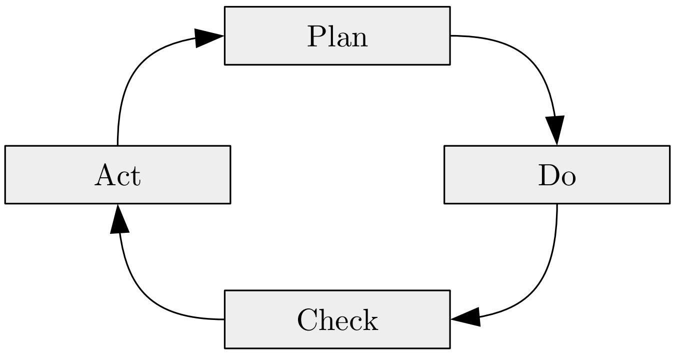


Figure 2.1.: Management process according to W. Edwards Deming (The W. Edwards Deming Institute 2016)

In the context of EAM each part can be described as follows. (Buckl, Matthes, Neubert, and Schweda 2009, p. 2) (The W. Edwards Deming Institute 2016) (Diefenthaler and Bauer 2014, pp. 1-2)

- ***Plan***
  In the *planning* phase, the current (as-is) and the future state of the enterprise architecture will be drafted. This includes the development of future scenarios to support decision making. After formulating these theories the start of the plan will be prepared.

- ***Do***
  In the *do* phase selected scenarios will be realized through programs, projects, and initiatives.

- ***Check***
  In the *check* phase outcomes are analyzed and evaluated. So the accrued Enterprise Architecture will be compared to the planed. Success and progress will be measured, but also emerging problems will be recognized. For measurement a KPI-based approach is recommended.

- ***Act***
  In the *act* phase potential process improvements will be identified based on the results from the *check* phase. Goals and plans can be adjusted, whereupon a new cycle can be prepared.

They main tasks of EAM, to support and execute the management phases, are documentation, communication, and analysis.

- **Documentation**
  The documentation task supports the plan phase in the management cycle, as it documents the current state of the EA. This presupposes the development of a suitable information model for the EA. Besides the current situation, also a future state should be derived from a long-term vision of the enterprise. (Buckl, Matthes, Neubert, and Schweda 2009, p. 2)
  Hereby different elements on different layers and cross-functions should be considered to get a holistic view on the architecture. While all layers are important for executing EA Management, the implemented degree of detail might differ between enterprises. Also the degree of abstraction might vary. The layers and cross-functions are depicted in Figure 2.2 (Matthes, Buckl, Leitel, and Schweda 2008, p. 29). Each layer includes different applied concepts. For a more detailed description of each layer and cross-function the written report of Matthes et al. is suggested (Matthes, Buckl, Leitel, and Schweda 2008).

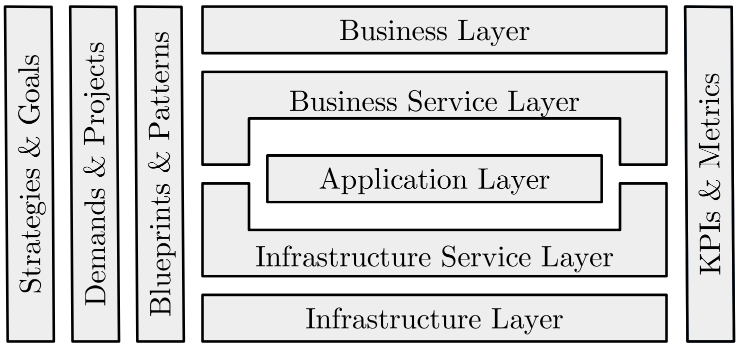


Figure 2.2.: Layers and cross-functions for a holistic view on the organization's architecture (Matthes, Buckl, Leitel, and Schweda 2008)

- **Communication**
  To support the planning phase, various stakeholders are involved, hence the required data is collected from different layers (Figure 2.2). This might be, for instance, Project Managers, Service Managers, or Enterprise Architects. The existing terminology gap between business and IT can impede the cooperation and communication within the EA Management process. To close this gap visualizations are often used. McDavid proposes an architectural approach by providing different levels of abstraction from business to IT (McDavid 2003). (Buckl, Matthes, Neubert, and Schweda 2009, pp. 1-2)

- **Analysis**
  Finally, the check phase of the current and future architecture has to be analyzed and evaluated. This supports the last phase of the management cycle. The results of the analysis task are provided to the act phase to improve decision making and to prepare the advancement by starting a new iteration. (Buckl, Matthes, Neubert, and Schweda 2009, p. 2)

Summarizing, EAM plans and manages the evolution of the Enterprise Architecture. Hereto classical management approaches can be applied. The different tasks support and implement those management processes. To state the overall goal of EAM Matthes et al. can be referenced (Matthes, Buckl, Leitel, and Schweda 2008, p. 24):

> *"The goal is a common vision regarding the status quo of business and IT as well as of opportunities and problems arising from theses fields, used as a basis for a continually aligned steering of IT and business."*

2.2. Hybrid-Wikis and SocioCortex

Wiki-platforms emerged from the evolution to the new web 2.0 paradigm. They follow the principle of harnessing collective intelligence (O'reilly 2007, pp. 22-27). Basically Wikis are content management systems for creating and editing content. (S. Murugesan 2007, pp. 2-3) Wikipedia, with more then 5.2 million articles, only in the English version, is probably the most famous instance of a Wiki available on the internet (Wikimedia Foundation 2016).
As collaborative information systems find growing interest in enterprises, the question arises, how to align the data with the corresponding model. The top-down approach migrates the data when the related model changes or evolves (model-first). In the bottom-up approach first data is generated, whereby the model aligns to that data (data-first). More details on this approaches can looked up in Reschenhofer et al. (Reschenhofer, Bhat, Hernandez-Mendez, and Matthes 2016, pp. 2-3).
A more collaborative approach is suggested in "Hybrid-Wikis: Empowering Users to Collaboratively Structure Information" by Matthes et al. This "Hybrid-Wiki" approach supports the evolution of the model and its data in a coherent and consistent manner (Matthes, Neubert, and Steinhoff 2011). (Reschenhofer, Bhat, Hernandez-Mendez, and Matthes 2016, pp. 1-2).
The goal of such a Hybrid-Wiki approach is to lower the barriers for non-expert users. Without knowing about the background, they should be able to enter structured data. To solve this problem Matthes et al. proposes to enrich these contents later with simple key-word-like annotations by experienced users or experts. While the notion of semantic annotations should be avoided, the user implicitly creates semantic, by filling data into fields, or create new fields in forms. (Matthes, Neubert, and Steinhoff 2011, pp. 1-2).
The corresponding data model of a Hybrid-Wiki approach is depicted in Figure 2.3. The concept is developed by the chair of Software Engineering for Business Information Systems (SEBIS) of the Technische Universität München. It is based on the modeling framework of a former tool called Tricia. (Matthes, Neubert, and Steinhoff 2011, p. 6)
Each wikipage can contain multiple attributes, while each attribute can have a list of values. A *Value* is an abstract type, concrete *Values* might be *StringValues* or *LinkValues*. Also a wikipage can have multiples tags assigned to it. On the lefthand side of the diagram, the *Wikipage*, *TypeTag*, *Attribute* and *Value* are provided for structuring the data in Hybrid-Wikis. On the righhand side *TypeTagDefinition*, *AttributeDefinition*, and *Validator* are concepts to specify integrity constraints. So the creation of a *TypeTagDefinition* specifies the values an user is urged to enter. While *TypeTagDefinition* and *TypeTag*, and *Attribute* and *AttributeDefintion* are loosely coupled, users only receive suggestions for new attributes or tags. Thus new attributes can be created top-down. Integrity

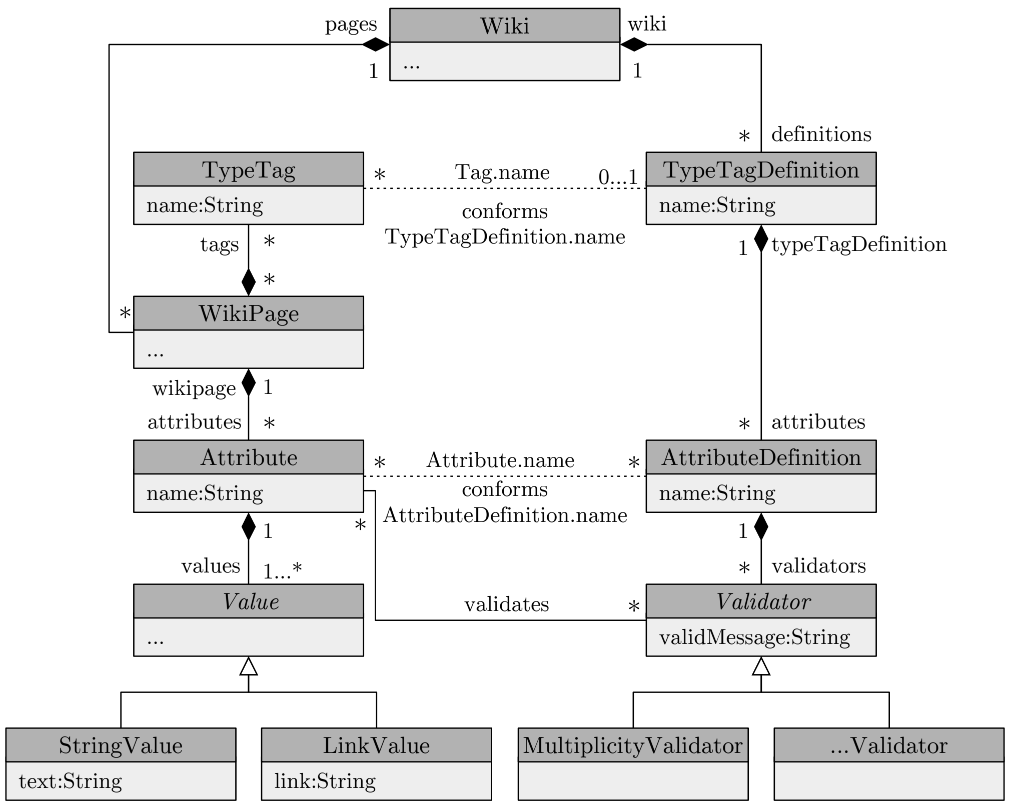


Figure 2.3.: Data model of a Hybrid-Wiki approach according to Matthes et al. (Matthes, Neubert, and Steinhoff 2011, p. 7)

constraints defined in *AttributeDefinition* can be specified using *Validators*. For example, a *MultiplicityValidator* specifies how many values an attribute can have (at-least-on, at-most-one, exactly-one, etc.). *Validators* are apparent by showing feedback messages for violated integrity constraints. They are called "soft validators", while they only warn users, but do not forbid the user to break them. (Matthes, Neubert, and Steinhoff 2011, pp. 6-7)
This Hybrid-Wiki approach was applied in the course of the Wiki4EAM community, by research of TUM with collaboration of 25 German enterprises. The outcomes of these projects, using this approach to document their Enterprise Architecture, showed positive aspects. It demonstrated, that stakeholders were empowered to collaboratively reveal their information demand. Furthermore the EA model emerged bottom-up in short time, while the content was created in early stages of EA initiatives. Further positive aspects as well as challenges are described in (Reschenhofer, Bhat, Hernandez-Mendez, and Matthes 2016, pp. 3-4).

As the Hybrid-Wiki approach is an extension of classical Wikis, the terminology of *Wiki* and *Pages* was applied. These terms already referred to kind of representation and content creation. As the application in the context of EAM revealed, Hybrid-Wikis are not only intended for traditional knowledge management, but also for capturing architecture elements. Examples might be processes, projects, and tasks. Therefore a new terminology evolved, while the semantics remained the same. Now *Wikipage* is called *Entity* and a *Wiki* is labeled as *Workspace*. The updated data model in the new context is depicted in Figure 2.4. While trying to reduce the complexity of the model and to keep the system intuitive and self-explanatory, the coupling between entities and entity types is simplified. An entity now can only be assigned to one type. So by removing the concept of *TypeTag*, *Entity* and *EntityType* are now directly connected to each other. The concepts of workspaces, entities, and entity types are now used throughout this thesis. (Reschenhofer, Bhat, Hernandez-Mendez, and Matthes 2016, pp. 5-6).

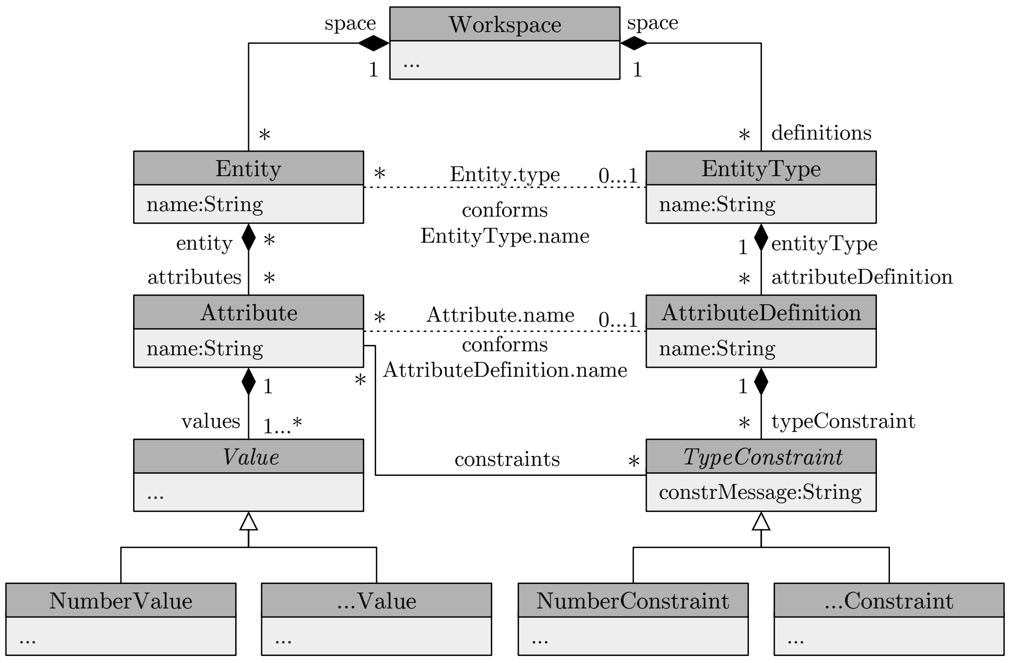


Figure 2.4.: Hybrid-Wiki data model in the context of Enterprise Architecture Management according to Reschenhofer et al. (Reschenhofer, Bhat, Hernandez-Mendez, and Matthes 2016, p. 5)

While integrating proven features of this Hybrid-Wiki approach with concepts of end-user-oriented quantitative model analysis, the support for knowledge-intensive processes, and by adding a standardized REST-API to access all of these features, a

new platform, named SocioCortex was developed. It provides the foundation for the development of diverse applications. A more detailed description and an overview of the architecture can be followed up at the website of the SEBIS-chair, which developed this platform (*SocioCortex - A Social Information Hub* 2016).
SocioCortex provides an interface to use its features. Furthermore the platform provides with MxL a powerful query language to access the data in the system. By practical tests, in the course of WIKI4EAM, positive aspects could be observed. Therefore the platform qualifies excellently for the usage as data provider and repository for a report generating application in the context of Enterprise Architecture Documentation. Herewith the confinement of a proper data basis is decomposed.

3. Requirements Analysis

Many reporting and documentation tools have their existing repository of information (Buckl, Dierl, Matthes, Ramacher, et al. 2008). This can be assumed to be present as section 2.2 outlined, by applying SocioCortex as technical environment. Therefore the analysis of requirements concentrates on the generation of artifacts.
From literature no concrete functional requirements could be derived, hence enterprises had to be asked for their requirements. This follows the design science paradigm of Hevner et al. by raising business cases from the environment. Based on these requirements a tool can be constructed to tackle these.
Four enterprises from different sectors showed interest in the development of a report generating tool in the context of EAM. By complying with confidentiality, no concrete names can be stated. However, Table 3.1 lists the sectors and amount of employees of the participating enterprises.

Sector	**Employees**
Logistics	95 000
Consulting	93 000
Finance	19 200
Healthcare	1 600

Table 3.1.: Sector and amount of employees of participating companies

With representatives of these enterprises (primary Enterprise Architects) personal interviews and discussions were conducted. From these talks, the problems and requirements for a report generating tool could be derived. The problem statements were already mentioned in section 1.2. In this chapter concrete requirements are analyzed and finally prioritized.
Mainly functional requirements could be derived from these interviews. They are listed and explained in section 3.1. By developing an application, also non-functional requirements should be considered. These are derived by literature research. As the tool should be a web application, only non-functional requirements for internet and corporate applications are considered. They are declared in section 3.2.

3.1. Functional Requirements

As the enterprises have different ideas and expectations, accordingly diverse requirements were stated. Therefore first an inventory of all requirements was created. In total 24 requirements could be collected. These are listed in Appendix A.
While it is neither expedient nor feasible to create customized solutions for all specific enterprises, the requirements were abstracted and consolidated. The result are 9 general requirements. By this additional step a more general solution can be realized. Thus it is not only applicable in a specific enterprise, but rather can be deployed in different environments with similar problem areas. For specific business cases, potentially small adaption of the application have to be made. The process is delineated in Figure 3.1.

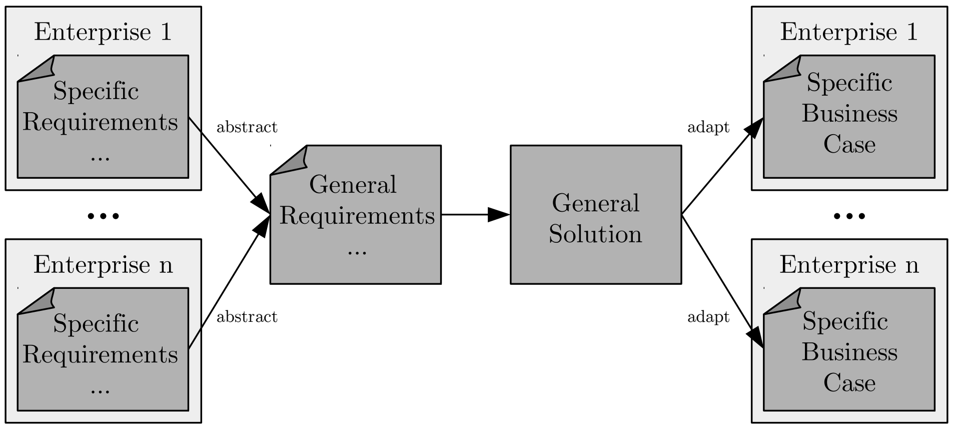


Figure 3.1.: Abstracting process to identify general requirements out of specific requirements

Following requirements could be derived from the list of specific requirements, which are numbered from R-1 to R-24.

- ***Integration of visualizations***
  This general requirement relates to R-5, R-17, R-23. Different kinds of visualizations should be integrated into generated documents. Thus standard picture formats should be supported, but also dynamic changing diagrams of entities in SocioCortex. The application should recognize the appearing of visualizations in the document and properly scale the forms with respect to design and legibility. While this requirement is important for organizations, other research projects are yet involved in this topic. For example, the SC-Visualizer, as part of the client suite of SocioCortex, empowers the user to visualize data from the system (Software

Engineering for Business Information Systems 2016d). Therefore this requirement has a rather low priority in the scope of this thesis.

- ***Document versioning mechanism***
The goal of this requirement, which relates to R-7, R-15, R-20, R-24, is to provide a version history, to clearly state the changes of documents. Furthermore it should be possible to determine the editor of the document. Every version of a document should be able to recover.
While this thesis strives a different approach, changes of documents should be avoided at all. By linking the content in the documents with up-to-date data, changes will be obsolete. However, storing the created documents in a repository makes sense to access the documents, filled with the outdated data. There might be, for example, regulatory or compliance reasons for this archiving. While the requirement of a concrete versioning mechanism has a low priority, the implementation of handling changes of the content has a very high priority.

- ***Publishing workflow***
This requirement relates to R-6. Only one company stipulated such a feature. While the requirement collection is not representative, this might be interesting for other companies, too. Thus a document should only by published, when responsible or accountable persons approve the content and form. As this requirement can be listed as extra feature for the application its priority is low.

- ***Reuse of sub-components***
The reuse of sub-components is related to the specific requirements R-1, R-2, R-3, R-12. As already mentioned, reports and especially technical documentations oftentimes include the same components (e.g. text or figures). Up to now these components are manually copied out from information sources, like SocioCortex, and processed in desktop-based tools. The occurring problems of this procedure and the consequential redundancy were stated in the problem statement (section 1.2). The approach of this theses, should link the sub-components with the data from SocioCortex. Thus every document relate to the same data source. This crucial requirement therefore has high priority.

- ***Interface to present data sources***
This requirement relates to R-1, R-3, R-19. While the enterprises require an interface to present data sources, this thesis mainly focus on the interaction with SocioCortex. For specific requirements the resulting prototype has to be customized to access other data sources as well. As the connection to SocioCortex is crucial for retrieving data and storing artifacts, this requirement has high priority.

- ***Integration of different file formats***
  The export to different file formats and the usage of different formats relates to the requirements R-4, R-16, R-18. Thus especially the possibility of exporting to non-editable documents should be possible. As enterprises also use presentation tools for documentation reasons, also presentation-formats should be integrated. This requirement has a medium priority, while the focus lies on the procedure of avoiding redundancy and further development might support specific formats required by the various enterprises.

- ***Usage of parameters and types***
  This requirements relates to R-8, R-9, R-10, R-21. Sub-components of documents could be of various forms. To address these components, different types of parameters should be applied. Thus simple parameters (e.g. strings, numbers, dates, etc.) as well as complex (e.g. lists) should be supported. This requirement has high priority, because it is inevitable to link components with data, without addressing them in a proper way.

- ***Serial generation of documents***
  This requirement did arise within an intermediate presentation in the course of the SocioCortex Community Workshop series (Software Engineering for Business Information Systems 2016a). As participants struggled with the pure amount of generating technical documentations, the demand of an automatically generation of a series of documentation was formulated. Thus it relates to the single requirement R-22. Hence the implementation was already in progress, this requirement is prioritized as a beneficial extra feature.

- ***Integration of formatting functions***
  This requirement corresponds to R-11, R-13, R-14. While the importance of properly content is undisputed, the enterprises often struggled with the proper formatting of documents. Current solutions, for example, support the export of entities to documents. However, the design and formatting of these documents were evaluated as being "poor". Therefore formatting functions of desktop-based tools (e.g. Microsoft Office) are an important issue. At the first glance this might be an extra feature to consider. But a report generating tool will not be accepted and utilized, when the resulting documents are not in the desired quality. Consequently this requirement has high priority.

An implementation of all requirements would go beyond the scope of this thesis. Therefore the requirements has been prioritized into 3 categories: High, medium, and low. Explanation for the priority is alluded in the above description of the requirements. Table 3.2 summarizes the requirements and shows the corresponding priority class.

Functional Requirement	**Priority**
Reuse of sub-components	High
Usage of parameters and types	High
Interface to present data sources	High
Integration of formatting functions	High
Serial generation of documents	Medium
Integration of different file formats	Medium
Publishing workflow	Low
Document versioning mechanism	Low
Integration of visualizations	Low

Table 3.2.: List and priority of general functional requirements

These functional requirements could be collected from enterprises. However, non-functional requirements also have to be considered. Next section will derive these from literature.

3.2. Non-functional Requirements

The non-functional requirement, in contrary to functional requirements, do not characterize what an application should do, but rather how well it operates. Examples are how fast a program finishes a task, or how fault-tolerant it behaves. These requirements can be distinguished in external and internal factors, whereas external factors are more important to the users, internal factors are used for development and maintenance reasons. While certain requirements are more important then others, depended on the problem, Wiegers declares following external non-functional requirements for internet and corporate applications. (Wiegers and Beatty 2013, Chap. 14)

External requirement	Description
Availability	The extent to which the system's services are available when and where they are needed
Integrity	The extent to which the system protects against data inaccuracy and loss
Interoperability	How easily the system can interconnect and exchange data with other systems or components
Performance	How quickly and predictably the system responds to user inputs or other events
Security	How well the system protects against unauthorized access to the application and its data
Usability	How easy it is for people to learn, remember, and use the system

Table 3.3.: External non-functional requirements for internet and corporate applications following Wiegers (Wiegers and Beatty 2013, Chap. 14)

A comprehensive description of the requirements can be found in the book "Software requirements" of Wiegers (Wiegers and Beatty 2013). Next the relevant requirements are briefly described and their impact on the development of the application is outlined.

Availability is a big issue when running the application in productive environment. For a prototypical implementation this plays only a minor role. Availability can be measured by the ratio of up-time to the sum of up and down-time. This measure is only applicable when running real servers. Of course this requirement should be quantified, through this measure, when running on a corporate server. Nevertheless the application should be implemented with failure-tolerant algorithms and functions in order to prevent crashes.

Integrity prevents information losses and checks the correctness of entered data. Handling this requirement, all input fields should be checked for syntactical, and where possible, for semantical correctness. Also files should always be checked for the right format. Integrity includes the check, if all data are written successful and not partial or not at all. Warning messages are a good tool, to provide feedback to the user, if any error occurs.
Interoperability describes the extent on how a system exchange information with other services or applications. Concreting this, already in die architecture planning, a clear separation of client- and server- functionality should be considered. Therefore other services can easily use the generic functions of the server. To go even further, the server-functions should be clearly separated. For this, a microservice architecture might be the way to go.
Performance describes the responsiveness of the system to various user queries and inputs. Beside the measure of response time, also the measure of data capacity and latency can be applied. For this purpose, program code, including algorithms and procedures, should be constructed as efficient as possible. While it is not possible to measure the efficiency, without running the application on real server, algorithmic theory, for example, can be used to classify algorithms according to their run time; therefore at least a theoretical measure can be applied.
Security blocks unauthorized users to access the system, including data and functions. Also the protection against hacking attacks should be regarded. The importance of security can be illustrated by reviewing a survey from Richardson. According to his survey of 522 computer security practitioners, 68% have and 18% will develop a formal information security policy. Only 1% have no security policies at all (Richardson and Director 2008). Nevertheless, for this prototypical implementation only minimal attention to security is given. Thus an authorization service should be implemented.
Usability can also be described as user-friendliness, ease of use, or human engineering. Thereby it measures the effort required for input actions, operating on it, and finally the interpretation of the output. For this purpose, especially the Graphical User Interface should be structured very clearly. Thus no instructions should be needed, but if nevertheless something goes wrong assistance and feedback messages should be provided. Furthermore, through utilizing modern web technologies, a good-looking design should be pursued.

Wiegers also defines internal factors for internet and corporate applications. While he only lists scalability, for this system also modifiability should be considered. This is because of the prototypical implementation, which needs the possibility for late changes and further development. Table 3.4 gives a short description of these two requirements.

Internal requirement	**Description**
Scalability	How easily the system can grow to handle more users, transactions, servers, or other extensions
Modifiability	How easy it is to maintain, change, enhance, and restructure the system

Table 3.4.: Internal non-functional requirements for internet and corporate applications following Wiegers (Wiegers and Beatty 2013, Chap. 14)

Similar to the external requirements a full list of internal requirements can be found in the book of Wiegers (Wiegers and Beatty 2013).
Scalability is the ability of the application to grow when the amount of users rise or when data streams increase. Of course this requirement could be tackled by increasing the computing capacity or the memory. This is not possible in the scope of this thesis. Nevertheless, the application itself can be constructed to be scalable. Thus server functionality should be programmed in a way to allow the replication and distribution on several parallel processors.
Modifiability describes how easily code can be understood, changed and improved. This requirement is particularly important for a prototypical implementation, while in the future developers might take this prototype as basis for their own applications or advance the prototype itself. For this purpose the code has to be clearly structured, independent functionality should be encapsulated as far as possible, and comments and documentations should be used extensively.

Now all the functional as well as the non-functional requirements are stated. The next step consists of the construction of an appropriate solution by modeling the structure and behavior of the application. For this purpose the requirements with the highest priority should be considered primary.

4. Modeling and Architecture

An accurate implementation requires models of the structure and behavior of the application. The goal is to communicate and understand the ideas of the software design, while abstracting details (Fowler 2004, preface). These provide implementation guidelines and point out coherences between different parts of the application. Thereby emerging problems can be detected early and further development and maintenance can be facilitated.
An accepted and widely used standard, specifying graphical notations, is the Unified Modeling Language (UML). It is provided by the Object Management Group (I. Object Management Group 2016). Dzidek et al. showed the benefits, by conducting experiments of implementations with and without the application of UML beforehand. Not only the functional correctness is significantly enhanced when using UML, also superior quality of programming code could be observed. Furthermore the complex systems, modeled with UML, had higher understandability (W. J. Dzidek, E. Arisholm, and L. C. Briand 2008).
UML specifies various diagrams, fulfilling different purposes. These can be clustered into two types: Structural and behavior diagrams. Structural diagrams describe the construction of software systems, whereby the static elements are irrespective of time. For example, a class diagram illustrates the structure of an object-oriented system, consisting of classes, attributes, methods, and relationships. On the contrary behavior diagrams describe the dynamic behavior of objects, including their methods, collaborations, activities, and state histories, over time. For instance, an activity diagram might describe the changing states of the system when the user undertakes a sequence of actions. (Object Management Group 2015, p. 725)
Notwithstanding the notation of UML is exactly specified, it is target-oriented to extend and modify the specification with custom elements. To goal is not to suffice the standards, but rather to build understandable and intuitive diagrams. In following sections the use case-, activity-, class-, and sequence diagram are employed. The class diagram is classified as structural diagrams, whereas the others belong to behavior diagrams. First the use case diagram reveals the use cases of the application and the interaction with its environment to provide an understanding of what (not how) the application is supposed to do. Thus it provides the general idea of the application. Subsequently the client-side is examined with the aid of activity diagrams. For the server-side a class diagram is used. Ultimately the communication between the components is delineated in a sequence diagram. For an overview of all UML diagrams with descriptions and examples the specification is recommended (Object Management Group 2015).

4.1. Use Cases

An use case diagram specifies what a system is supposed to do and how it connects with its external environment. This might be users, servers or other entities. It consists of actors, use cases, and their relationships with each other (Object Management Group 2015, p. 679). Hence an overview of the application is provided. At this point technical details are not considered. To separate the tasks of the application a 3-fold step to generate final reports and documentations is introduced. Figure 4.1 depicts an overview of this process. Also one abstract level below, the generated artifacts are delineated.

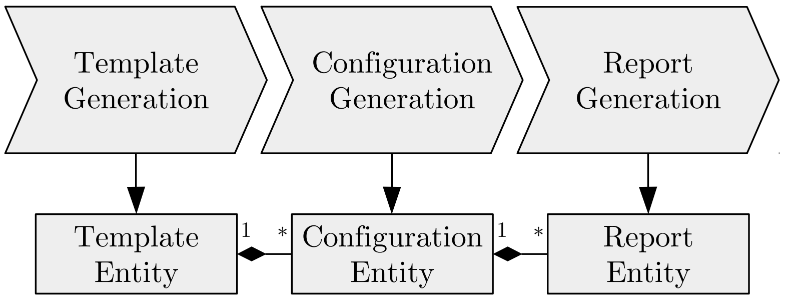


Figure 4.1.: Process and resulting artifacts of the application

First of all a template has to be created by the user manually, including the generation of a template-file. As in most businesses the standard office tools are prevalent, this file is created within these. This approach addresses the stated requirements, as formatting functions are fully available. By applying a template language, different types of parameters can be used. The used language is outlined in the implementation chapter. These parameters are extracted by the application to proceed with the next step, the creation of a configuration. For each step an entity is created within the SocioCortex system. Thus in this step a template-entity is created within the entityType *Template*.
To clarify the terminology: A *template* is used as generic term for labeling the whole process. It consists of several components (template-file, template-name, etc.). A *template-entity* is the representation of the template within the SocioCortex system. Therefore template and template-entity might be used interchangeable. The same pattern is used for the following steps.
In the configuration step, queries will be defined to link the parameters with data from SocioCortex. A description of the query language is also described in the implementation chapter, as technical details should be avoided in this section. The result is the creation of a configuration-entity. Also it will be stored in SocioCortex. One template-entity can correspond to many configuration-entities. As a consequence it

will be possible to use a general template to define different clusters of queries. For example, a configuration might address all applications, while another configuration might address only strategic relevant applications. Both could use the same template. The requirements of using sub-components is therefore met. Also the definition of standard values (e.g. Strings, Numbers, etc.) and more complex values (e.g. lists) are supported by utilizing a query language.
In the last step, the final report is created. Therefore the queries will be executed to retrieve the actual data and the parameters in the template-file is exchanged with the results. Different kinds of output formats should be available. In doing so text-templates, for example, should be convertible to .pdf documents. Also a report entity is created in the SocioCortex system. One configuration can correspond to several reports. This approach is important, as it allows the creation of updated documents without changing the document itself.

Hence the general procedure is explained, Figure 4.2 illustrates the whole application in a more coherent manner with the aid of an use case diagram. The primary actor of the application is the user. Secondary actors are the SocioCortex system and the server of the application. The functionality of the server is encapsulated, to fulfill the internal non-functional requirements of modifiability and scalability.
The overall goal is the generation of reports. Therefore this is the main use case. Also the creation of templates and configurations could be use cases itself. The reporting use case includes the creation of a configuration as a configuration is a precondition to generate reports. To generate a configuration, the precondition of a existing template has to be fulfilled. In the creation of a template the parameters are defined and in the creation of a configuration the queries are defined. In the creation of the final report queries are executed and the output format is chosen. As mentioned, the end of each process includes the creation of an entity in SocioCortex.
A precondition for the generation is a properly authentication, also settings should be defined. A setting might be the definition of the workspace in which the entities should be stored. By authentication the security requirement is addressed.

The use case diagram provided an overview of the application. In the next step the functioning of the client-side will be elucidated. For this purpose the notation of activity diagrams is utilized.

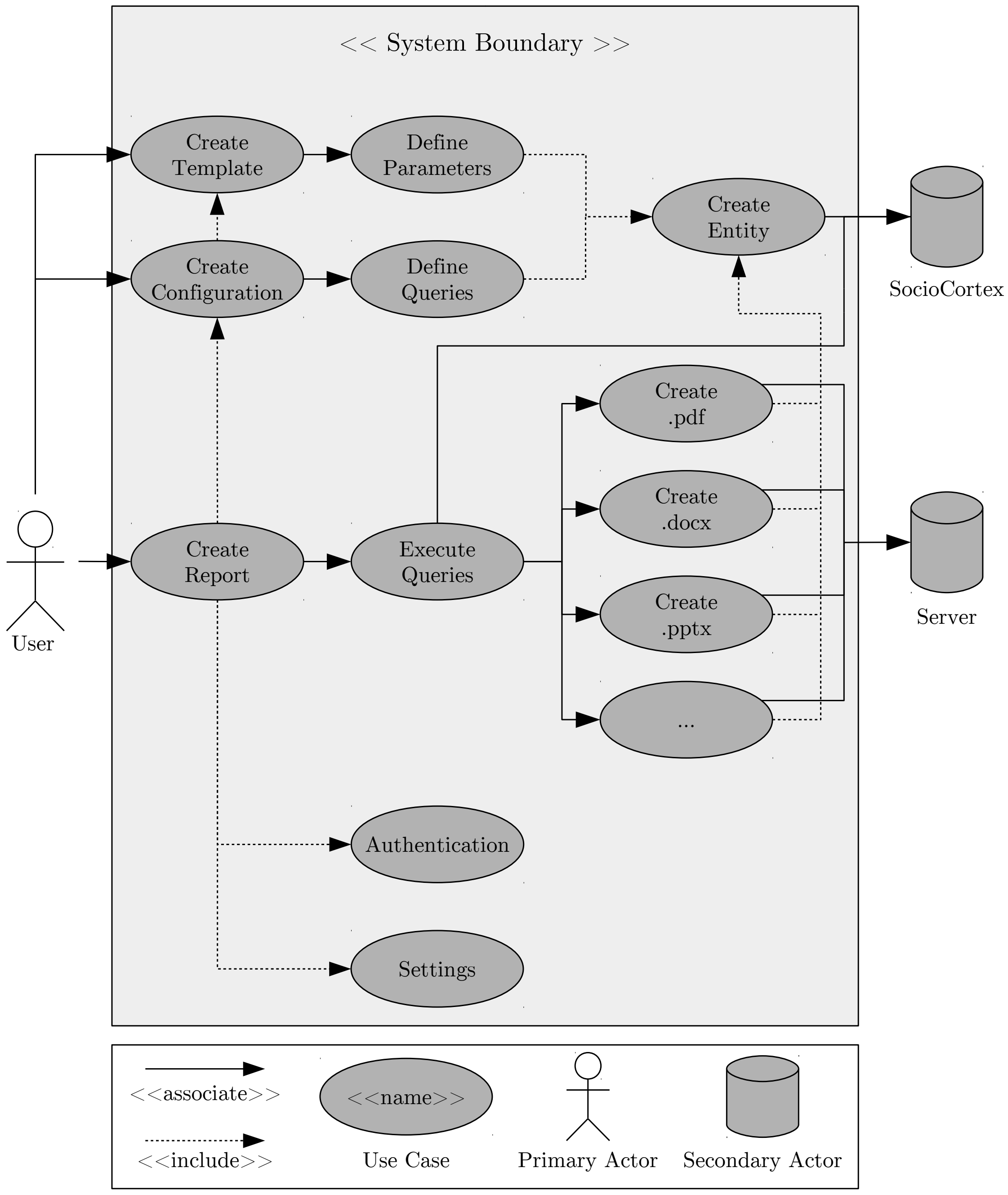


Figure 4.2.: Use Case diagram: Use cases of the application and its interaction with the external environment

4.2. Client-side

In this section the uses cases are examined in more detail. With the aid of activity diagrams, the execution of work-flows can be described by following a graphical notation. Different constructs, like conditions and iterations are possible and specified within UML. Hence this is the modeling of the client-side, a work-flow correspond to the navigation of the user via inputs on the graphical user interface. As the implementation of a web-application is intended, the different graphical surfaces correspond to different web-pages and web-dialogs. The implementation of the client-side can be looked up in section 5.2.
First of all, the user has to authorize himself, before any functionality can be used. This step is outlined in subsection 4.2.1 After successfully logging in, the steps, mentioned in the use cases (see Figure 4.1), will be passed trough. The first step, the generation of a template, is delineated in subsection 4.2.2. The second step, the generation of a configuration is outlined in subsection 4.2.3. In the final step, described in subsection 4.2.4, the creation of the final report is described.

4.2.1. Authentication

An authentication-mechanism is necessary to prevent an abuse of the system. This conforms with the non-functional requirement of security. Unauthorized persons should have no possibility to get access (reading and writing) to data. This factor is imperative, when running the application in real enterprises, as data is a huge part of the corporate secrets. While the application itself does not store and archive data, but rather processes all this information via the SocioCortex system, it is required to give the user only access, when he also has access on the SocioCortex system. The assignment of permissions and the registration-mechanism therefore is handled only in SocioCortex, consequently no new mechanism has to be implemented.
Therefore only a log-in function should be available. Further mechanism are imaginable when deploying the application in a productive environment. For example, the functionality of creating templates and configurations could be accessible only by a specific user group, while the creation of reports would be available for all users.
Figure 4.3 depicts the procedure in the UML notation. After opening the web application the user gets prompted to input the user name and the corresponding password. After checking the credentials in the SocioCortex system, he gets forwarded to the homescreen. If they are incorrect, an error message should notify the user. Messages and hints are instruments to increase the usability.

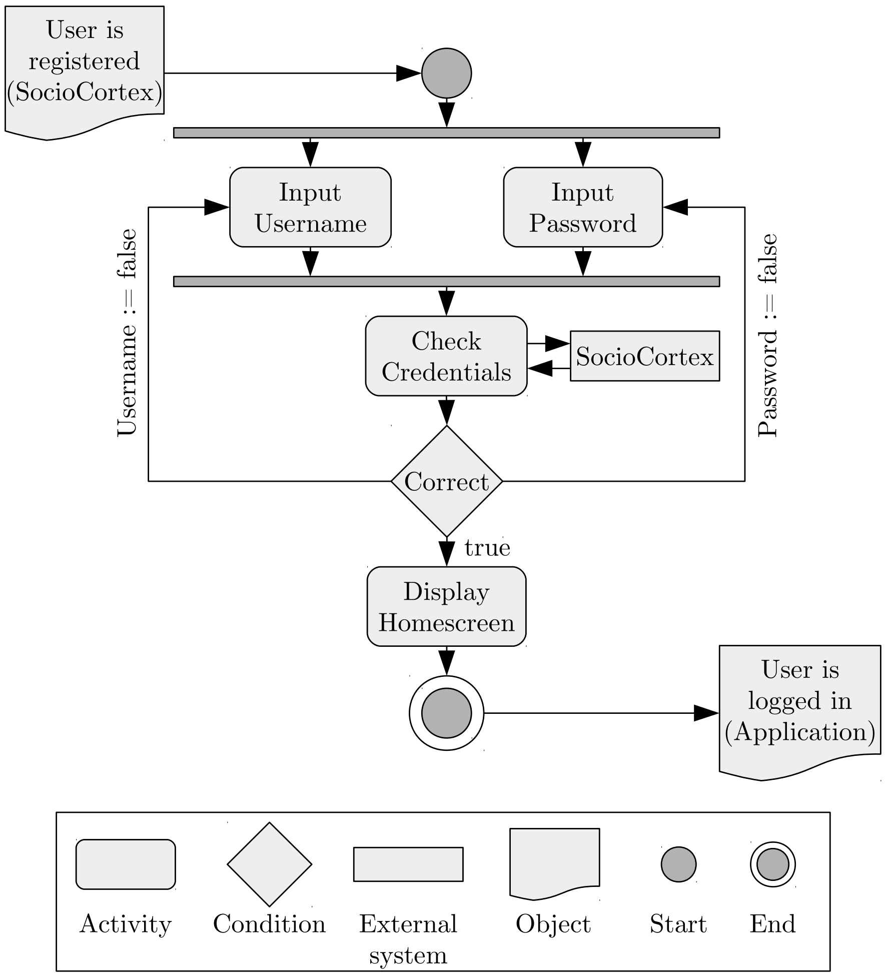


Figure 4.3.: Activity diagram: Client-side authorization mechanism

4.2.2. Create Template

After a successful authentication the homescreen of the application is displayed. It provides an overview of all templates, configurations, and reports, but has no inherent functionality. Consequently no modeling is necessary, as only the graphical surface has to be developed. For further development, of a more complex graphical interface, the technique of drawing mock-ups first is suggested.
Besides the overview of templates, the possibility of creating a new template is presented. To recap: The generation of a template is the first step towards the creation of final reports. For this purpose, also a concrete template-file is needed in the right format (.docx or .pptx). Other template formats are thinkable in future developments. The creation of template-files is described in subsection 5.1.1, as this involves the selection and application of a template language. In this modeling chapter it should be abstracted from detailed problems. Figure 4.4 illustrates the procedure of the creation of a template. The corresponding implementation is described in the subsection 5.2.2 of the implementation chapter.
First a new surface (e.g. dialog or page) is displayed, enclosing all activities of this step. In doing so, the functionality of creating a template can be easily encapsulated. As a consequence the programming code becomes more structured and therefore simpler to maintain. Furthermore the usability enhances, while the user exactly knows in which part of the process he is located.
On this page the user defines a name for the template first of all. Subsequently a template-file in an eligible format has to be uploaded. In addition the parameters utilized in that file has to be listed. Here the server provides the functionality to extract the parameters automatically. This function is described in more detail in section 4.3. Parameters could also be defined manually, in the case the extraction of the application is incomplete or erroneous. A parameter consists of a name and a type. The name identifies the parameter and the type serves as safety check in the generation of configurations. In doing so, incorrect defined queries can be identified more accurately. When the name, file, and all parameters are defined, the process can continue. Subsequently completeness and correctness are checked. When failures are identified, the process returns to the faulty spot and displays a feedback message. This cycle repeats until all entries are correct. Now a new template-entity is created within the SocioCortex system. SocioCortex serves therefore not only for the information gathering, but also provides a repository to store all generated artifacts (templates, configurations, reports). Finally the application returns to the homescreen and refreshes the template list.

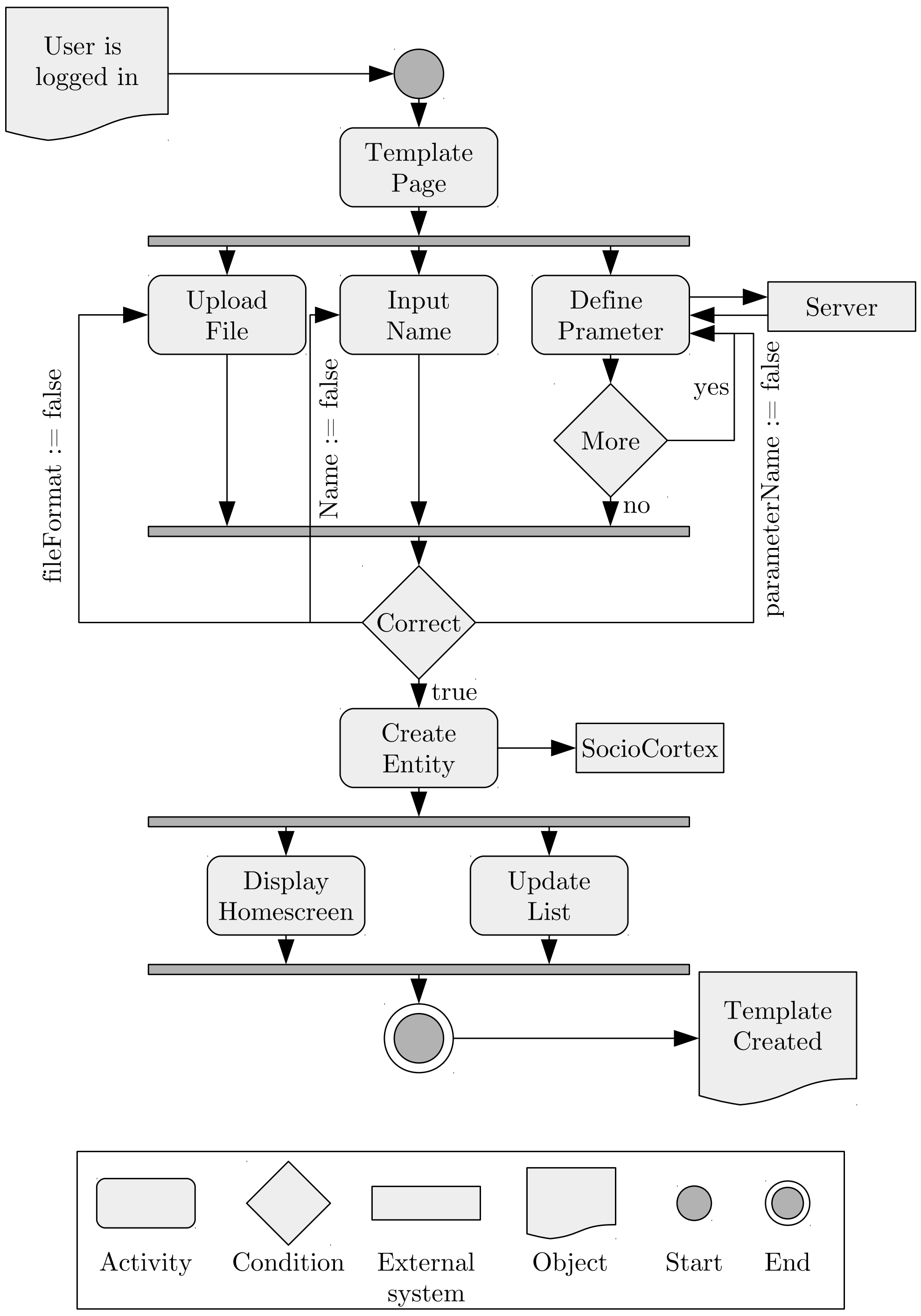


Figure 4.4.: Activity diagram: Client-side generation of a new template

4.2.3. Create Configuration

The creation of a configuration is the second step for the generation of the final report. Here the queries for the corresponding parameters are defined. For this step the query language MxL is used. A more detailed description, with examples, is stated in the implementation (subsection 5.1.2). However, only in the creation of the report the queries will be executed. The process is outlined in the activity diagram in Figure 4.5. The related implementation is elucidated in subsection 5.2.3 of the implementation chapter.

To create a configuration, preliminary an already created template has to be chosen from the template list showed at the homescreen. Doing so, a list of all created configurations for that template are displayed. Consequently diverse configurations might be created for one distinct template. An use case for this, for example, is the creation of configurations for different clusters of applications. Many other examples are imaginable, as long as the configurations share the same origin template.

Beside the list of already created configurations, of course the possibility of creating new ones exist. Like the creation of the template, all functionality should be encapsulated within one window (*Activity: Configuration Page*). As later described, each step can be implemented with its own controller, with explicit interfaces to the other controllers.

Beside the definition of a name, a workspace (see section 2.2 in Foundations) is selected. Thereby in further steps the entry of queries can be simplified. Thus keywords, entities, and entityTypes for the selected workspace can be suggested by the system. Furthermore the query can be checked for correctness.

The defined name and selected workspace is checked for completeness and correctness. Entering no name, or a wrong workspace should produce a feedback message to increase usability. When all entries are correct the next step can be executed.

Now for each parameter of the template a corresponding query has to be defined. To avoid failures when executing these queries in the report generation step, each parameter will be checked for correct syntax and type. For this reason, the definition of the type for each parameter was necessary. By doing this correctness checks, the non-functional requirement of integrity can be fulfilled.

As already mentioned, the queries will not be executed at this step. If syntax and type are correct, the last activities can be executed. As in the creation of template, a new entity will be created within the SocioCortex system. Subsequently the configuration is finally accomplished and the application can return to the homescreen, while parallel updating the configuration list with the new configuration.

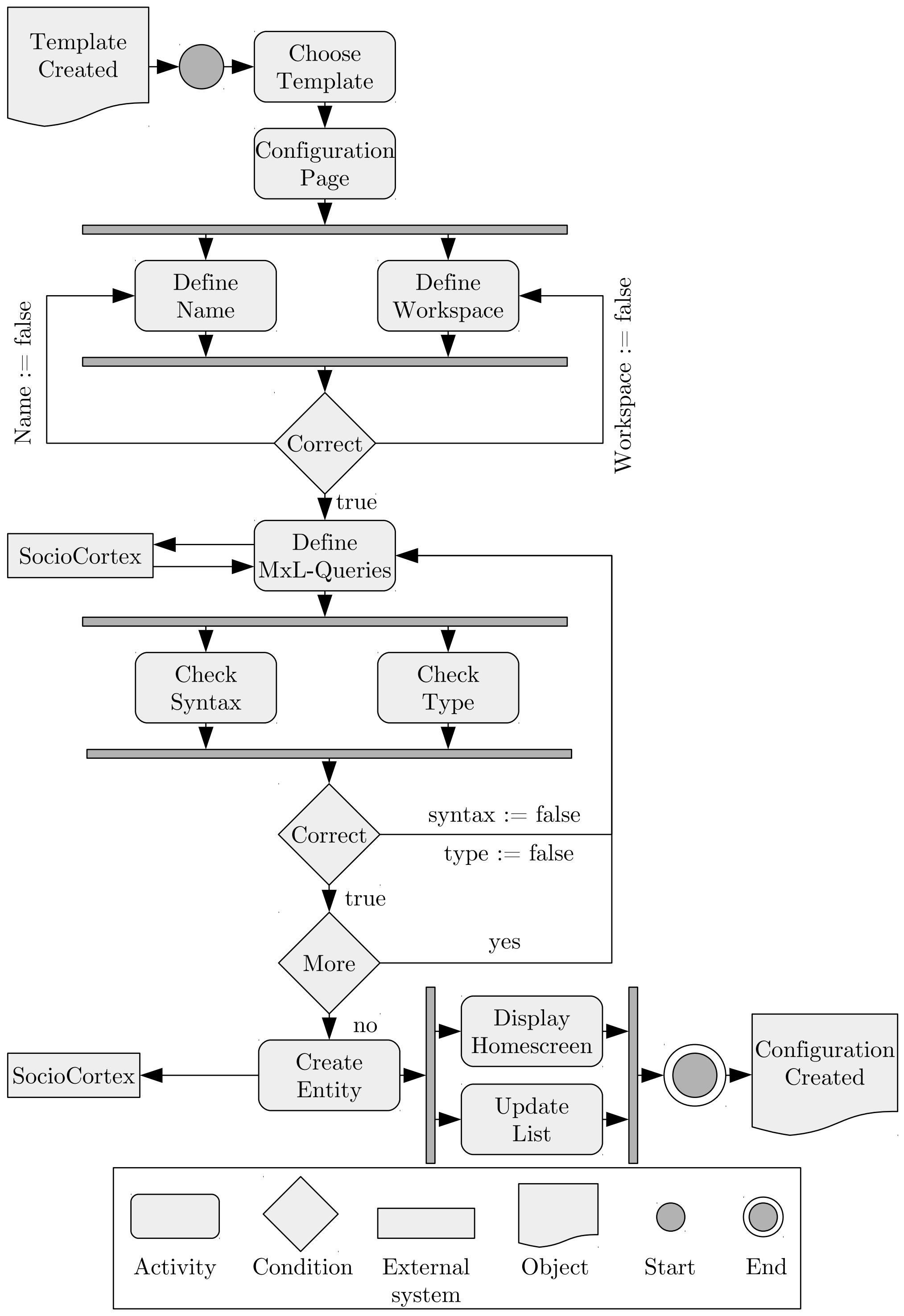


Figure 4.5.: Activity diagram: Client-side generation of a new configuration

4.2.4. Create Report

After the successful creation of a configuration and a template, the corresponding final report can now be generated. This is the last of the 3-step procedure.
For this purpose, a configuration has to be chosen from the homescreen, as a report always connects to a configuration. On the homescreen all reports for this configuration are now displayed. Therefore it is possible to create several reports for one configuration. By this approach, the problematic redundancy of components is prevented. Since the queries will be executed only in the creation of the report, changing data will always be up-to-date. Therefore, when a report is needed, it will be created ad hoc containing up-to-date content. Additionally this approach fulfills an indirect versioning mechanism, as outdated reports will not be deleted and are always accessible, even when new ones are created. Of course the possibility of deleting reports should also be included. A concrete version-history, however, is not intended.
The procedure of creating a report is delineated in the activity diagram of Figure 4.6. The corresponding implementation is outlined in subsection 5.2.4.
First of all, the dialog for the creation of a report is displayed. As already emphasized in the creation of templates and configurations, the whole functionality of this process can be encapsulated. This procedure requires the template-file created in step 1 and the defined queries, created in step 2. However, it does not accesses this information directly from the above mentioned procedures, but rather access their created entities within the SocioCortex system. Therefore the SocioCortex system represents the interface for the different steps.
Again a name has to be defined. This name does not only represents the name of the entity, but also serves as name for the created document. Additionally the required format has to be chosen. The application should hereby automatically propose the available formats. Consequently for text-template-files the possible output could either be text-files again or pdf-files. For this prototypical implementation, the formats .pptx and .docx should be supported. As output format additionally the .pdf format is available. Of course it should be easy to include additional formats in further development. Now the queries, defined by the configuration, are executed and their values are retrieved. All information are now present to create the final report. The template-file will be sent, together with retrieved values from the queries to the server. The server handles the creation of the final report and returns it. The server-side is outlined in detail in the next section 4.3.
As in the other steps an entity, containing the report-file, is created in SocioCortex. Therefore it is permanent available. The application returns to the homescreen and updates the report-list. In this list the report can be chosen, downloaded and used for the user’s purposes.

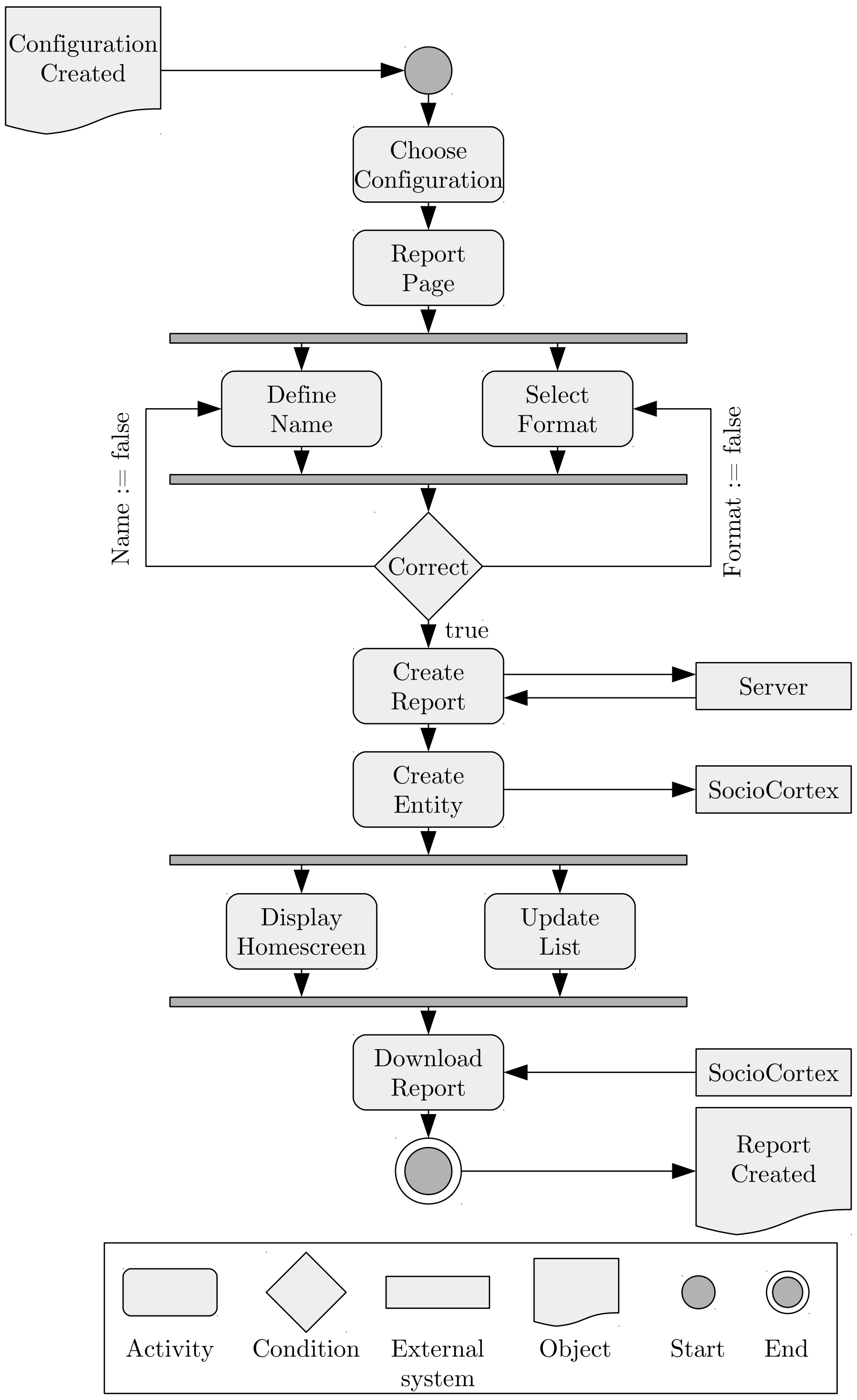


Figure 4.6.: Activity diagram: Client-side generation of the final report

4.3. Server-side

In the description of the different client-side procedures, different activities were supported by the usage of server-side functions. In this section their architecture and functionality is outlined in more detail.
To seperate each functionality the microservice architecture is utilized. Thus the internal requirements of modifiability and scalability can be fulfilled.
Microservices are small and autonomous units, handling only few specific tasks as good as possible. Oftentimes the programming-code of an application is growing very large and therefore becomes unmanageable, with regard to maintainability and further development. By applying microservices, the functionality of these applications are separated into small, clear, and monolithic services. Therefore the "Single Responsible"-principle is accomplished. The separation can go so far, that only by reading the name of the service, the underlying functionality should become apparent. Thus the code, handling one functionality is recognized instantaneously. Each service should be encapsulated from the other services, enhancing the reusability of functionality. Through this autonomy these services function independent of each other and furthermore, they can also be deployed independently. As consequence microservices can be distributed on different systems and are therefore very scalable. (Newman 2015)
To address a package of microservices an Application Programming Interface should be provided. In the context of web-applications the architectural style of REST-APIs is de facto standard.
As many descriptions and instructions are available on the internet, only a short explanation of REST-APIs is provided. A REST-API is utilized for the purpose of facilitating the communication between web-clients and web-servers. Therefore the REST-API can be seen as face of the web-server, listening and responding to requests, but hiding the procedures and functionality. Typical supported HTTP methods, used by an REST-API are following. (Masse 2011, pp. 4-6,23-27)

- **GET** to retrieve an element from the server
- **PUT** to replace an element on the server
- **POST** to create an element on the server
- **DELETE** to delete an element on the server

In this application mainly two functions are done by the server-side. First the support of defining parameters, by extracting them from template-files. This function is used in the activity diagram of the creation of templates (see Figure 4.4). The second functionality consists in the creation of reports, by substituting the parameters in the template-file with the retrieved values from SocioCortex. This function is used in the activity diagram for the creation of reports (see Figure 4.6). As different formats should be supported, a microservice should be provided for each one, as each format has their own characteristics. A pdf-output, for instance, requires the additional step of converting the document. Hence different enterprises request different formats, only the needed microservices can be supplied. For instance, an enterprise only utilizing .docx documents only the corresponding microservice has to be deployed. Figure 4.7 depicts the microservice architecture of the server-side, by utilizing an modified form of an UML class diagram.

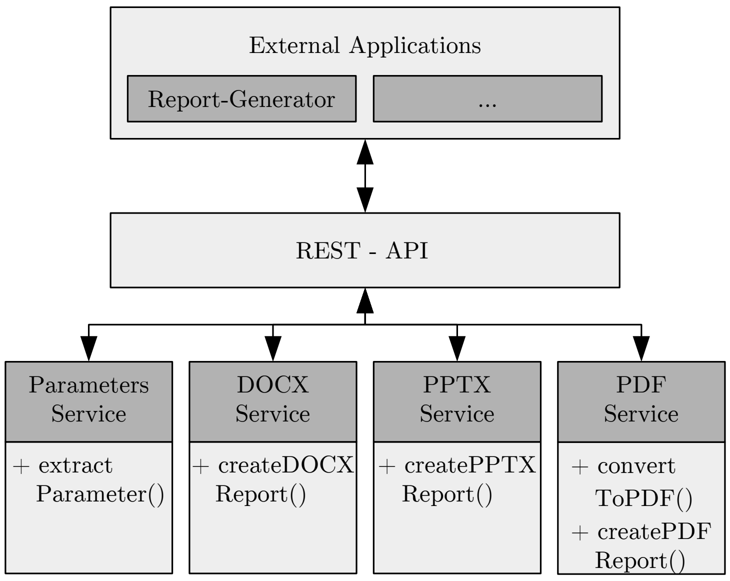


Figure 4.7.: Class diagram: Server-side structure utilizing the microservice architecture

The server-side therefore consists of 4 microservices: *ParametersService*, *DOCXService*, *PPTXService*, *PDFService*. Alone by reading the name, the function of the service should become clear. Thus the *ParametersServices* extracts the parameters in the definition of parameters, while the others create the final report in the desired format. The REST-API provides the access to these functionalities. As the functions are independent of each other and also from the client, other applications might utilize the provided functionality. The concrete realization of these services is described in the corresponding implementation chapter (see section 5.3).

4.4. Interaction of the Components

While also the architecture of the server-side is described, all parts can be assembled to show the communication of the different components. This includes the server-side, client-side, and SocioCortex. For this purpose the UML sequence diagram is utilized. It is a behavior diagram and a subtype of interaction diagrams. In contrast to a structure diagram it is time-dependent. Therefore different actions are executed in temporal order. The different components exchange messages and data.
Thus this diagram examines another view on the application and consequently facilitates the understanding and implementation of it. The sequence diagram for this application is illustrated in Figure 4.8.

The client, depicted in the middle line, is the central unit of this application, as it manages request calls to the server and to SocioCortex. The server and SocioCortex have no connection to each other at all. Also it represents the interface to the user. So by using this application the user neither cares about the server nor SocioCortex functionality.
In the first step, the users authenticates in the application. The credentials are verified with the aid of SocioCortex. When the credentials are correct, access to the crucial functionality is provided.
As outlined in more detail in subsection 4.2.2 the client (by the user) defines the template. It uses the microservice from the back-end, as it extracts the parameters from the template-file and therefore supports this step. Also a template-entity is stored for later access in SocioCortex. In the next step, as outlined in subsection 4.2.3 the configuration is created. Queries are defined and validated and a configuration-entity is created. In this step no functionality of the server-side is used. In the last step the report is created. For this purpose, the corresponding template and configuration is needed. Therefore these entities are returned from SocioCortex. Subsequently the queries are executed, thus the results are retrieved from SocioCortex. In the next step, the server creates the final report by processing these data. A report entity is created, storing also the final report-files. Ultimately the user can download these files.

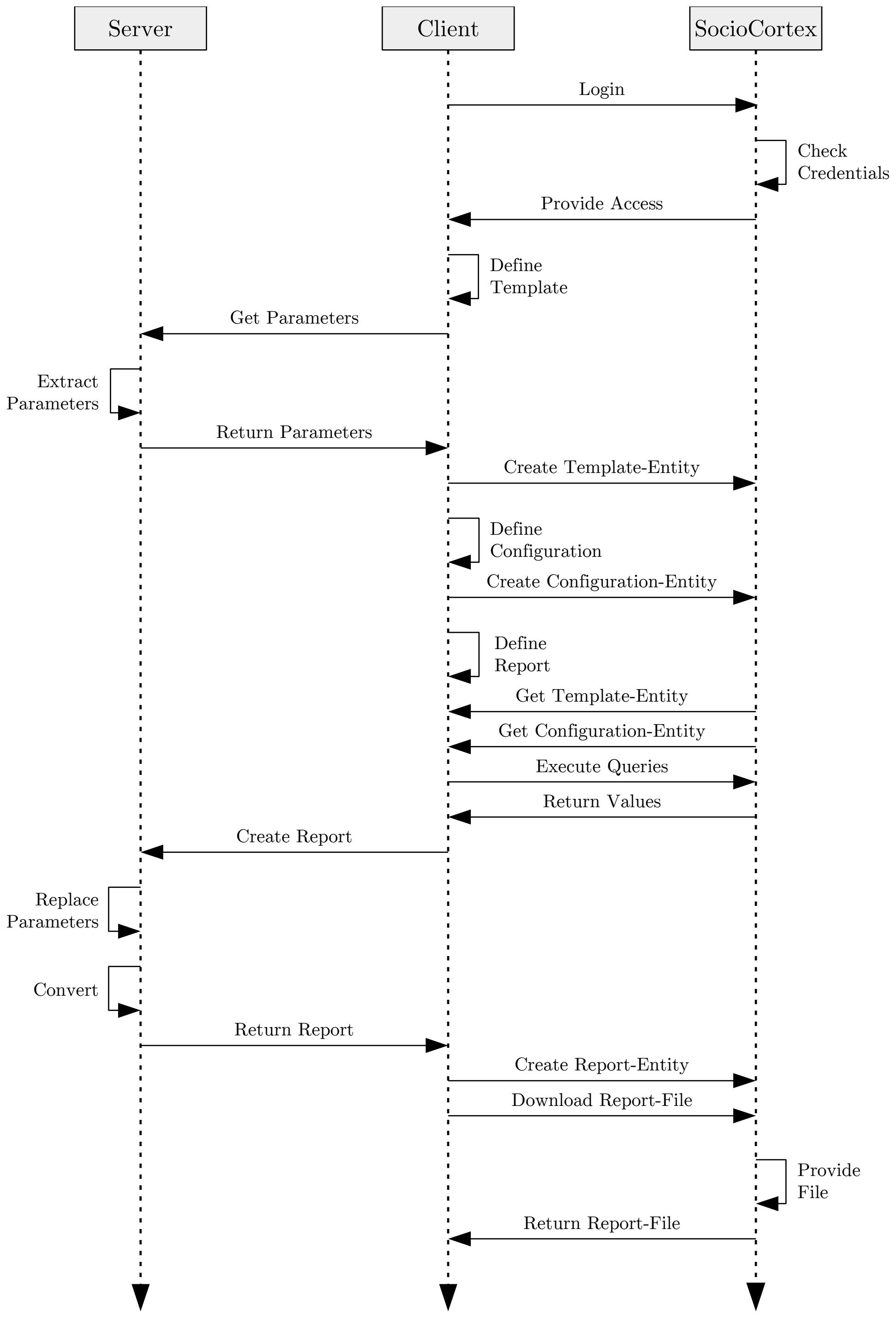


Figure 4.8.: Sequence diagram: Interaction between client, server, and SocioCortex

5. Prototypical Implementation

The client's and server's structure and behavior was modeled with the aid of UML. Based on these, the implementation of the application can be undertaken. The full programming code is freely available at the SEBIS GitHub-repository under the project "sc-reportgenerator" (SEBIS - GitHub 2016).
By the demand of growing sophistication, modern software steadily increases in complexity (Winter and Bhattacharya 2012, p. 244). While it is neither possible, nor reasonable to implement every part of the software by oneself, specialized libraries and frameworks are used for specific functionality. These external resources, utilized throughout this implementation, are explained and clarified in section 5.1.
Afterward, guided by the models and architectures, the implementation of the client-side is elucidated in section 5.2. Accordingly the authentication, the creation of templates, configurations, and reports will be clarified. In section 5.3 the functionality of the server is implemented. As outlined in the corresponding architecture, the idea of microservices will be utilized. The server generally fulfills two functions: The extraction of parameters from template-files and the final creation of reports, by replacing parameters with the corresponding query results.

5.1. External Resources

The starting point is the creation of template-files. For this purpose a template-language is used to differentiate parameters from plain text and furthermore, to provide additional functionality. There are proficient template languages available. Thus no own implementation is necessary. The choice on the utilized language fell on Velocity. Exact explanation and examples are provided in subsection 5.1.1. To query data from the SocioCortex system, the Model-based expression Language (MxL) is used. Its designed to query data from Hybrid-Wiki-based systems and therefore perfectly suitable for application in connection with SocioCortex. The syntax and functionality is presented in subsection 5.1.2. To replace the defined parameters in the template-file with the corresponding results from the MxL queries, a library, namely XDocReport, is utilized in server-side implementation. This library is introduced in subsection 5.1.3. Finally the REST-API of SocioCortex is outlined to access crucial functionality of the system.

5.1.1. Creation of Template-files: Velocity

In the creation of template files the Microsoft Office formats .pptx and .docx are supported. Although further formats are conceivable, these should suffice for a prototypical implementation, as they are widespread in practice. The concrete file therefore in created manually within the corresponding desktop-based tools (Microsoft Word and Microsoft PowerPoint).
While creating this template-file, most formatting functionality of these tools can be used. But instead of copying the relevant data from SocioCortex, parameters are specified at the required positions. The actual content will be later inserted by the application.
For this purpose a template-language, namely Velocity, is utilized. As the real task of this language is to reference objects in the Java-programming language, it provides an appropriate syntax to reference other components as well. Moreover further constructs, for example conditions, are provided.
Parameters defined in .docx files are created within MergeFields. An instruction on the usage of these fields can be looked up on the Microsoft support (Microsoft Office-Support 2016). Parameters defined in .pptx fields are simply created as normal text. To label the parameters the $-sign is prefixed. Following an example is listed.

```
Hello. My name is $Name_String. I was born in $Birthday_Date. Consequently i
am $Age_Number years old.
```

Listing 5.1: Example of using Velocity to demonstrate the syntax of parameters

Additionally to the standard Velocity syntax, the type of the parameter can be indicated. This is optional, but facilitates the definition process of parameters in the application, as it allows an automatic extraction (the definition of parameters is part of the template creation process: see subsection 4.2.2). An exception is the definition of lists (sequences), as they always have to be specified manually.
Beside simple parameters, also sequences can be used. As .pptx formats do not allow the use of MergeFields, sequences are only partially usable (only in bullet point lists). A sequence can be loop by following syntax.

```
#foreach($app in $applications)
  $app.Title
  $app.Description
#end
```

Listing 5.2: Example of using Velocity to demonstrate the iteration over sequences

As mentioned also the usage of condition is available. Hereto classical if-statements are applied. The following example assumes the documentation of an application-entity with an attribute *applicationPoints*. Dependent on the instance, different sentences appear in the final documentation.

```
#if($applicationPoints_Number == 5 || $applicationPoints_Number == 4 )
    $applicationName_String is of strategic importance.
#elseif($applicationPoints_Number == 3 || $applicationPoints_Number == 2)
    $applicationName_String is of operational importance.
#else
    $applicationName_String is obsolete.
#end
```

Listing 5.3: Example of using Velocity to demonstrate the the usage of conditions

As the examples above illustrate, very general templates can be created. By only creating one template for a technical documentation, for instance, all applications in companies, assumed a properly data basis, can be documented without copying the data manually from the systems of each application. Furthermore the final documents are all consistently formatted.
Further descriptions and examples are outlined on the website of the Velocity project (Apache Software Foundation 2016).

5.1.2. Querying SocioCortex: Model-based expression Language

To fill the parameters with values, the data in the SocioCortex system has to be queried. For this purpose the Model-based expression Language is utilized. MxL is a domain-specific language, building on the data-model of SocioCortex (see section 2.2). It allows the definition of queries, business rules and constraints. Since this application utilize MxL to retrieve the data, only the querying function is applied. MxL is characterized by following features (Software Engineering for Business Information Systems 2016c).

- ***Functional***
  The language is characterized by invoking functions. As a consequence for a typical query operation, like "select", a corresponding function gets called (select-function, where-function, etc.)

- ***Sequence oriented***
  MxL concentrates on the usage of sequences (ordered sets) and supplies various functions to support these. More on the usage of sequence in the subsequent course of this chapter.

- ***Object oriented***
  SocioCortex entities are considered as objects and entityTypes as classes. Therefore the data-model, mentioned in chapter 2 can be queried.

- ***Statically type safe***
  The static-semantics is validated as soon as the user enters a query. By analyzing semantic dependencies automated refactoring is possible.

On the definition of queries, the types depicted in Figure 5.2 are supported. Thus simple as well as complex attribute types are part of the type-system of MxL.

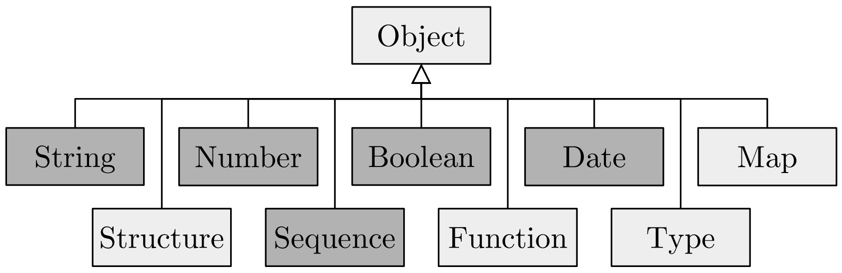


Figure 5.1.: Basic types supported by the Model-based expression Language

The template language Velocity supports basic attributes and lists, therefore the MxL-queries usable in this application are limited to the highlighted attributes.

To implement lists the MxL type *Sequences* is used. In velocity a list basically consists of objects with various attributes (e.g. $Application.Title, $Application.Points). To reproduce this in MxL, Sequences of Structures are used. A *Structure* consists of basic attributes (excluding *Sequences*). Following diagram illustrates the allowed types in the context of this application.

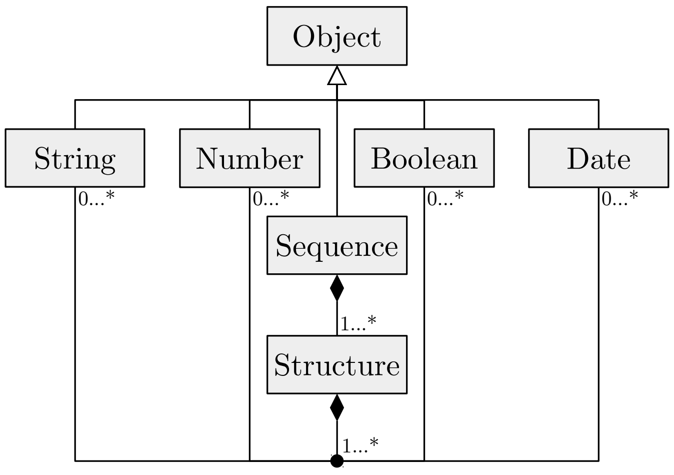


Figure 5.2.: Type system supported in the context of the report-generator

The user defines the type of the parameters within the application. By using the appropriate syntax in velocity, this procedure can be supported automatically. The type of the parameters has to be equally to the type returned by the queries. If not, the query will throw an exception. This procedures prevents the user to formulate wrong queries and therefore serves as additional integrity component.

As MxL is sequence-oriented and the report-generator also supports this type, the usage of various functions can be utilized. These functions are based on the Microsoft standard query operators and are applied on the *Sequence* type (Reschenhofer, Monahov, and Matthes 2014, p. 2). A full list of functions are listed on the introduction to MxL (Software Engineering for Business Information Systems 2016c).

- ***Query operators***
  Query operators are used to select, return, and filter data. Examples are *select()*, *where()*, and *groupBy()*.

- ***Aggregation operators***
  Aggregation functions providing a single output value, while inputting a source sequence. Examples are *count()*, *sum()*, *min()*, and *max()*.

- ***Quantifier operators***
  Quantifier functions, return either true or false (boolean), while inputting a sequence as source. Examples are *isEmpty()*, and *contains()*.

- ***Set operators***
  Classical set operators are provided. Based on a source sequence a sequence of the same type is returned. Examples are *distinct()* or *except()*.

- ***Element operators***
  Element operators return a distinct element of a sequence. Examples are *first()* and *last()*.

- ***Partitioning operators***
  Partitioning operators split the source sequence and return a smaller sequence of the same type. Examples are *rest()* and *skip()*.

MxL also comes along with a build-in query editor. It is applied and described in the section of the configuration creation (subsection 5.2.3).

5.1.3. Creation of Reports: XDocReport

XDocReport is a Java-API to merge a XML-based document (like .docx or .pptx, but also Openoffice or Libreoffice formats) with a given Java model to generate a final report. Also converting to .pdf is possible (opensagres 2016). The template-language Velocity and Freemarker are supported (Apache Software Foundation 2016) (*Apache FreeMarker* 2016). Therefore this API is perfectly suitable for the usage in the server-side implementation.
As this is a pure Java-API, also the server-side has to be implemented in this programming language. Further details herto in section 5.3. The usage of XDocReport can be summarized in following code snippet.

```java
// 1) Load template-file and set template language
InputStream input = new FileInputStream(templateFile);
IXDocReport report = XDocReportRegistry.getRegistry().loadReport(input,
                                                    TemplateEngineKind.Velocity);

// 2) Create a context to replace parameters
IContext context = report.createContext();
context.put("parameterName", value);

// 3) Generate report by processing the context
OutputStream out = new FileOutputStream(reportFile);
report.process(context, out);
```

Listing 5.4: Report creation process with the usage of XDocReport

This code-snippet only demonstrates the functioning of XDocReport; for a working code-fragment it is referenced to the final code of the application.
First the template-file should be loaded into an Java *InputStream*. As the Java-specification and many other books explain the usage of standard in- and output features of Java, an explanation is omitted. Furthermore the used template-language is defined. In this case Velocity is used (*TemplateEngineKind.Velocity*). In the next step a context is created to replace the parameters with the corresponding value. Finally an *OutputStream* is generated for the final report and the context is processed.
The second main functionality of XDocReport is the usage of converters. As a requirement, different formats should be producible. Mainly the support of .pdf formats are required, due it is not editable. Listing 5.5 shows the implementation of a converting process.

```java
// 1) Create option: DOCX to PDF
Options options = Options.getFrom(DocumentKind.DOCX).to(ConverterTypeTo.PDF);

// 2) Get the converter from the registry
IConverter converter = ConverterRegistry.getRegistry().getConverter(options);

// 3) Convert DOCX to PDF
InputStream input = new FileInputStream(DOCX_File);
OutputStream output = new FileOutputStream(PDF_File);
converter.convert(input, output, options);
```

Listing 5.5: Converting process with the usage of XDocReport

First the document type and the resulting type are declared. In the second step a converter is provided. Finally the converter processes on an input-file and creates the corresponding output-file in the defined format.

5.1.4. Accessing SocioCortex: REST-API and sc-angular

MxL serves to query the data-model of SocioCortex. But also the function of creating and editing entities is needed. Thus SocioCortex also should serve as repository to store the created artifacts. Furthermore an authentication mechanism is needed.
For this purpose SocioCortex provides a REST-API. In Listing 5.6 are examples of the most needed operations for this thesis. Of course there are many more operations available. A full list is provided in the SocioCortex documentation (Software Engineering for Business Information Systems 2016b).

```
// Create a new entity
POST /entities

// Returns an entity with given ID
GET /entities/{entityId}

// Get all available workspaces
GET /workspaces

//create a new file
POST /files

//Handle authentication by providing a JSON Web Token
GET /jwt
```

Listing 5.6: Example requests of the SocioCortex REST-API

To facilitate the usage of this REST-API, SocioCortex provides the library sc-angular, used within AngularJS, wrapping the operations in easy-to-use functions. When possible the sc-angular library is used throughout this implementation.

Now all the utilized external resources are explained. These have to be used in an approrpiate way. In the first step of the implementation, the client-side will be elucidated.

5.2. Client-side

In this chapter the implementation of the client-side is described, corresponding to the models provided in section 4.2. Because of the growing demand of web-applications in enterprises, the client should be accessible via a website. This fact is also due to the improvements of technology. New versions of JavaScript, HTML, and CSS emerged and entirely new applications, with respect to usability and functionality, could be build. The emergence of AngularJS is part of this evolution, by allowing the development of productive, flexible, and maintainable web applications, without the usage of plug-ins, like Flash or SilverLight (Branas 2014, Chap. 1). Also other client-side frameworks emerged, though the choice was AngularJS, because of the support of the SocioCortex REST-API with the package sc-angular, as described in subsection 5.1.4.
As outlined in the use cases of the application (section 4.1) there are 3 steps towards the final creation of a report. The creation of templates, the creation of configurations, and the creation of the reports. To provide a navigation for the end-user a central screen was implemented. It is depicted in Figure 5.4. For this screen, as well as for all following illustrations it should be remarked that this are only prototypical realizations. By inspecting the application in the future, these screens might change in form heretofore. Additionally only cutouts of the relevant elements are provided.

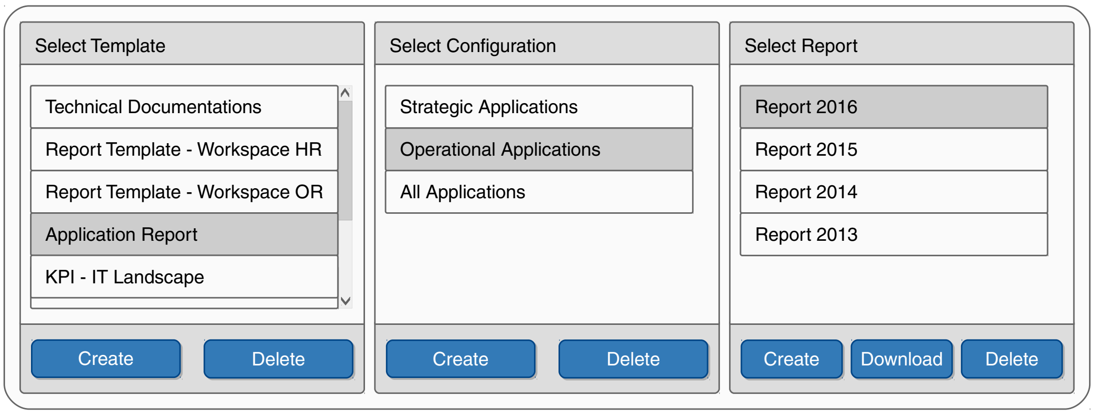


Figure 5.3.: Implementation of the homescreen

In the further description this screen is labeled as *homescreen*. Like the process itself it is splitted into 3 different parts. The used example might clarify the reason for this approach. A list of available artifacts is provided for each step. Also the functionality of creating the corresponding artifact is provided. Each part is described in the next sections. Apart from that, no critical functionality is done in this homescreen.

5.2.1. Authentication

The system only accesses resources from SocioCortex. Consequently no registration mechanism has to be developed. Thus the above illustrated homescreen can only be accessed through successful authentication. Access is only permitted if a valid account is registered on SocioCortex. As described in the model, the user will be prompted, when opening the application, to input his user name, and password. After submitting, the corresponding function is called to validate the inputs. This simple procedure is depicted in Figure 5.4. The warning message only appears, when the user entered wrong credentials. When everything is correct, he gets forwarded to the homescreen.

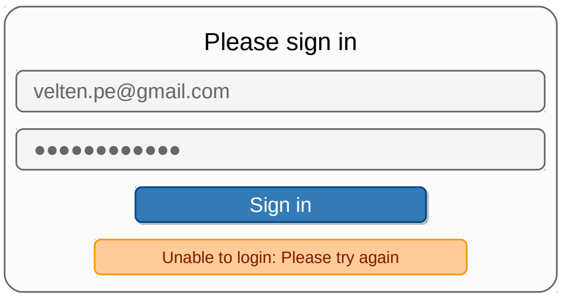


Figure 5.4.: Implementation of the authentication procedure

The function of checking the entered credentials is done with the aid of the sc-angular package. Following listing shows the important code-fragments, while omitting details.

```
// Gets called when clicking 'Sign in'
$scope.login = function(){
    //call the login-function from sc-angular
    scAuth.login($scope.user, $scope.password)
        .then(
            //login successful
            function (result) {
                //Code when successful
                //Navigate to homescreen
                $location.path("/home");
            },
            //login not successful
            function(reason){
                //Show error message
                //No further navigation
            });
};
```

Listing 5.7: Authentication mechanism utilizing sc-angular

Although this mechanism behaves as expected, it became apparent, that there is the possibility of navigating to the homescreen by following external links, or by direct entering the corresponding URL (.../home), without correct authorization as described above. In this case, the user obviously could not harm any data, while access to critical SocioCortex-functionality is not granted. Nonetheless this behavior is not destined and further checks for authorization should be avoided. Therefore a procedure to prevent this behavior has to be developed. For this purpose the attempt of changing the location without proper authorization should be prevented. Following code snippet reveals the function to implement such a mechanism.

```
1 //Triggers event when changing location (URL)
2 $rootScope.$on('$locationChangeStart', function (event) {
3    //Check if user is authenticated
4    if( !scAuth.isAuthenticated() ){
5           //User is not authenticated -> prevent the navigation
6           event.preventDefault();
7    }
8 }
```

Listing 5.8: Prevent navigation for non-authorized users

The function *scAuth.isAuthenticated()* from the sc-angular library is used in line 4. It returns the boolean value *true*, if an user is logged in. By calling *event.preventDefault()* the navigation to an other URL is prevented. Again the crucial functionality is pointed out and details are omitted.
Logging out is possible with the simple method *scAuth.logout()*.

5.2.2. Create Template

In this section, the activity diagram for the creation of a template is implemented (subsection 4.2.2). The name and the parameters have to be defined and a template-file, as described in subsection 5.1.1, has to be uploaded. Of course, to use the automatic extraction of parameters, the template-file has to be uploaded beforehand. Figure 5.5 pictures the implementation of these functions.

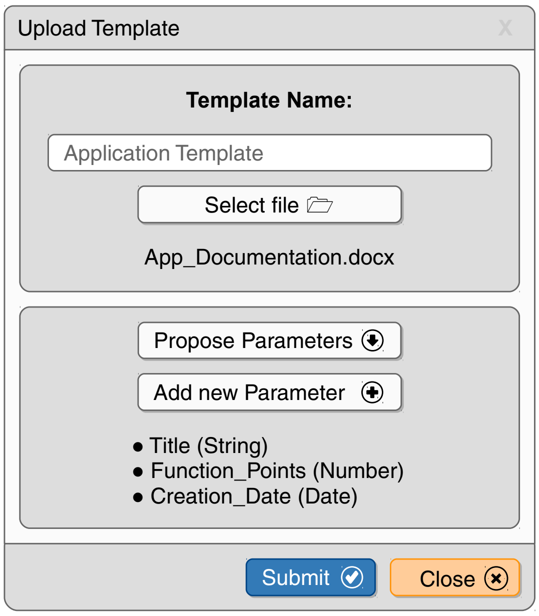


Figure 5.5.: Implementation of the template generation procedure

The user defines each parameter, which is used in the template. Otherwise the parameter can not be used in later steps. A parameter composes of a name and a type. The type, is used to validate the queries in the creation of the configuration. In doing so wrong defined queries can be recognize by the system. To facilitate the procedure of manually typing in every parameter (the amount of parameters is theoretically unlimited), the function of proposing parameters is available. The button is only clickable if an actual template-file is uploaded. The microservice ParametersService is called as described in Listing 5.9.

```
// Gets called by clicking 'Propose Parameters'
// This function is only available if a template-file is uploaded, thus no
security checks are needed
 $scope.proposeParameters = function(){
   //set loading bar to show a running process
   $scope.loadingBar = true;

   // Use helper-library Upload to facilitate the uploading-process.
   // Call POST /proposeParameters and pass the template-file as argument
   Upload.upload({url: '/proposeParameters',
                data: {
                  file: $scope.file
                }
            }) .then(function(response) {
                //Server call successful
                //Assign response to variable (json of parameters)
                $scope.parameters = response.parameter;
            }, function(response){
                // Server call not successful
                // Show error message
            }).finally(function(){
                // In both cases, stop the loading bar
                $scope.loadingBar = false;
            });
 };
```

Listing 5.9: Call server operation to extract parameters from a template-file

The manual adding of parameters is depicted in Figure 5.6.

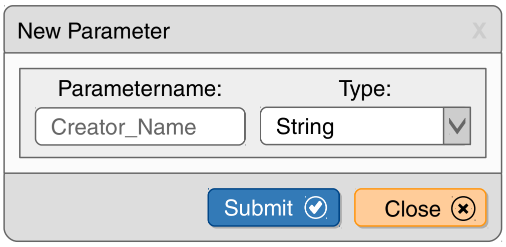


Figure 5.6.: Implementation of the manual definition of parameters

Finally, following Listing 5.10 demonstrates the creation of an entity within the Socio-Cortex system.

```
//create varibale, containing all information to create an entity (name,
attributes, entityType)
var entityAttributes = scData.Entity.arrayifyAttributes({
    name: $scope.templateName,
    attributes: {
        //Here all attributes with values
        Parameters: $scope.parameters,
        ...
    },
    entityType: {
        //Get the Template entityType ID
        id: $scope.getTemplateID()
    }
});
//Utilizing sc-angular function to create entity
scData.Entity.save(entityAttributes, function () {
   // Entity successfully created
}, function (data) {
   // Errors, while creating Entity; Show error message
});
```

Listing 5.10: Create a new entity in the SocioCortex system

To keep the code clear and comprehensible first the variable *entityAttributes* is declared. This variable defines all parameters, of which the final entity consists. Also the entityType is declared in this variable. Finally the variable is passed to the sc-angular function *scData.Entity.save()*. It creates a new entity in the SocioCortex system within the declared entityType. In the corresponding success and error function, it is defined what should happen in the respective situation. In case of success, the window can be closed and the user is forwarded to the homescreen. In case of an error, a feedback message should appear.

5.2.3. Create Configuration

After the successful creation of a template, a configuration can be generated. This procedure is modeled in the Architecture chapter in subsection 4.2.3. First a configuration name has to be defined and a workspace has to be selected. In a second step the definition of parameters take place. This two-step procedure has technical reasons. By opening the editors for the queries, the workspace has to be available beforehand. Figure 5.7 shows the implementation of the first step.

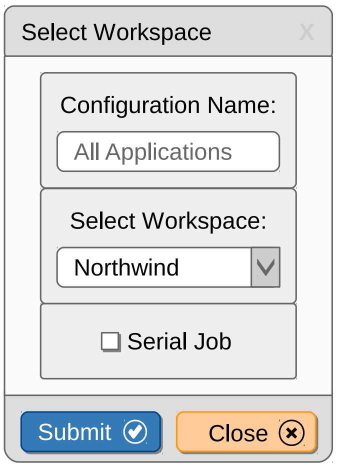


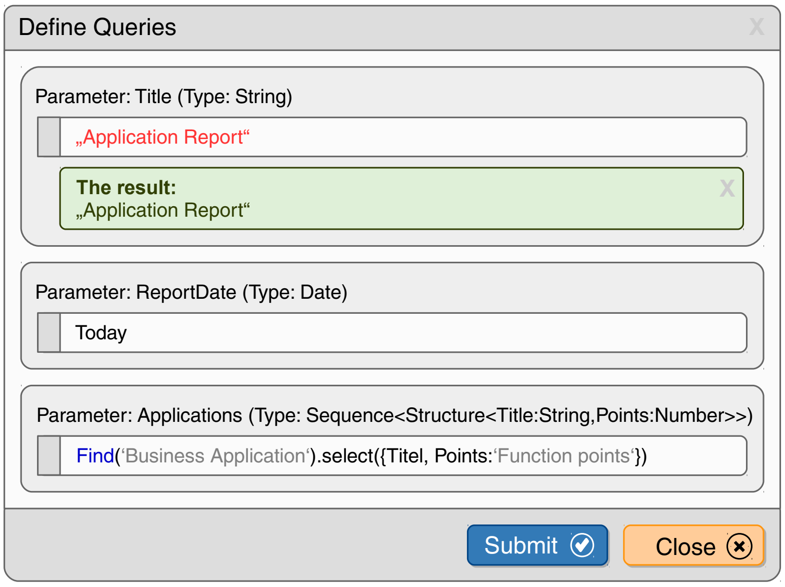
Figure 5.7.: Implementation of the first step towards the generation of a new configuration: Definition of configuration name and workspace

By choosing *Serial Job*, the procedure remains the same but the user is allowed to choose a distinct entityType. In the reporting step each report is then created for each entity of that type. Thus the requirement of a serial generation of reports is addressed.
A list of workspaces can easily be retrieved by utilizing following function. Of course it also would be possible to call the REST operation directly. The same is true for all other sc-angular operations. However, for more complex operations sc-angular simplifies the implementation process significant and to stay consistent within the code, most REST operations are executed via the sc-angular package.

```
scData.Workspace.query().then(
  function (workspaces) {
    $scope.workspaces = workspaces;
  });
```

Listing 5.11: Retrieve a list of available workspaces

After submitting, the user will be forwarded to the next screen, which displays step 2. Here MxL-queries will be defined. The implementation is depicted in Figure 5.8.


Define Queries

Parameter: Title (Type: String)

„Application Report“

The result:
„Application Report“

Parameter: ReportDate (Type: Date)

Today

Parameter: Applications (Type: Sequence<Structure<Title:String,Points:Number>>)

Find(‘Business Application‘).select({Titel, Points:‘Function points‘})

Figure 5.8.: Implementation of the second step towards the generation of a new configuration: Definition of MxL-Queries

The code editor, already mentioned in the external resources, of the MxL query language is used. Its easily assimilable into the HTML code. For each parameter an own input field is provided. These has to be created dynamically, while it is not clear beforehand, how many input fields are needed. The AngularJS directive ng-repeat is used for this issue. Each MxL-input field is passed the type and workspace. Ad hoc suggestions and checks are therefore possible. The entered query is stored for each parameter in the parameters array (*parameters[]*).

```
<!--angularJS directive for dynamically creation of MxL-editors -->
<div ng-repeat="param␣in␣parameters">
    <!--headline for each editor -->
    <label>Parameter: <i>(Type: {{param.type}})</i></label>
    <!--MxL editor with attributes workspace and type -->
    <mxl-expression class="mxl-form-control"
        ng-model="param.query"
        sc-workspace="{{workspace.id}}"
        mxl-expected="{{param.type}}">
    </mxl-expression>
</div>
```

Listing 5.12: Embedding of the build-in MxL-editor to define queries in the creation of a configuration

To make sure every query is defined correctly, they will be checked after submitting with *scMxl.validate()*. Thus it is possible to provide related feedback messages. Subsequently a configuration-entity is created within SocioCortex. This is analogous to the creation of a template-entity.

5.2.4. Create Report

Ultimately, after creation of template and configuration, the final report can be generated. Figure 5.9 depicts the implemented dialog for this step.

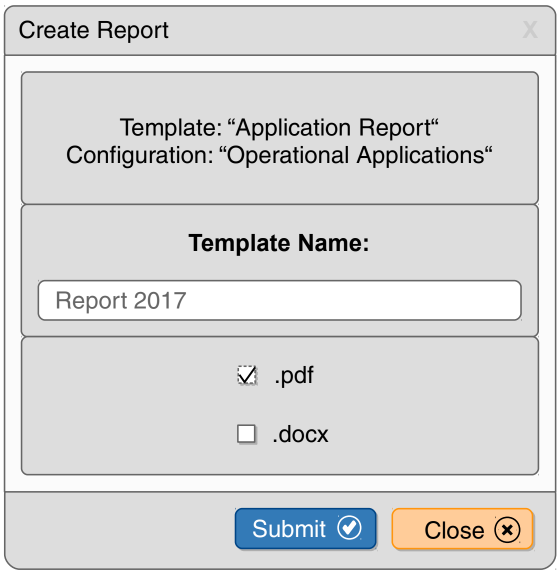

Create Report
X
Template: “Application Report“
Configuration: “Operational Applications“
Template Name:
Report 2017
.pdf
.docx
Submit
Close

Figure 5.9.: Implementation of the report generation procedure

The user defines the name and the format of the report. If all entries are correct the user can submit and therefore finalize the process. In the background the queries of the configuration are executed. The resulting values are subsequently sent to the server to create the report. The execution of queries is outlined in Listing 5.13.

```
//Get queries and associated type from configuration
var config = SharedVarService.getConfigurationAttributeParameters();
//Store promises in promiseList-array
var promiseList = [];
//Loop through each parameter
angular.forEach(config, function(item){
    //Call sc-angular function to execute query; pass workspace, query, and
    type
    //store promises, as all queries are executed in parallel (asynchronous)
    var promise = scMxl.query(
                  {workspace:
                  {id: SharedVarService.getConfigurationAttributeWorkspace()}},
                  { expression: config.query,
                  expectedType: config.type},
    function (result) {
            store the values in the same config array
            config.value = result.value;
    });

    //Use promises to prevent doing requests before all queries succeeded
    promiseList.push(promise);
});

//$q-function continues when all promises are resolved
$q.all(promiseList).then(function() {
  //continue with proceeding
  //Now final report-file can be generated
});
```

Listing 5.13: Execute queries to retrieve the values for the creation of the final report

As the server-side is constructed as microservice architecture, for each format an own operation can be used. The file together with the values have to be sent to the server, calling the required operation. To achive this, the file and the attributes have to be combined within a *FormData*. Listing 5.14 outlines this procedure.

```
//All promises resolved from Listing 5.13
$q.all(promiseList).then(function() {
    //Use FormData to combine several data to one variable, thus only one
    server call is required
    var formData = new FormData();
    //Append template-file to the formData
    formData.append("file", fileAsBlob, filename);
    //Append the parameters with the values (see Listing 5.13)
    formData.append("data", config);

    //Call POST /createPDF (the corresponding microservice to the required
    format)
    //Specify appropriate arguments
    $http.post('/createPDF', formData, {
            transformRequest: angular.identity,
            headers: {'Content-Type': undefined},
            responseType: 'arraybuffer'
        }).then(function (response) {
          //Server call successful
          //response contains the final report-final
          //create report-entity
        },function (response) {
          //Handle error here
        });
);
```

Listing 5.14: Combine parameters and template-file for a single server request to create the final report

Subsequently the application lists all available report-files. The user can download each required report separately, as depicted in Figure 5.10.

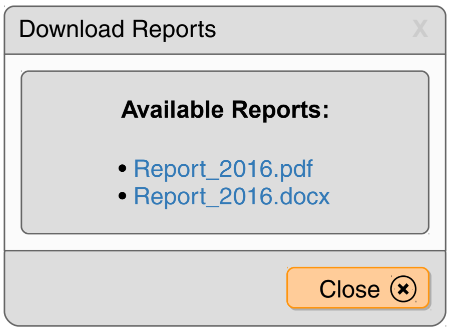


Figure 5.10.: Implementation of the list of final report-files

5.3. Server-side

As already mentioned in the external resources (see section 5.1) the API XDocReport is utilized for the server-side implementation. As this API is purely Java-based, the whole server is implemented in this programming language.
For this purpose the Play! framework is utilized. It facilitates server-, as well as client-side implementation by providing crucial functionality. By applying an asynchronous model and by engaging the paradigm of stateless functionality, highly scalable application can be developed. More information as well as extensive documentation can be looked up at the website. (Play! 2016)
The server generally is responsible for two functions. First it eases the definition of parameters, by extracting basic-type parameters from template-files. Second it creates reports by exchanging parameters with the corresponding query results.
By applying the microservice architecture these functionality are completely separated from each other. Therefore it becomes easy, for instance, to exchange the functionality of extracting parameters by a more sophisticated version. Each microservice provides its own REST-request. Following microservices with their corresponding REST-operations are implemented.

```
//Microservice: ParametersService
//Input: template-file
//Output: List of parameters
  POST /proposeParameters

//Microservice: DOCXService
//Input: template-file and query-values
//Output: final .docx report
  POST /createDOCX

//Microservice: PPTXService
//Input: template-file and query-values
//Output: final .pptx report
  POST /createPPTX

//Microservice: PDFService
//Input: template-file and query-values
//Output: final .pdf report
  POST /createPDF
```

Listing 5.15: Server-side REST-requests to address the different microservices

5.3.1. Extract Parameters

First the functionality of extracting parameters from the XML-based formats .pptx and .docx is explained. By requesting the corresponding REST-operation *POST /parameters* the template-file is transmitted to the server and the function *extractParameters()*, as defined in the server architecture (see section 4.3), gets executed.
To understand the extration process, first the file structure of the template-files is analyzed: A .pptx or .docx file are basically packed files, consisting serveral different .xml files. These files, like other packed files (e.g. .zip-format) can be unpacked with a standard unpacking-tool (The reader is suggested to try it out to see the intrinsic structure of these files). The general structure is depicted in Figure 5.11.

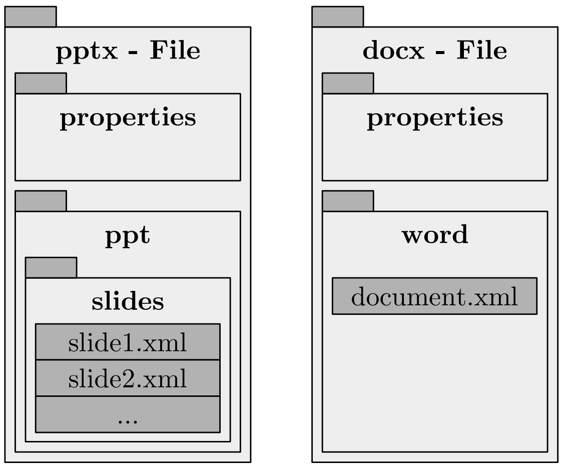


Figure 5.11.: File structure of Microsoft PowerPoint and Microsoft Word documents

Each document therefore consists of subfolders and files. A .pptx file mainly consists of two subfolders. In the folder *properties* several files are listed storing the properties of the .pptx file; for instance, the dimensions of the slides. In the second folder *ppt* the content of the document is stored. It also consists of a subfolder slides, which stores a numbered .xml file for each created slide in the document.
The .docx file is structured similarly. Also it consists of a properties folder, for the same purposes. Furthermore a folder *word* is available, storing a .xml file *document.xml*. This file hosts the content of the file (e.g. plain text).
To extract now the parameters, the files can be unpacked, and the above mentioned content files can be scanned for the parameters pattern. A parameter starts with a dollar sign ($). As this is the only required pattern, further patterns are added. Therefore the notation of ending the parameters with the corresponding type is sufficient. In

doing so, the pattern is adequate to accurately identify parameters and additionally it automatically specifies the type of the parameter. This added notation is optional. But the user has to define each parameter, without labeled by this notation, manually in the application.
These patterns can be specified with the usage of regular expressions. The notation should not be further explained. For interest the specification of the corresponding Java-library is suggested (Oracle America 2016b). Listing 5.16 shows the used regular expression (remark: dependent on the environment, escape sequences has to be added).

```
$[a-zA-Z0-9]*_(Date|date|Boolean|boolean|String|string|Number|number)
```

Listing 5.16: Regular expression to define the pattern of parameters

To realize the extraction of parameters, two steps have to be implemented: First the extraction of the template-files and second, the scan through the .xml-files with the aid of the regular expression. As the code is more complex, the following pseudo-code abstractly describes the procedure.

```
//specify regular expression to search in file
regEx := "...";
//specify parameters-array: should be filled within this function
parameters[];
//get template-file
file := fileFromClient();
//unpack the template-file and return all subfiles
subfiles[] := unpackFile(file);

//loop through all subfiles
foreach(subfile in subfiles){
  //check if subfile is content-related
  if(isContentFile(subfile)){
      //scan the content-file with the aid of the regular expression
      x[] = searchForRegex(subfile, regEx);
      parameters.add(x);
  }
}
//template-file successful scanned
return parameters;
```

Listing 5.17: Extract parameters by scanning through template-files with the aid of regular expressions

5.3.2. Generate Reports

The second functionality that needs to be done by the server is to generate the specific report. For every file-format a corresponding microservice is provided. This might create some redundant code. However, it can be decided for each environment which microservices should be deployed. Thus some enterprises might need no support for the PowerPoint format. In addition, the operations can be better parallelized, hence the microservices can be distributed on different processors.
For this functionality a file needs to be uploaded in connection with the parameters. Parameters are provided in a JSON format. It consists of the attributes *name* and *value*, wherein the value reflects the result of the query (subsection 5.2.4 describes the procedure on client-side). The following listing shows the corresponding server-side as pseudo code fragment.

```
templateFile := templateFileFromClient();
parameters := parametersFromClient();

// 1.) Initialize XDocReport procedure with templateFile: See subsection 5.1.3

// 2.) Loop over every parameter
foreach(parameter in parameters){
  //Replace parameters
  context.put(parameter.name, parameter.value)
}

// 3) Generate report by processing the context: See subsection 5.1.3
```

Listing 5.18: Server-side generation of the final report utilizing XDocReport

All features of the application are described. However, they are implemented prototypically and can of course be supplemented by further functionality. Concrete improvement approaches are described in the outlook (section 7.3).

6. Evaluation

Besides the building of artifacts, also the evaluation is part of the design science paradigm. It determines and assesses the utility, quality, and efficacy of the artifact. In this thesis the implemented application is evaluated. For the best evaluation results, the tool should be assessed in real business environments. Since this is not feasible within the scope of this theses, this procedure is simulated by presenting the tool to representatives of enterprises. Hevner proposes various types of evaluation methods. For novel and innovative applications the descriptive evaluation is mostly suitable. It is divided into the scenario-based- and informed argumentation approach. Both are engaged in the evaluation of this application. Figure 6.1 illustrates the evaluation methods according to Hevner, and highlights the applied ones. (Hevner, March, Park, and Ram 2004, pp. 85-87)

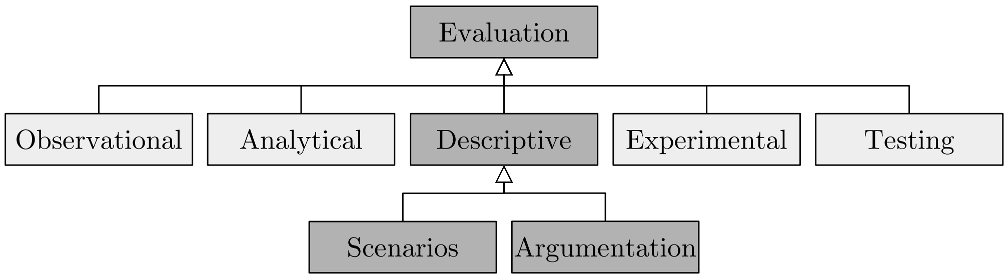


Figure 6.1.: Evaluation methods in the context of the design science research paradigm according to Hevner (Hevner, March, Park, and Ram 2004, p. 86)

In the scenario-based approach, use case scenarios are constructed and solved within a demonstration by utilizing the implemented application. The defined requirements will be assessed within this demonstration. In this evaluation this approach is used for assessing the functional requirements (see section 3.1). For deployment in productive environment, further evaluation methods should be applied. Thus the experimental and testing methods might be suitable to further assess the operational capability of the application. (Hevner, March, Park, and Ram 2004, pp. 85-86)
In the second part of this evaluation the non-functional requirements are assessed by providing factual arguments. This approach was taken, as many non-functional requirements, especially the internal, could not been demonstrated within a short period of time of a presentation. Thus the participants had to use and apply the application by themselves for a longer period to declare meaningful statements.

6.1. Scenario-based Evaluation

In this section the functional requirements are evaluated by applying a scenario-based approach. For this purpose use case scenarios are predefined and presented to the interested companies. Representatives of 2 enterprises participated in this evaluation.

- ***Scenario 1**: Generation of a report, showing the relationships of applications in an enterprise*
  In this scenario a single report is generated. This simulates a management report by showing the relationships and dependencies of applications in the company. Thus aggregated attributes should be used and the most important applications should be highlighted.
- ***Scenario 2**: Generation of a series of technical documentations of all applications in an enterprise*
  In this scenario, technical documentations are created. Therefore all attributes of an application should be formatted and outlined in a properly style. For every single application an own documentation should be created.

While presenting the stated scenarios, a questionnaire was distributed to quantify and qualify the fulfillment of the functional requirements. Thus each requirement was assessed through an absolute quantifier between 1 and 5, whereby 5 stated the agreement for an entirely successful implementation of the requirement. This quantifiers, however, give only indicators for the fulfillment of the requirements, as the amount of participants is not representative. More interestingly, for each requirements informal comments, suggestions, further requirements for productive application, concerns, and problems were discussed. The questionnaire can be inspected in Appendix B. Following, for each requirement the essential results of the evaluation are pointed out. At the end the general impression of the participants are expounded.

- ***Integration of visualizations***
  This requirement was not implemented in the prototype, since it was only low prioritized. Therefore this requirement was almost omitted in the discussion. However, for deployment in productive environment, the participants stated, that an integration of visualizations are important. Especially charts and diagrams are necessary for properly reports.
- ***Document versioning mechanism***
  This requirement was assessed with 3.5 points by the participants. Therefore the

realization can be considered as averagely fulfilled. The participants coincide, that a version-history, contained in the documents are not necessary by following the approach of re-creating documents with actual data. Nevertheless, in the created application the created reports were only versioned by the name of the report-entity. The enterprises, however, demanded concrete version-numbers. Thus the created documents should be extended with a versioning attribute, which monotonically increases for each created report. As the SocioCortex-system already provides an versioning mechanism, the idea came up to not create a new report each time, but rather replace the former one. The versioning-mechanism of SocioCortex will archive the previous versions and therefore they are still available in future. As for now, the reports are stored in a central repository. This also allows the creation of reports about the reports itself. While the participants were satisfied with this approach, they missed the opportunity to store the created reports directly to the entity, for which the report is created. This feature, combined with a new versioning-mechanism would massively enhance the utility of the application, as the end-user, requiring reports or documentations of specific entities, could always accesses the last version in an easy and efficient way.

- ***Publishing workflow***
  As this feature was prioritized as low, due to the scope of this thesis, it was not implemented. Nonetheless, for productive application, such a workflow should be available. The participants stated, that many persons are involved in the creation of reports or documentations, as oftentimes specific regulatory requirements have to be complied. Thus the definition of different roles and their access rights are needed. Therefore specific persons should be allowed to release the reports, while others are only allowed to download final reports.

- ***Reuse of sub-components***
  This general formulated requirement was divided up in the discussion into several points of aspects. The first aspect described the reuse of sub-components of different created reports or templates. In the followed approach, by generating only the whole report from components of SocioCortex, it was quantified with 1 point; therefore not implemented. Thus it is not possible to reuse existing documents for extracting information or even merge different documents to a comprehensive report.
  The second aspect addresses the reuse of the template to generate different reports with the same formatting and also the same text-components. This was considered as very useful and expedient (5 points).
  The third aspect was the reuse of components of SocioCortex. Supported in this application are the usage of mainly attributes of entities. By defining queries

and linking them to parameters, the attributes are reused in different kinds of reports. This approach were suffice to the requirements of the participants and was assessed with 4 points. However, the integration of SocioCortex-text (with enumerations, sections, etc.), which is defined via HTML, is not supported properly. For this purpose an intermediate step of parsing those HTML sections for the usage in text files is necessary.

- ***Interface to present data sources***
  The question focuses one the interaction with SocioCortex. Thus if SocioCortex is properly used to retrieve the data, and furthermore, if the generated artifacts are stored in a proper way.
  This requirement was assessed with 4 points. The retrieval of information was satisfactory. Thus the usage of MxL, a powerful query language to retrieve the information, was utilized. Also the connection of the definition of types in the template, with the check for the right type in the queries was considered as helpful. However, as already briefly mentioned, storing the artifacts in a central repository is indeed useful, but there are use cases, where the reports should be stored in the corresponding entity / entityType. With fulfilling this additional requirement, the integration within SocioCortex would be assessed as excellent.

- ***Integration of different file formats***
  This requirement was assessed with 4 points. The supported formats of .pptx and .docx for templates and the additional .pdf-format for reports are the main application areas and therefore expediently implemented. Nevertheless reason for not completely satisfying the participants was the missing format of Microsoft Excel. The generation of Excel-reports is crucial, as it allows complex further processing. The participants stated, that they often need further calculations to finalize a report.
  Furthermore the converting mechanism to .pdf struggles with complex formatting; thus reports, depended on the template, are skewed and warped. For this purpose, of course, a faultless support of this format is requested.

- ***Usage of parameters and types***
  The application of the template-language velocity and the application of parameter types was considered as excellent (5 points). Not only the integration of conditions, loops, and sequences were expedient, but also the easy-to-use syntax and application in the word-environment. Thus no additional macros or plug-ins are needed. The definition of the parameters type directly in the template, with the in subsection 5.1.1 defined notation, was considered as good. Therefore no additional parameter-types have to be supported in practice.

- ***Serial generation of documents***
  The serial generation of reports and documentations was considered as fully implemented. Thus 5 points were coincidentally assessed by the participants. Especially the definition of entity-based queries were emphasized. Additionally the fact, that all reports had the same structure and formatting was beneficial. One participants noticed, that the dynamic definition of the series of entities might be useful.

- ***Integration of formatting functions***
  This requirement was assessed with 3 points. Even the approach of using desktop-based tools was considered as very well, some limitations are available, mainly through the usage of parameters. While it is possible to format the usage of simple parameters, the full functional-spectrum of the tools of Microsoft cannot be used with the definition of sequences. Thus it is not possible to define sophisticated tables. Additionally the usage of the diagrams provided within these tools is not applicable with information inserted by the parameters.
  Another aspect was the missing integration of the HTML-components of Socio-Cortex. For this purpose further development has to be done.

Summarizing the participants perceived the application as very useful and expedient. The usage of this application simplifies the creation of documentations and furthermore avoids errors. The presentation demonstrated successfully how a report-generator within the SocioCortex environment might look like. However, for productive usage some parts are still missing and further development is inevitable.

As the scenario-based evaluation only concentrated on the functional requirements, the non-functional requirement are also important to evaluate. For this purpose an informed argumentation is provided in the next section.

6.2. Informed Argumentation

For the evaluation of the non-functional requirements the method of *informed argumentation* is utilized. This follows the design science research paradigm by Hevner, by conducting a descriptive evaluation (Hevner, March, Park, and Ram 2004). For this purpose convincing arguments are provided, to critically reflect the realization of the requirements, outlined in section 3.2.
For the external non-functional requirements following arguments can be stated.

- ***Availability***
  The requirement of availability is hard to measure, while not deploying the application in a productive environment, as it depends on the used servers, the internet connection, and other environmental influences. However, by applying properly exception handling, both in the client- and server-side, the application is prevented from crashes. Furthermore the input to the server is checked, when calling REST-API operations, and proper failure-messages are returned. This also prevents the collapse of the system. Therefore it can be assumed, that the application is very robust and should have high availability, when deploying it in a proper environment.

- ***Integrity***
  By applying different kinds of checks and validations, the integrity constraint is fulfilled. Thus all input fields in the client-side are validated. Therefore data inaccuracy is prevented. For the file-inputs, however, only the right format is checked. While uploading the template-file, no checks for correct usage of the template-language is done. For this reason, an additional step should be implemented, testing the template-files for correct syntax, or another approach might be the implementation of a word plug-in, checking and supporting the definition of parameters.

- ***Interoperability***
  For the scope of this thesis, the application only had to connect with the SocioCortex system. To fulfill the requirement of interoperability, different design decisions were made. Therefore the server was constructed and implemented with the microservice architecture. Furthermore it does not involve any connection specific operations (e.g. to SocioCortex). Additionally by providing a REST-API, other application can easily access the server's functionality for their own purposes. However, the client is more specified to access only SocioCortex. So by connecting the system with other systems, several parts have to be changed and adapted.

For this reason, among others, especially modifiability was focused, which is described later.

- ***Performance***
  The performance is also a requirement which is hard to determine, since it also depends on the environment. While the system itself behaved rather slow on the development environment, tests on different computer showed, that the application runs very smoothly. This behavior was as expected, since no complex tasks (NP-complete or higher order run-time algorithms) were used at all. Also the server- and SocioCortex-calls, where tried to execute in parallel. An exception to this is the execution of a serial job. For implementation reasons these server calls were synchronized. This leads to a delay in the process. In the future a more talented developer might replace this behavior to a more sophisticated version.

- ***Security***
  For security reasons, an authorization mechanism is provided. No other endeavors were contrived for this requirement, as it would exceed the scope of this thesis. For application at an enterprises, which handle sensitive data and where all applications have to confirm with security polices, of course more security measures have to be implemented: Examples are the encryption of data or the prevention against hacker attacks.

- ***Usability***
  For this external measure, mainly the client-side is liable. Different efforts are implemented to increase the usability for the user. So first, in case of wrong inputs or miscellaneous failures, feedback messages are provided. Also loading-bars and other indications are implemented to show the user that an operation is in progress. For the design of the website, the front-end framework bootstrap was used. It provides sleek and intuitive web-components which also decrease the effort of development (Bootstrap 2016).

For the internal non-functional requirements following arguments can be stated.

- ***Scalability***
  By using a scalable framework for the server-side development in combination with the architecture of microservices, the server-side should be very scalable. Thus the services can be distributed among several processors and by the stateless implementation, all server-calls can be processed in parallel. However, this procedure is not tested and therefore only of theoretical nature. The application does not store any data and therefore a growing amount of users, will not change the situation, yet more processors should be deployed.

- ***Modifiability***
  While building a prototype, the modifiability-requirement was especially emphasized, since subsequent developers might take this application for the basis of new systems, or even further develop the application itself. Therefore different aspects were respected. The first aspect is the general structure of the application. By using the microservice architecture, the programming code is well separated and encapsulated. Thus it becomes easy to change and maintain a specific functionality. A similar approach was strived in the structure of the client-side, so the different steps to the creation of a final report were separated into encapsulated packages of code. Here even a step further was taken, by implementing separate controller for each coherent package of functionality. Each client-side controller therefore is only responsible for a manageable amount of code. By only considering the name of the controller, the functionality it contains, should become clear. While not only implementing in this structured way, also detailed models were provided, which can serve as anchor to get an overview and to understand the structure of this application. Here a de facto standard notation for modeling software architectures was utilized.
  A further endeavor is the extensive documentation of the programming code. So for the server-side, which was implemented with the aid of java, the notation of Javadoc was used. As a consequence it is possible to generate a HTML based documentation of the functionality. More information about Javadoc is provided at the website of the Java documentation (Oracle America 2016a). Also the client-side was well documented by providing comments and explanations for every available function. Of course the programming code was formatted properly and also the variables and functions were labeled according to their functionality and usage. Ultimately a readme-file, consisting of brief instructions, was created to get a fast access to the application.

The last step of evaluating the application was accomplished by showing the strengths and weaknesses. This can serve as an anchor for further developments. To complete this thesis a conclusion is provided in the next chapter.

7. Conclusion

In the evaluation chapter, the strengths and weaknesses of the application were discussed. This was the final step of the design science research paradigm. However, to finalize this thesis section 7.1 will summarize the done work. Hereafter the application as well as the thesis approach will be critically examined. Ultimately an outlook proposes ideas for further research and development.

7.1. Summary

The main task of this thesis was the analysis of requirements from practice to implement a suitable report generating tool in the context of Enterprise Architecture Management. For this reason the topic was motivated, the problems derived, the objective stated and to frame the proceeding, the design science paradigm was introduced. It describes the procedure to develop and evaluate artifacts in information systems research. The input hereto is derived from practical business cases and from theoretical foundations.
Subsequently the subject area was defined more accurately. Thus the tasks and goals of Enterprise Architecture Management was defined by carving out the essential aspects from academia.
Even though the topic of the Documentation of the Enterprise Architecture consists, among other points, also of the collection of data, this field was only partially treated. By using the system SocioCortex as data provider, this confinement could be attenuated. SocioCortex generates and collects the data, applying a collaborative Hybrid-Wiki approach.
In the next step, functional requirements were collected by conducting interviews with Enterprise Architects from different enterprises. These were abstracted to derive more general requirements for the goal of providing a more general solution. Also, based on the statements of the participants and to comply with the scope of this thesis each requirement was prioritized. Beside the functional requirements, also non-functional requirements for corporate and internet applications were identified through literature research.
Based on these requirements a solution was constructed. For this reason, the modeling-language UML was utilized. It specifies different types of diagrams to illustrate the behavior and structure of applications. UML can be considered as industrial standard. The provided diagrams not only facilitate the realization in programming code, but

also provide an overview for further development or maintenance. The use cases of the application and their relationship to the outer environment was described with the aid of the use case diagram. The activity diagrams depict the procedure on the client-side and a derivation of a class diagram was utilized to constitute the server-side. The structure of the server was inspired by the microservice architecture. Thus functionality were encapsulated to fulfill the requirement of scalability and modifiability. Finally a sequence-diagram described the communication between the different components of the application.
This models were subsequently implemented in programming code. Different external resources were used for this purpose. These enabled and facilitated the implementation of such a reporting-tool. Thus the template-language Velocity was applied to create template-files. To generate the final reports the Java-library XDocReport was used and for the access to SocioCortex resources the corresponding REST-API were utilized. Additionally the Model-based expression Language (MxL) was introduced to formulate queries to access content and attributes of the SocioCortex-system.
Each step of the implementation was briefly described. Hereto the implementation followed the previously produced models. Thus in the client-side the authentication, the creation of template, the creation of configurations, and the creation of final reports was described with the aid of figurative representations and by providing small and declarative code-snippets.
The server was developed with the application of the Play! framework. It allows the stateless and scalable development of web applications. However, it was only used within the server-development. Furthermore it supports the Java-programming language. This was required as the XDocReport library is purely Java-based. In principle the server fulfills two functions. First the extraction of parameters from template-files and second the generation of reports, by exchanging parameters with concrete values. Supported formats are Microsoft Word and Microsoft PowerPoint files.
In the last step, the application was presented to interested enterprises, by performing predefined scenarios. With the aid of a questionnaire and by extensive discussion an evaluation could be conducted. Consequently the strengths and weaknesses of the application could be determined. Also required functionality for productive application could be derived in this conversations.
To evaluate the non-functional requirements, the method of informed argumentation was utilized. It is an evaluation method, proposed in the design science paradigm, to assess the quality with factual and convincing arguments.

7.2. Critical Appraisal

This section provides a critical appraisal of the done work. For this purpose, not only the limitations of the outcome (the implemented application) are reflected, but also the approach and procedure of this thesis is illuminated.

7.2.1. Limitations

While the weaknesses of the application were already discussed in chapter 6, here a more generally view should be provided by inspecting the limitations of the implemented prototype. Thus the perspective of an entrepreneur is applied, to filter out the missing criteria to apply the system in real enterprises.
One critical limitation is the creation of more sophisticated reports, since the creation of reports is the main functionality of the application. Especially for reports on the management level, crucial components, for instance visualizations and diagrams, are missing. It is neither possible to create diagrams out of the SocioCortex systems nor to create word-integrated diagrams, filled with data of SocioCortex. Also by converting the documents, many components, provided by the Office tools are skewed. Thus a more elaborated converting function has to be implemented. For this implementation an open-source converter was applied. For the application in enterprises a propriety solution might be more convenient.
For the employment in real enterprises all missing requirements should be fulfilled. Additionally further requirements arose in the evaluation discussion, which can also be classified as crucial. Thus a workflow should be implemented to release documentations and reports. In this context also roles and rights needs to be defined. As mentioned, visualizations should be integrated properly. Also the versioning-mechanism needs to be enhanced. A necessary requirement, stated in the evaluation discussion, is the integration of Microsoft Excel files, as they are extensively used for further calculations. Through this, also the usage of the Office diagrams might be possible, as they use the Excel-format as data basis. By implementing the support of the Excel-format, it is suggested to implement this functionality direct within the XDocReport project. The adaption of the application to this new format should be straightforward.
A more technical limitations is the intricateness of defining parameters within the application, as not all parameters can be extracted properly from the template-file. This laborious process may also lead to erroneous input. A more sophisticated algorithm may completely extract the parameters with the corresponding type. Especially the identification of sequences, with the immanent attributes is an important challenge. This feature would simplify the step of the creation of a template significantly.

7.2.2. Thesis Approach

In this section the applied procedure and approach of this these is critically analyzed. While the general paradigm of design science research provided a foundation in the procedure of this theses, different aspects could have been enhanced to not only increase the additions to the knowledge base, but also to provide a better solution for specific business-cases. First the collection of functional requirements is regarded. Although the collected requirements were abstracted to formulate more general requirements, they cannot be considered to be representative. Thus other enterprises may have other problem areas. An elaborated approach would be to interview more companies, for instance through applying a representative survey. Another approach might be the focus on only one specific environment, and to concentrate on specific use- and business- cases. In doing so, more profound content could be added to the knowledge base. Also coordination problems could have been avoided. As the reader might may already asked himself, why four participants contributed their requirements and only two companies evaluated the finished prototype.
The problem of representative results can also be transferred to the evaluation step. Thus it cannot be stated that the implemented application solves the problem of creating documentation for all other companies, although the participants remarked the potential of the application.

Also the modeling and architecture process could be enhanced by a more iterative approach. Therefore meetings and feedback sessions should be conducted. Even if concrete requirements were collected, they left a big margin, which on the one-hand left many freedoms for the development, but on the other hand the requirements could not be fully comply with the different views of the companies. In feedback sessions, the models could be adapted and improved in mutual accordance. In doing so also the problem of late arising requirements could be prevented. For example, the implementation of a serial job occurred in a midterm review. As the process of implementing was still in the early phase, this functionality could be implemented. In contrast, the requirement of supporting the Excel-format arose in the evaluation meeting. Of course the implementation was finished, and due to strict submission dates and a defined scope of this theses, this requirements could not been implemented. Also requirements as the enhanced versioning mechanism, could easily been implemented, if it came up in an iterative approach. This approach migh orient, for instance, on the agile-framework Scrum, by implementing only small artifacts in sprints.

7.3. Outlook

The limitations of the application were identified. Due to the prototypical implementation, arising restrictions and open issues were expected. However, the application, presented at real enterprises, turned out to be very appealing. It showed the existing interest of the participants and thus proved the potential of this approach of generating documentations.
Assuming the fully implementation of the missing requirements, different directions are conceivable for further utilization. Following some of these are outlined, which arose both in the course of this thesis and in the conversation with the participating enterprises. It should be remarked, that these are only suggestions and in the context of other purposes different ideas might come up.
The most target-aimed aspect, would be a more elaborated integration within SocioCortex. For example, template-files might be assigned to entities. As the they have the same structure, this could take place on the entityType level. Thus for every entityType, consisting of many entities, only one template-file will be generated, representing the text-output of an entity. In this context the template-file would serve as a kind of style-sheet. If the end user now, would like to export an entity to a concrete file, the report generating step is executed and the user would always get access to formatted, consistent, and coherent documents. This functionality would not only enhance the powerfulness of SocioCortex, but also increase the usability for the end-user.
A related approach would be the central generation of reports, but the single created reports are not stored centrally but rather assigned to the different entities. Thus the report-generator would be only accessible for designated groups of persons, for example Enterprise Architects or the compliance department of an enterprise.
A functionality which would increase the utility of the application to a large extent is an automatic update of generated reports, when data changes. This functionality, combined with the above mentioned ideas and a sophisticated versioning-mechanism would create a very valuable and powerful tool to manage documentations and reports in a company.
The success of this application will be revealed in the future, as it depends on the commitment of other developers and the extent of the support from interested enterprises. This thesis, however, laid the foundation for further developments and paved the way to not only create serious solutions preventing the problems of today's documentation procedures, but also to enrich this domain with novel and innovative approaches and ideas.

Bibliography

Apache FreeMarker (2016). URL: `http://freemarker.org/` (visited on 09/14/2016).

Apache Software Foundation (2016). *Velocity*. URL: `http://velocity.apache.org/` (visited on 09/14/2016).

Bernard, S. A. (2012). *An Introduction to Enterprise Architecture*. Third Edition. AuthorHouse. ISBN: 9781477258019.

Bootstrap (2016). *getbootstrap.com*. URL: `http://getbootstrap.com/` (visited on 09/14/2016).

Branas, R. (2014). *AngularJS Essentials*. Packt Publishing Ltd.

Büchner, T., F. Matthes, and C. Neubert (2009). "A Concept and Service based Analysis of Commercial and Open Source Enterprise 2.0 Tools." In: *KMIS*, pp. 37–45.

Buckl, S., T. Dierl, F. Matthes, R. Ramacher, and C. M. Schweda (2008). "Current and future tool support for ea management." In: *Proceedings of Workshop MDD, SOA und IT-Management (MSI 2008), Oldenburg*, pp. 9–24.

Buckl, S., T. Dierl, F. Matthes, and C. M. Schweda (2010). "Building blocks for enterprise architecture management solutions." In: *Working Conference on Practice-Driven Research on Enterprise Transformation*. Springer, pp. 17–46.

Buckl, S., F. Matthes, C. Neubert, and C. M. Schweda (2009). "A wiki-based approach to enterprise architecture documentation and analysis." In: *17th European Conference on Information Systems (ECIS2009)*, pp. 1476–1487.

Buckl, S. and C. M. Schweda (2011). *On the state-of-the-art in enterprise architecture management literature*.

Buschle, M., H. Holm, T. Sommestad, M. Ekstedt, and K. Shahzad (2012). "A Tool for Automatic Enterprise Architecture Modeling." In: *IS Olympics: Information Systems in a Diverse World: CAiSE Forum 2011, London, UK, June 20-24, 2011, Selected Extended Papers*. Ed. by S. Nurcan. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–15. ISBN: 978-3-642-29749-6. DOI: `10.1007/978-3-642-29749-6_1`.

Chan, Y. E. (2002). "Why haven't we mastered alignment? The importance of the informal organization structure." In: *MIS Quarterly Executive*, pp. 97–112.

Chan, Y. E., S. L. Huff, D. W. Barclay, and D. G. Copeland (1997). "Business Strategic Orientation, Information Systems Strategic Orientation, and Strategic Alignment." In: *Info. Sys. Research* 8 (2). jun, pp. 125–150. ISSN: 1526-5536. DOI: `10.1287/isre.8.2.125`.

Diefenthaler, P. and B. Bauer (2014). "Using gap analysis to support feedback loops for enterprise architecture management." In: *MKWI*.

Farwick, M., R. Breu, M. Hauder, S. Roth, and F. Matthes (2013). "Enterprise architecture documentation: Empirical analysis of information sources for automation." In: *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. IEEE, pp. 3868–3877.

Fowler, M. (2004). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

Grunow, S., F. Matthes, and S. Roth (2013). "Towards automated enterprise architecture documentation: Data quality aspects of SAP PI." In: *Advances in Databases and Information Systems*. Springer, pp. 103–113.

Hanschke, I. (2009). *Strategic IT management: a toolkit for enterprise architecture management*. Springer Science & Business Media.

Hauder, M., F. Matthes, and S. Roth (2012). "Challenges for automated enterprise architecture documentation." In: *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation*. Springer, pp. 21–39.

Hevner, A. R., S. T. March, J. Park, and S. Ram (2004). "Design Science in Information Systems Research." In: *MIS Q* 28 (1). mar, pp. 75–105. ISSN: 0276-7783.

"ISO/IEC/IEEE Draft Standard for Systems and Software Engineering – Architectural Description" (2010). In: *ISO/IEC/IEEE P42010_D8, June 2010*. July, pp. 1–53.

iteratec GmbH (2016). *Iteraplan*. URL: `www.iteraplan.de/en/` (visited on 09/14/2016).

Langenberg, K. and A. Wegmann (2004). *Enterprise architecture: What aspects is current research targeting*.

Luftman, J., R. Kempaiah, and E. Nash (2006). "Key issues for IT executives 2005." In: *MIS Quarterly Executive* 5 (2).

M. Farwick, B. Agreiter, R. Breu, M. Häring, K. Voges, and I. Hanschke (2010). "Towards Living Landscape Models: Automated Integration of Infrastructure Cloud in Enterprise Architecture Management." In: *2010 IEEE 3rd International Conference on Cloud Computing*. July, pp. 35–42. DOI: `10.1109/CLOUD.2010.20`.

Masse, M. (2011). *REST API design rulebook*. O'Reilly Media, Inc.

Matthes, F., S. Buckl, J. Leitel, and C. M. Schweda (2008). *Enterprise architecture management tool survey 2008*. Technische Universität München.

Matthes, F., C. Neubert, and A. Steinhoff (2011). "Hybrid Wikis: Empowering Users to Collaboratively Structure Information." In: *ICSOFT (1)* 11, pp. 250–259.

McDavid, D. (2003). "The business-IT gap: A key challenge." In: *IBM Research Memo.*

Microsoft Office-Support (2016). *Insert and format field codes in Word 2010*. URL: `https://support.office.com/en-us/article/Insert-and-format-field-codes-in-Word-2010-7e9ea3b4-83ec-4203-9e66-4efc027f2cf3` (visited on 09/14/2016).

Newman, S. (2015). *Building Microservices. Designing Fine-Grained Systems*. O'Reilly Media, Inc.

Niemann, K. D. (2006). *From enterprise architecture to IT governance*. Vol. 1. Springer.

Object Management Group (2015). *OMG Unified Modeling Language (version 2.5)*. Version formal/2015-03-01.

Object Management Group, I. (2016). *OMG. Object management Group*. URL: `http://www.omg.org/` (visited on 09/14/2016).

opensagres (2016). *XDocReport (MIT license)*. URL: `https://github.com/opensagres/xdocreport` (visited on 09/14/2016).

Oracle America, I. (2016a). *Javadoc SE Documentation - Javadoc*. URL: `http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html` (visited on 09/14/2016).

– (2016b). *Specification: java.util.regex*. URL: `https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html` (visited on 09/14/2016).

O'reilly, T. (2007). "What is Web 2.0: Design patterns and business models for the next generation of software." In: *Communications & strategies* (1), pp. 17–37.

Play! (2016). *Play Framework*. URL: `https://www.playframework.com/` (visited on 09/14/2016).

Reschenhofer, T., M. Bhat, A. Hernandez-Mendez, and F. Matthes (2016). "Lessons learned in aligning data and model evolution in collaborative information systems." In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, pp. 132–141.

Reschenhofer, T., I. Monahov, and F. Matthes (2014). "Type-Safety in EA Model Analysis." In: *EDOC Workshops*, pp. 87–94.

Richardson, R. and C. S. Director (2008). "CSI computer crime and security survey." In: *Computer Security Institute* 1, pp. 1–30.

Roth, S., M. Hauder, M. Farwick, R. Breu, and F. Matthes (2013). "Enterprise Architecture Documentation: Current Practices and Future Directions." In: *Wirtschaftsinformatik*, p. 58.

S. Murugesan (2007). "Understanding Web 2.0." In: *IT Professional* 9 (4). July, pp. 34–41. ISSN: 1520-9202. DOI: `10.1109/MITP.2007.78`.

SEBIS - GitHub (2016). *GitHub Repository*. URL: `https://github.com/sebischair/` (visited on 09/14/2016).

SocioCortex - A Social Information Hub (2016). Software. URL: `https://wwwmatthes.in.tum.de/pages/13uzffgwlh8z4/SocioCortex` (visited on 09/14/2016).

Software Engineering for Business Information Systems (2016a). *SocioCortex Community Workshop Series*. URL: `https://wwwmatthes.in.tum.de/pages/z404x99bysf0/SocioCortex-Community-Workshop-Series` (visited on 09/14/2016).

– (2016b). *SocioCortex - Documentation*. URL: `http://www.sociocortex.com/documentation/` (visited on 09/14/2016).

– (2016c). *SocioCortex Tutorial*. URL: `http://www.sociocortex.com/tutorial/` (visited on 09/14/2016).

– (2016d). *SocioCortex - Visualizer*. URL: `http://visualizer.sociocortex.com` (visited on 09/14/2016).

The W. Edwards Deming Institute (2016). URL: `https://www.deming.org/` (visited on 09/14/2016).

W. J. Dzidek, E. Arisholm, and L. C. Briand (2008). "A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance." In: *IEEE Transactions on Software Engineering* 34 (3). May, pp. 407–432. ISSN: 0098-5589. DOI: `10.1109/TSE.2008.15`.

Weill, P. and J. W. Ross (2009). *IT Savvy: What top executives must know to go from pain to gain*. Harvard Business Press.

Wiegers, K. and J. Beatty (2013). *Software requirements*. 3rd ed. Pearson Education.

Wikimedia Foundation (2016). *Wikipedia - The free Encyclopedia*. URL: `https://www.wikipedia.org/` (visited on 09/14/2016).

Winter, V. L. and S. Bhattacharya (2012). *High Integrity Software*. Vol. 577. Springer Science & Business Media.

Appendices

A. List of Requirements

ID	Requirement
R-1	Connection with different data sources (PlanningIT, Wikis, SocioCortex)
R-2	Combine text parts to whole documents
R-3	Use wikipage as data source
R-4	Export document to .pdf
R-5	Including of .jpeg files in the document
R-6	Approval work flow
R-7	Version-mechanism: Differentiate between old and new versions
R-8	Inclusion of simple parameters
R-9	Inclusion of plain text
R-10	Inclusion of lists
R-11	Separation of Content and Design
R-12	Merging of documents
R-13	Allow printing of whole wikipages
R-14	Formatting functions (compared to Office Word): fonts, section breaks, etc.
R-15	Locking up created documents
R-16	Availability to further process the documents
R-17	Generation of pictures
R-18	Generation presentation files (e.g. .pptx)
R-19	Integration of different file systems (e.g. Dropbox, Owncloud)
R-20	Annotation of document versions
R-21	Definition of attribute categories
R-22	Creation of a document for each entity in an entityType at once
R-23	Integration of pictures and diagrams from entities
R-24	Archiving of old documents

B. Evaluation Questionnaire

1. ***Reuse of sub-components***
   Sub-components can be reused appropriately in different documents. Therefore a laborious process to amend this documents is prevented.

2. ***Usage of parameters and types***
   Both simple and complex parameters are used suitably and sufficiently to generate reports and documentations.

3. ***Interface to present data sources (SocioCortex)***
   The data are suitably extracted from the system. Thus all necessary data to create documentations can be loaded from the system. In addition, created reports are stored appropriately in the system.

4. ***Integration of formatting functions (excluding visualizations)***
   The formatting functionality to create documentations is in line with expectations. Thus no formatting functionality to generate final documentations is missing.

5. ***Serial generation of documents***
   The functionality for serial processing of documents is sufficient. It will not only save time, but also errors are avoided.

6. ***Integration of different file formats***
   The supported formats are sufficient for documentation and reports.

7. ***Document versioning mechanism***
   The versioning of documentation is sufficient. By the procedure no classical histories are needed.