

FAKULTÄT FÜR INFORMATIK

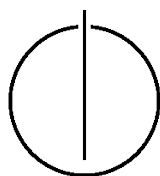
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Generation of Recommendations for Process Templates
Based on the Analysis of Change Histories

Generierung von Vorschlägen für Prozesstemplates
basierend auf der Analyse von Änderungshistorien

Author: Sebastian Fröhlich
Supervisor: Prof. Dr. Florian Matthes
Advisor: Felix Michel
Date: December 15, 2015



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, December 15, 2015

Sebastian Fröhlich

Acknowledgments

I want to thank everyone who supported me during this thesis. First and foremost, I thank my advisor Felix Michel for his knowledge and valuable ideas, which helped me to complete this work. In addition, I want to thank the whole team of the SocioCortex project to get an overview about the project during the biweekly team meetings.

Further I would like to thank my family and my girlfriend for their moral support.

Abstract

Ad hoc processes have become a big role in modern organizations. Nevertheless, traditional workflow solutions have problems to support those processes. As a result, there is the trend moving away from those inflexible processes to collaborative solutions. Here contributors are able to design or customize existing steps for different activities, with their domain specific knowledge. Here the problem arises, that these flexible and maybe unstructured systems lead to a higher effort for recurrent processes, because every step has to be rethought and created repeatedly for similar cases. Additionally, unskilled users may be overstrained, because of the lack of guidance. Then, experienced users with their domain specific knowledge are needed to complete a process successfully. However, these experienced users are often not involved in the modeling part of those processes, where templates are made.

In this work concepts are developed to help users with no or less domain knowledge to handle the modeling part of those processes successfully. Therefore, the implicit knowledge of the change histories of different cases is used to give recommendations. In a first step, the entire change history of the *SocioCortex* server is analyzed for its features. Afterwards a concept is developed, to prepare the data in a way to make it usable for the recommender. Out of these prepared data, a recommender is propagated to help users to create the task related templates.

Finally, a prototypical implementation of these concept is done within the *SocioCortex* eco system, to evaluate them.

Contents

Acknowledgements	v
Abstract	vii
Outline of the Thesis	xix
I. Introduction	1
1. Introduction	3
1.1. Motivation	3
1.2. Problem Description	4
1.3. Objective	5
II. Background and Related Work	7
2. Recommender Systems and Datamining	9
2.1. Basic Definitions	9
2.2. The Data Mining Process	10
2.2.1. CRISP-DM	11
2.2.2. Other Relevant Models	13
3. Frequent Pattern Mining	15
3.1. Description	15
3.2. Models and Definitions	16
3.2.1. Itemset Mining	17
3.2.2. Association Rule Mining	19
3.2.3. Generalization	20
3.3. Algorithms	20
3.3.1. Brute Force	20
3.3.2. Apriori	22
4. Process Mining	25
5. Related Work	27
5.1. Frequent Pattern Mining	27
5.2. Process Mining	28
5.3. Other Relevant Work	29

III. Conceptual Approach	31
6. Approach	33
6.1. Business Understanding	34
6.1.1. Create New Task Definition with Attributes (Use Case 1)	35
6.1.2. Remove Existing Task Definition (Use Case 2)	36
6.1.3. Add Recommended Attribute to Existing Task Definition (Use Case 3)	37
6.1.4. Remove Attribute from Task Definition (Use Case 4)	38
6.2. Data Understanding	38
6.2.1. Class Model	39
6.2.2. Supported Attributes	40
6.2.3. Supported Pages	42
6.2.4. Analysis of Change Histories	43
6.3. Data Preparation	46
6.3.1. Select	47
6.3.2. Clean Data	49
6.3.3. Construct Data	50
6.4. Modeling	53
6.4.1. Supported Attribute Sets	53
6.4.2. Score	53
6.4.3. Group Recommendations	55
6.4.4. Individual Score	56
IV. Technical Approach	59
7. Front End	61
7.1. The Client in the Context of the SocioCortex Eco System	61
7.2. User Interface	62
7.2.1. Create new Task Definition with Attributes (Use Case 1)	63
7.2.2. Add Recommended Attribute to Existing Task Definition (Use Case 3)	64
7.2.3. Remove Attribute from Task Definition (Use Case 4)	65
7.2.4. Remove Existing Task Definition (Use Case 2)	65
7.3. Technical Structure	66
7.3.1. File Structure	66
7.3.2. Controller	66
8. Backend	69
8.1. Structure	69
8.2. BasicRecommendationUtil	71
8.3. FPMUtil	72
8.4. Used Parameters	73
9. Evaluation	75
9.1. Usefulness of Results	75
9.2. Scalability	78

9.3. Impact of Parameters	81
V. Conclusion and Outlook	83
10. Conclusion and Outlook	85
10.1. Summary	85
10.2. Outlook	87
Appendix	91
A. Appendix	91
Bibliography	95

List of Figures

2.1. CRISP-DM Model	12
3.1. Lattice of itemsets	18
3.2. Execution lattice of Apriori algorithm	24
4.1. Example process in BPMN notation	26
6.1. Basic steps of the approach as part of the CRISP-DM model	33
6.2. Conceptual class diagram of entire system	39
6.3. Completeness of pages	43
6.4. Time differences between subsequent changesets	45
6.5. Example of group creation	52
7.1. The SocioCortex eco system	61
7.2. Overview Frontend	62
7.3. Recommended TaskDefinition with attributes	63
7.4. Existing TaskDefinition with recommendations	64
7.5. Remove of an existing TaskDefinition	65
8.1. Process in Backend	70
9.1. Existing TaskDefinition with recommendations	79

List of Tables

3.1. Transaction Example	16
3.2. Transaction example in transactional data format	17
3.3. Transaction example in tabular data format	17
3.4. Example for quantitative data	20
3.5. Support of all itemsets	21
3.6. Calculated association rules	21
4.1. Example of an event log	25
6.1. Use Case 1: Create new task definition with attributes	35
6.2. Use Case 2: Remove existing task definition	36
6.3. Use Case 3: Add recommended attribute to existing task definition	37
6.4. Use Case 4: Remove attribute from task definition	38
6.5. Attribute usages of "Publication"	41
6.6. Attribute usages of "Article"	41
6.7. Simplified log example	44
6.8. Share of Changesets with single/multiple Changes	45
6.9. Cleaned log	50
8.1. Used parameters	73
9.1. Scope of available data	75
9.2. Recommendation without existing tasks	76
9.3. Recommendation based on existing tasks	77
9.4. Used Parameters for scalability tests	78
9.5. Recommended tasks (Part 1)	81
9.6. Recommended tasks (Part 2)	81
A.1. Time differences between two subsequent ChangeSets	91
A.2. Completeness of pages	92
A.3. Executing times of algorithm	93

Listings

3.1. Apriori algorithm	23
6.1. Find attribute occurrences	47
6.2. Find supported attributes	48
6.3. Filter pages	48
6.4. Group changesets to transactions	51
6.5. Construct transactions	52
6.6. Find minimum distance	54
6.7. Find group recommendations	55
6.8. Calculate scores	57
7.1. Add generic attribute to TaskDefinition	67
7.2. Create task definition with attributes	67
7.3. Structure of resulting recommendation	68
7.4. Structure of TaskRecommendations	68
7.5. Structure of RecommendedAttribute	68
8.1. Find groups	71

Outline of the Thesis

Part I: Introduction and Theory

CHAPTER 1: INTRODUCTION

The introduction states out the problem of template creation in collaborative system and briefly explains the approach.

Part II: Background and related work

CHAPTER 2: RECOMMENDER SYSTEMS AND DATAMINING

This chapter contains the basic information, which are relevant for this work. This contains elementary definitions and classifications in recommender systems as well as the data mining process itself.

CHAPTER 3: FREQUENT PATTERN MINING

In this chapter frequent pattern mining is introduced. This includes basic definitions and an exemplified application of the algorithms.

CHAPTER 4: PROCESS MINING

The fourth chapter covers the basics of process mining.

CHAPTER 5: RELATED WORK

The related work contains approaches from the area of frequent pattern mining as well as from process mining.

Part III: Conceptual Approach

CHAPTER 6: APPROACH

The sixth chapter contains basic use cases and the analysis of the data. In addition to that concepts to prepare the data and to model them are proposed.

Part IV: Technical Approach

CHAPTER 7: FRONTEND

This chapter illustrate the visible result of the shown concepts. Therefore, the user interface is described by the underlying use cases.

CHAPTER 8: BACKEND

Here the implementation of the proposed concepts and algorithms is briefly explained.

CHAPTER 9: EVALUATION

The evaluation chapter interprets the results by their usefulness and scalability.

Part V: Conclusion and Outlook

CHAPTER 10: CONCLUSION AND OUTLOOK

This chapter summarizes the thesis and gives a short outlook about possible future improvements.

Part I.

Introduction

1. Introduction

1.1. Motivation

In general, people make future decisions based on their experiences.

In collaborative processes this is the same. People work together and use their knowledge to work through a more or less flexible process. Here the problem arises, that expert knowledge is mostly restricted to their specific domain. In this case existing data could be used to improve that knowledge.

Nowadays in almost every area like economics, marketing or sales huge amounts of data is created. Thus [Toedt, 2014, p. 9] states that "Data is everywhere". The quantity of data is even enormous how [Gantz and Reinsel, 2010] indicate. The IDC study determines that in 2009 the digital universe had already grown to almost 0.8 zettabyte (1 zettabyte = 1 million petabytes). Moreover [Gantz and Reinsel, 2012] claim that this value will double every two years until 2020 and reach about 40 zettabytes then. About 30 years ago John Naisbitt already remarked that "We are drowning in information, but starving for knowledge" (translated from [Naisbitt, 1986, p. 41]). But what are the difficulties when using this huge amount of data to extract useful knowledge?

[van der Aalst, Wil M. P., 2011, p. 2] claimed that most of the data in the digital universe remains unstructured and companies have big problems to benefit from them. Another problem is a lack of tools, which makes the usage of this data difficult. Nevertheless, [Toedt, 2014, p. 9] implies that the usage of Big Data is an important factor for business success today. Therefore, the extraction of data has become a big challenge for companies in these days [van der Aalst, Wil M. P., 2011, p. 2].

At the same time the handling of ad hoc processes has become a big issue in modern organizations. Nevertheless, there are a lot of difficulties with those processes to be supported through traditional solutions, because of restrictions in workflow management systems [van der Aalst, W.M.P. et al., 2003a]. As a result, there is a big trend moving away from those inflexible process flows to collaborative technologies [Motahari-Nezhad and Bartolini, 2011]. Here contributors collaboratively design or customize steps with different activity sequences to handle various purposes.

On the contrary these flexible systems may force users to define specific attributes or tasks manually and repeatedly for similar cases [Motahari-Nezhad and Bartolini, 2011]. Apart from that unskilled workers may be overstrained, because of the lack of guidance. Then experienced users are needed to master processes successfully. However, many of those activities in different steps have been done by other users or in similar previous cases. Therefore, the gathering and sharing of activities from previously related and re-

solved cases can become very useful to handle cases effectively.

As a logical result these activities are increasingly recorded in form of process execution logs. These logs can either hold simple information about the activity within a timestamp, but they can furthermore also include information about the associated user, possible user groups and affiliated tasks. Afterwards that information can be used to detect regularities and patterns. [Schönig et al., 2012]

These patterns can be taken to create templates that contain tasks, attributes and documents which are typically required for the specific case. Reasonable templates can afterwards help users to increase:

- **Usability:** Especially unexperienced users have a good starting point for different cases they may have never done before.
- **Structuring:** Equal naming of tasks and attributes leads to a higher recognition and helps users to orientate in multiple cases because of the same basic structure.
- **Efficiency:** Users can reuse the predefined and established structure of the template with a single click.

1.2. Problem Description

The previously mentioned circumstance is that users may have problems with collaborative processes. Either they have never done such a process or they have no domain knowledge. This represents a huge problem in those processes. Here predefined templates with necessary tasks or steps might be helpful.

The creation of those templates needs domain knowledge, too. However, often only people from the respective business domains have this knowledge. This bears the problem that exactly those people are mostly not engaged in modeling tasks in these collaborative or knowledge intensive process platforms. This means that those modelers e.g. need to ask domain experts or to analyze process data manually, to get the required experience. This can lead to an enormous expenditure of time, because every case is different and maybe concerns various domains.

Therefore, it should be possible to extract knowledge about already (successfully) completed processes automatically and in standardized form to help modelers. This means the existing data should be analyzed and prepared to find patterns. Afterwards these patterns can support modelers to create such templates.

1.3. Objective

Taking the problems into consideration, there is the need to provide implicit knowledge about those processes from the change histories to the modelers of such platforms.

Therefore, the aim of this master's thesis is to help modelers of those systems to create templates with usually performed tasks. These can afterwards be used for normal users as starting point.

Hence the first step is to identify possible use cases and how recommendations can help professional modelers to create such common template skeletons. Consequently, the existing change history has to be analyzed to develop a concept how the data needs to be prepared and how to extract that knowledge. This analysis should contain all attributes and pages to find out which of them are reasonable to be taken into consideration. In addition, the change histories may need to be extended, if they are not able to extract these information.

Secondly these gathered activities of previous cases should be used to create possible patterns.

Lastly this thesis should explain how a prototypical implementation can be achieved. This contains the implementation of the concepts, as well as the presentation of the recommendations in an adequate and intuitive way. Afterwards an user should be able to detect attributes, which fit well together and to adapt the templates with these information.

Part II.

Background and Related work

2. Recommender Systems and Datamining

This chapter contains the basic information which are relevant to this work. The first part shows elementary definitions and classifications in recommender systems in common. The second part gives an overview about the data mining process. There the Cross-Industry Standard Process for Data Mining (CRISP-DM) model is introduced which serves as foundation for the approach.

2.1. Basic Definitions

A recommender system is a system which actively recommends useful elements for a user as subset out of a given entity set in a specific context. This context can e.g. consist on the user profile, the entity set and the situation. The user profile contains explicit (e.g. age, gender, topics) and implicit information like the amount of visits or what the user has done before. The aim of this process is to maximize the benefit, the user or the service providing company gain with the recommended items. [Klahold, 2009, p. 1]

One the one hand possible benefits for commercial sites can e.g. be an increasing number of sales or to sell different items, which can only be found hard. In common the benefit can also be an increase of user satisfaction and fidelity which can help to higher system usage and to increase the loyalty of users with the given system. Last but not least it can help to understand what the users want. [Ricci et al., 2011, p. 5]

Common goals for users on the other hand are e.g. finding good items or to improve their profile with the contribution of ratings [Herlocker et al., 2004].

The basis for the later recommendations is Data mining. Data mining itself has the aim to extract useful knowledge out of mass data like customer and organizational data or process data. This should be implicit knowledge which has not been known before. [Cleve and Lämmel, 2014, p. 2,38]

For a better understanding of recommender systems and data mining, first some basic terms are introduced:

Data Data is the aggregation of characters with their syntax. Data itself can be distinguished in unstructured data like text or images, semi structured data like text within the structure of a website and structured data like tables in a database or formats like comma-separated values (CSV). This structured data should have a fixed order and defined attributes with a specific data type. [Cleve and Lämmel, 2014, p. 37]

Information Information is data with a meaning. This means data turn to information if you use it in a specific context to get the meaning. [Cleve and Lämmel, 2014, p. 38]

Knowledge If you get the ability to use information this information can be labeled knowledge. [Cleve and Lämmel, 2014, p. 38]

User / User profiles Users or user profiles in the context of recommender systems can contain highly diverse properties and goals. These properties can range from simple lists how a user ranks an item to demographical information about e.g. age or gender up to complex behavior patterns like how he or she browses. Which information is used afterwards depends on the given recommender system. Collaborative filtering algorithms e.g. use a simple user-item ranking list for its recommendations. [Ricci et al., 2011, p. 8,9]

Item Items are the objects which can be recommended. Items can have diverse attributes like in films a genre or a list of actors. Normally an item is characterized by a value or a utility. This value can be positive, if the user benefit from it or negative if the item would be a bad choice for the user. In this context the cost for such recommended items should also be taken in consideration. The costs can be cognitive (for searching) or be real costs like paying money for an item. As a result the benefit of an acquired information out of a recommender should be higher as the cost e.g. for searching it. Additionally, the presentation of items can also be done in different ways. This ranges from simple ids, up to the presentation of different attributes and ends with complex correlations e.g. in ontologies. [Ricci et al., 2011, p.8]

A recommender system can be used for various reasons. Naming all of them would be impossible so the following capabilities are only examples. According to [Klahold, 2009, p. 4 ff.] there exists the following general classification:

Content The recommendation of content is probably the most common use case like [Klahold, 2009, p. 5] claims. The suggested content can be a product, text e.g. in online newspapers, or what song or video to hear/watch next. The web site amazon (www.amazon.de) for example shows users recommendations according to their last visited or bought book. [Ricci et al., 2011, Klahold, 2009, p. 4]

Persons Recommendations of persons can be experts in a specific domain or groups with the same interests. [Klahold, 2009, p. 4,15]

Processes Process suggestions can be additional information about special points of interest in the near of the route from a mapping tool or recommendations for possible restrictions on search pages. [Klahold, 2009, p. 4]

2.2. The Data Mining Process

By definition "Data mining is the process of discovering useful patterns and trends in large data sets"[Larose and Larose, 2015, p. XXI]. However, the steps of this process can vary. In literature there exists several process models, which show how a typical workflow in a data mining application should work. All of them have a lot of overlaps in general. In the following subsections the most important ones are mentioned.

2.2.1. CRISP-DM

The Cross-Industry Standard Process for Data Mining (CRISP-DM) is probably the most common one. The CRISP-DM was developed by analysts from large companies like Daimler-Chrysler, SPSS and NCR to get a uniform industry-,tool- and application-neutral standard. The resulting process is a non proprietary and free standard. [Larose and Larose, 2015, p. 6]

The process consists of 6 phases: The *Business Understanding*, the *Data Understanding*, the *Data Preparation*, a *Modeling*, the *Evaluation* and the *Deployment* phase. The phases mostly depend on the output of another one. For instance, the *Modeling* phase directly depends on the results of *Data Preparation*. After the *Modeling* phase it can be possible that the results of this phase indicate that it is retroactively important to adapt the *Data Preparation* before the *Evaluation* phase can start. [Larose and Larose, 2015, p. 6]

The following figure 2.1 illustrates these phases and their frequent dependencies. The arrows around indicate that the whole process is iterative. Thus the solution and lessons learned from one problem can lead to new questions or demands for other processes [Shearer, 2000].

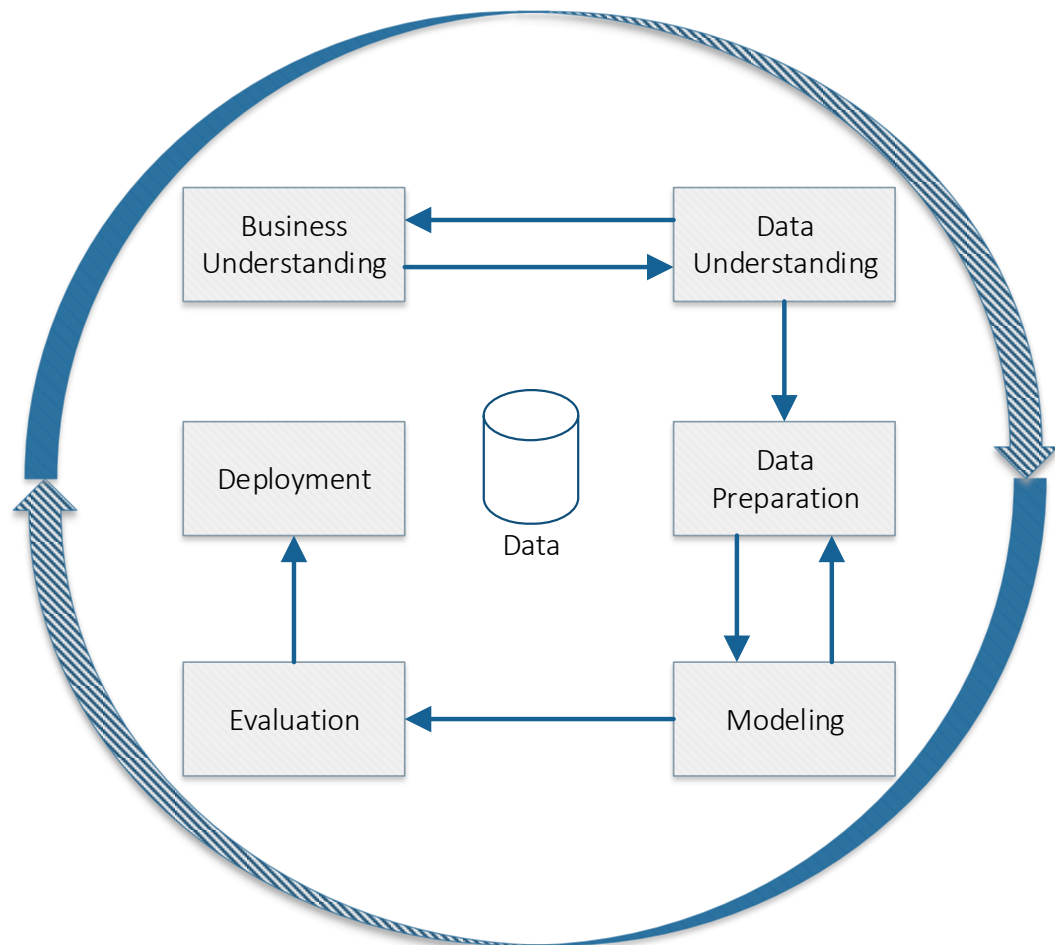


Figure 2.1.: **CRISP-DM Model**. Each square represents a phase. The edges represent frequent dependencies. Adapted from [Cleve and Lämmel, 2014, p. 6]

In the following every phase is shortly explained:

1. Business Understanding Phase

The first phase is one of the most important ones. Usually considerable analyses are necessary to achieve good results later. Therefore, first business objectives should be determined. This includes checking the background of the whole project. Emanating from the current state it's important to know which resources are there and which results should be achieved within the data mining. This can be done by defining business success criteria to measure the success later. In addition to that all requirements, assumptions and constraints and possible risks should be gathered to create a project schedule at the end of this phase. This plan also includes initial assessments of the used tools and techniques. [Shearer, 2000, Cleve and Lämmel, 2014, p. 7]

2. Data Understanding Phase

The second phase starts with the collection of initial data. This data should be de-

scribed and explored within a report to get a basic understanding of them. The explorations should also contain some metrics like minimums, maximums or average values to define the quality of the data. Beside of this it is important to check which data is necessary and available depending on each objective. [Shearer, 2000, Cleve and Lämmel, 2014, p. 7]

3. Data Preparation Phase

The aim of this phase is to prepare the data for the subsequent modeling phase by constructing the final used datasets out of the selected data. Thus the first step is to select relevant data for the given data mining goals by considering the quality and technical constraints like data types. Afterwards this data has to be cleaned and incomplete or inconsistent data must be corrected. This can be done by replacing faulty data with new attributes or construct new ones by merging them. Another aspect is to transform and reformat the given data. Reformatting can also include e.g. the removing of illegal characters. [Shearer, 2000, Cleve and Lämmel, 2014, p. 7]

4. Modeling Phase

In this phase a model is created how to treat the data of the last step. A model can e.g. be rules, clusters or decision trees. To achieve this, first a suitable modeling technique is selected. Afterwards some tests and experiments are designed and configured to check how good the model is. For instance in classification models error rates are used as quality measure. Out of this results used parameters can be modified and the model be refined until the results suits to the requirements. [Shearer, 2000, Cleve and Lämmel, 2014, p. 8]

5. Evaluation Phase

Before the model can be deployed it is important to evaluate the results with the success criteria of the first phase. If they are not reached its important to identify possible (business) reasons why it does not work properly. This failure analysis may reveal errors in the data preparation or the modeling. Then the analyst can return to these phases and correct them. In addition to that the next possible steps like finishing the whole project are made in this step. [Shearer, 2000, Cleve and Lämmel, 2014, p. 8]

6. Deployment Phase

The final phase deals with the usage of the results of phases before. This can either be the generation of a report but also be the implementing of the process. In order to implement it, a strategy for the deployment and a plan how to evaluate it must be made. At the end of the project, a final report containing the summary of the project and a presentation of the data mining results have to be created. In addition to that the process should be reviewed with the experiences made in the current project to identify failures and possible improvements for future projects. [Shearer, 2000, Cleve and Lämmel, 2014, p. 8]

2.2.2. Other Relevant Models

In literature some other data mining models can be found. The *Fayyad* model ([Fayyad et al., 1996]) e.g. only have the 5 phases *Selection, Preprocessing, Transformation, Data Mining*

2. Recommender Systems and Datamining

and *Interpretation/Evaluation*. Compared with the CRISP-DM model it does not contain the first two steps and the last step of the CRISP-DM model. However, the third phase of the CRISP-DM model is here divided in 3 steps. As a result, the Fayyad model focus on the data analysis and preparation whereas the CRISP-DM model show the process in the view of the industry. [Cleve and Lämmel, 2014]

Another similar model is the in [Runkler, 2010, p. 2,3] shown process which is divided in *Preparation, Preprocessing, Analysis* and *Postprocessing*. The main difference compared to the CRISP-DM model is that the first two steps of CRISP-DM model can be bundled into the *Preparation phase*. The remaining process is very similar to it.

In addition to them there exists some other models like the SEMMA model [Cleve and Lämmel, 2014, p. 8,9] and an general model shown in [Aggarwal, 2015, p.3-6].

3. Frequent Pattern Mining

This section starts with a briefly introduction to the origin of *Frequent Pattern Mining* and its applications. Next the basic terms and definitions are illustrated. Finally, some frequent pattern mining techniques like the *Apriori* algorithm are exemplified.

3.1. Description

The area of Frequent Pattern Mining (FPM) or Association Rule Mining (ARM) turned up in the 1990s in the context of supermarkets [Aggarwal, 2015, Karthikeya and Ravikumar, 2014, p2]. The aim of this topic is to infer so called association rules in the form $A \Rightarrow B$ out of frequent itemsets. In this case existing transactions of customers are used as data basis for the itemsets. The mainly used terms "itemset" and "transaction" which are commonly used in literature also descend from this market basket analysis. The association rules can then be used to predict customers habits.

Assume we have 500 transactions from 500 customers in a supermarket. From these customers bought 100 beer and out of these 25 chips. Under the use of ARM we can conclude that $\frac{100}{500} = 20\%$ of all transactions includes beer. This is called "support". In the next step we can conclude that $\frac{25}{100} = 25\%$ of the people who have bought beer also bought chips. This measure is called "confidence". Out of these observations we can refer that the article beer is well sold and that many people who buy beer also want to buy chips. As a consequence, supermarkets could do some individual advertisement within these chips for customers who buy beer. Alternatively, they can put the chips next to the beer. [Aggarwal, 2015, p. 94]

According to [Aggarwal and Han, 2014, Aggarwal, 2015, p. 93,94] there exists a lot of possible application:

- **Customer Transaction Analysis:** Here transactions of customers are analyzed to get associations between their bought items. This application is the before mentioned analysis of the buying behavior of customers.
- **Text Mining:** Texts can be illustrated as bags of words. Each of those words can be used like the items in the introduction example. Afterwards FPM is able to find keywords or terms which occur together. This can e.g. be helpful for digital libraries to manage books effectively. [Aggarwal, 2015, p. 429]
- **Generalization to dependency-oriented data types:** In addition to the original model it can be generalized for time-series data, sequential data and spatial data. These can e.g. be used for Web-design or recommendations by analyzing the patterns of browsing behavior for web mining [Srivastava et al., 2000] and software bug detection. [Aggarwal, 2015, p. 93] It also includes applications like finding the proportion

of children, who are good readers, if their parents are also good readers [Larose and Larose, 2015].

- **Subroutine of data mining problems:** Frequent Pattern Mining can be used as for other domains like clustering, classification and outlier analysis. [Aggarwal, 2015, p. 93]

The problem of FPM is the dimensionality. That's because the number of association rules rise exponentially with the number of items. Thus if you have k items there are already 2^k different itemsets. [Larose and Larose, 2015] Additionally, if you only count positive values this can lead to $k * 2^{(k-1)}$ possible association rules [Hand et al., 2001, Larose and Larose, 2015].

3.2. Models and Definitions

Before executing the algorithms, it is important to have a look on the data model for Frequent Pattern Mining to understand them. As mentioned before the input for the later process are Transactions like the minimal example in table 3.1. These consists out of a Transaction ID and one or more items. In addition to that other attributes like the persons who are involved or the dates are possible but not necessary. Out of this data there exists two kinds of representations for the later algorithms: The transactional data format and the tabular data format which are shortly introduced in the following tables 3.2, 3.3. [Larose and Larose, 2015, p. 604, 605]

Table 3.1.: **Transaction Example**

Transaction ID	Items
1	Chips
2	Beer, Cola, Chips
3	Cola
4	Beer, Chips
5	Cola, Chips
6	Bread
7	Bread, Beer, Cola, Chips
8	Beer, Chips

The transactional data in table 3.2 format has only two columns. One for the ID and one for each single item. This means that each itemset can have multiple entries:

Table 3.2.: Transaction example in transactional data format. Data belongs to table 3.1

Transaction ID	Item
1	Bread
2	Beer
2	Cola
2	Chips
...	...

The other very popular format is the tabular data format. Here every possible item has an own column. This allows it to use flags (e.g. bought / not bought) for every item. If there are many different items, this will lead to the problem that there would be a lot of unused columns. This kind of format is shown in table 3.3:

Table 3.3.: Transaction example in tabular data format. Data belongs to table 3.1

Transaction ID	Bread	Beer	Cola	Chips
1	0	0	0	1
2	0	1	1	1
3	0	0	1	0
4	0	1	0	1
5	0	0	1	1
...

3.2.1. Itemset Mining

Let D be the transactional database with n transactions T ($D = \{T_1, \dots, T_n\}$) like them shown in table 3.1. Every transaction contains a set of items / attributes $T_i = \{i_1, \dots, i_m\}$. Such a set of items $A \subseteq T_i$ is called an itemset. The itemsets themselves can be classified as k -itemset by their cardinality k . This means for example that 2-itemsets are all itemsets which have exactly 2 different items. All possible items U in this universe have the dimension $d = |U|$. [Aggarwal and Han, 2014, Aggarwal, 2015, p. 94]

Let A and B be two itemsets for the next definitions.

The fraction of transactions which contains a specific itemset as subset is called frequency or **support** [Aggarwal, 2015, p. 95]. This means the support of an itemset A is defined as:

$$support(A) = \frac{|\{T \in D | A \subseteq T\}|}{|D|} \quad (3.1)$$

If every combination of items is used this can lead to huge amount of different itemsets. Therefore, the aim of this step is to get so called frequent itemsets. Frequent itemsets are all itemsets which have at least a specific minimal support (*minsup*). This *minsup* can either be an absolute or a relative value. [Aggarwal, 2015, p. 95]

Relative means that it is a fraction of the Transactions e.g. 3% of all transactions. There are good reasons for both of them. If we use relative values, it is easier to handle huge

differences in the number of transactions you have. Let e.g. in one case the number of transactions be 500 in another one 10000 then finding an absolute minimum support for both of them would be nearly impossible. On the other hand, an absolute *minsup* can help blocking frequent itemsets, if you have an insufficient amount of transactions.

In the following some properties and definitions for itemsets are shown. An important property of itemsets is the **Support Monotonicity Property**. According to [Aggarwal, 2015, p. 96] this indicates that for every itemset, A which is a subset of B , the following apply:

$$support(A) \geq support(B) \forall A \subseteq B \quad (3.2)$$

A simple explanation is, that if a transaction for a specific itemset B exists, every subset of it must also be in this transaction. Additionally, there is the possibility that other transactions exist, which only include the specific subset but not the whole B .

The **Downward Closure Property** says that all subsets of frequent itemsets must also be frequent [Aggarwal, 2015, p. 96]. This can be deduced by the Support Monotonicity Property. According to it the support of all subsets must be at least as high as for the set itself. If an itemset reaches the *minsup* and is therefore frequent every subset must have the same or a higher support and as a result it must be frequent too. This property is important for later algorithms to improve the performance.

Maximal Frequent Itemsets are all itemsets which fulfill the *minsup* (and are therefore frequent) and have no superset which is also frequent [Aggarwal, 2015, p. 96]

Closed Frequent Itemsets are all frequent itemsets which have no frequent superset with the same support. [Aggarwal and Han, 2014, p. 48]

The following example in figure 3.1 shows these circumstances:

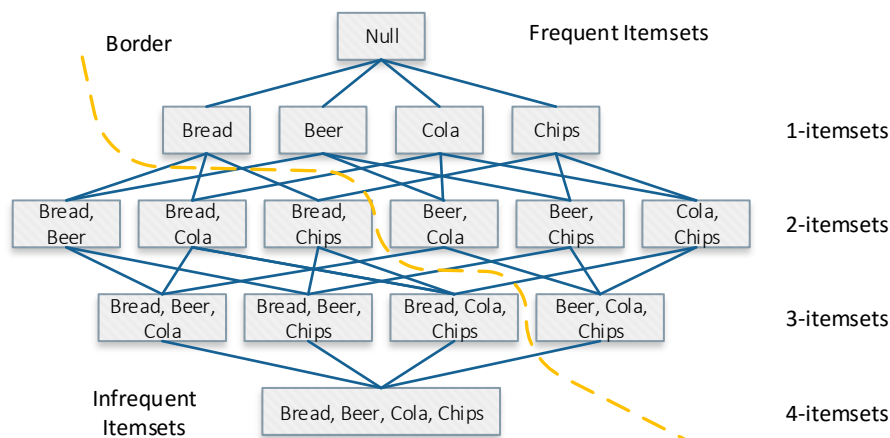


Figure 3.1.: **Lattice of itemsets**. Adapted from [Aggarwal and Han, 2014, p. 23]

The figure conceptually shows all 2^4 (formular in subsection 3.1) combinations of itemsets from the universe U with $|U| = 4$ of the basic example in table 3.1. The nodes of this graph represent the itemsets. Every edge between them indicates how the itemsets are associated together. Every linked itemset differs by one item. Every line represents all itemsets within the same cardinality k , e.g. {Bread, Beer} with $k = 2$ as 2-itemset. The border separates the frequent itemsets from the infrequent ones. All maximal frequent itemsets are directly above this border, because they have no frequent superset. In addition to that this lattice shows how the algorithms work. According to [Aggarwal, 2015, p. 97] all frequent pattern mining algorithms implicitly or explicitly traverse this graph to determine frequent patterns.

3.2.2. Association Rule Mining

Apart from the generation of frequent itemsets, it is important to create association rules. The confidence of an association rule $A \Rightarrow B$ is the probability that an itemset contains B if it already contains A . This means referred to [Larose and Larose, 2015, p. 606] the confidence is defined as:

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{\text{support}(A \cup B)}{\text{support}(A)} \quad (3.3)$$

Analogously to the *minsup* of the itemsets, there exists a minimal confidence *minconf* as lower bound.

In general, the generation of association rules can be divided in two steps:

1. Generate frequent itemsets which fulfill *minsup*
2. Create association rules out of the frequent itemsets which fulfill *minconf*

A precise description how this works in details can be found in section 3.3. Emanating from these steps many rules like $A \Rightarrow B$ can be generated. According to [Aggarwal, 2015, p. 98] each of those created association rules have the following properties:

- $\text{support}(A \cup B) \geq \text{minsup}$
- $\text{confidence}(A \Rightarrow B) \geq \text{minconf}$

Consequently, all association rules have a minimum dimension of occurrences (*minsup*) and a specific minimal strength (*minconf*). [Aggarwal, 2015, p. 98]

According to [Aggarwal, 2015, p. 98] the first phase is the computationally most expensive one. Thus most of the literature concerns with optimization possibilities for this phase. In subsection 3.3 and 5.1 the probably most popular one can be found.

Similarly to the support, the confidence also has a monotonicity property called **Confidence Monotonicity**. Let A, B, I be itemsets and $A \subset B \subset I$. On the basis of [Aggarwal, 2015, p. 98,99] the following statement hold:

$$\text{conf}(B \Rightarrow I \setminus B) \geq \text{conf}(A \Rightarrow I \setminus A) \quad (3.4)$$

3.2.3. Generalization

Until now only boolean values have been used. However, association rules are not limited to those values. Therefore, quantitative and categorical data can also be used. Let's imagine we have the age and the number of kids as numerical properties. Then the idea is to build meaningful intervals and handle the data like before [Cleve and Lämmel, 2014, p.189-193]. You can e.g. choose intervals "< 20 years", "20 – 29 years" and "> 29 years" for the age. If you then have a transaction with age = 16, kids = 0 it can be presented in tabular form:

Table 3.4.: Example for quantitative data

Transaction ID	< 20 years	20-29 years	>29 years	...
1	1	0	0	...
...

Another possibility is hierarchical data like detergent from different brands. If every of them will be presented as a own item there would be a lot of different itemsets with less meaning. If you group them to "Detergent" later associations rules would have a much higher support as before. [Cleve and Lämmel, 2014, p. 190]

3.3. Algorithms

In the last sections the basic terms and ideas have been shown. A lot of work has been done to improve e.g. the performance and the memory usage. Showing all of them would be too much or this work. Therefore, first the whole process will be shown in a brute force manner. Afterwards the common improvement, the *Apriori* algorithm is introduced. In subsection 5.1 more approaches can be found.

3.3.1. Brute Force

To sum up the last sections we first have to generate the frequent itemsets. Afterwards create association rules out of them. A simple approach for the first step would be to generate all possible itemset combinations. Afterwards the support of every of these sets have to be calculated. As basis for the following process the transactional data of table 3.1 is used. There are 4 different items which lead to $2^4 = 16$ different itemsets (incl. empty set). Let the required parameters be $minsup = 0.25$ and $minconf = 0.6$:

Table 3.5.: **Support of all itemsets** without empty set divided by dimension

Itemset	Count	Support	\geq minsup
Bread	2	25%	✓
Beer	4	50%	✓
Cola	4	50%	✓
Chips	6	75%	✓
Bread, Beer	1	13%	
Bread, Cola	1	13%	
Bread, Chips	1	13%	
Beer, Cola	2	25%	✓
Beer, Chips	4	50%	✓
Cola, Chips	3	38%	✓
Bread, Beer, Cola	1	13%	
Bread, Beer, Chips	1	13%	
Bread, Cola, Chips	1	13%	
Beer, Cola, Chips	2	25%	✓
Bread, Beer, Cola, Chips	1	13%	

The first step is to count the number of transactions which contain the particular itemset in column 1. The result is shown in the second column. Out of these we can calculate the support. The overall number of transactions is 8. Therefore, we e.g. get for {Cola, Chips} $support(Cola, Chips) = \frac{3}{8} \approx 0.38$. After the calculation 8 itemsets fulfill the *minsup* and are therefore frequent. If we choose a higher *minsup*, e.g. 0.5, we will get only the 4 frequent itemsets {Beer}, {Cola}, {Chips}, {Beer, Chips} instead of the 8. This leads to a performance boost in the next step because there are less possible rules to be checked.

The next step is to calculate the confidence out of the frequent itemsets. Therefore, we have to divide all frequent itemset C into two subsets A and B with $B = C \setminus A$ which are not empty and build up rules like $A \rightarrow B$.

Table 3.6.: **Calculated association rules**

Rule	confidence	\geq minconf
Beer \rightarrow Chips	100%	✓
Chips \rightarrow Beer	67%	✓
Cola \rightarrow Chips	75%	✓
Chips \rightarrow Cola	50%	
Beer \rightarrow Cola	50%	
Cola \rightarrow Beer	50%	
Beer \rightarrow Cola, Chips	50%	
Beer, Cola \rightarrow Chips	100%	✓
Beer, Chips \rightarrow Cola	50%	
Chips \rightarrow Beer, Cola	33%	
Cola \rightarrow Beer, Chips	50%	
Cola, Chips \rightarrow Beer	67%	✓

After building all rules in column 1 the confidence must be calculated. We already have calculated the support of all possible itemsets so we do not need to calculate them again. The confidence can be calculated with equation 3.3 and is e.g. $confidence(Cola, Chips \rightarrow Beer) = \frac{support(Beer, Cola, Chips)}{support(Cola, Chips)} = \frac{2}{3} \approx 0.67$. All rules which are checkmarked fulfill the $minconf$ and the $minsup$ and are therefore our searched for association rules. If the $minsup$ in the last step would be the already mentioned 0.5 only the 2 association rules $Beer \rightarrow Chips$ and $Chips \rightarrow Beer$ had to be calculated.

This small example is computationally no high effort, because the current universe has only 4 different items. Lets assume it would be $k = 100$ items. Then we would get $2^k = 2^{100} > 1.2 * 10^{30}$ different itemsets which need to be counted in the database. Doing this for every request would take a probably huge amount of time and is not advisable. Consequently, there are a lot of algorithms to improve the efficiencies. According to [Aggarwal, 2015, p. 100] these algorithms use one or more of these approaches:

- Reducing the number of itemsets by using tricks, such as the downward closure property.
- Improve the counting of the support by disregarding transactions that are irrelevant for the counting of itemsets.
- Using a better datastructure for the candidates or transactions which allows a more efficient counting.

3.3.2. Apriori

As mentioned before the "Brute Force" attempt is not advisable for transactional databases with a lot of different items. The in [Agrawal and Srikant, 1994] proposed *Apriori* algorithm was the first attempt to solve this problem. Therefore, the *Apriori* uses a level wise approach to create the itemsets and uses the in subsection 3.2.1 proposed downward closure property to prune itemsets. Level wise means that first all itemsets of length k are created and afterwards the itemsets of length $k + 1$ and so on. [Aggarwal, 2015, p. 100]

Algorithm 3.1 Apriori algorithm

```

1: Input: Database T, Minimal Support minSup
2: Output: Collection of frequent itemsets

4:  $F_1$  = all frequent 1-itemsets
5:  $k = 1$ 

7: while  $F_k$  not empty DO
8:    $k = k + 1$ 
9:   //Join itemsets of last step
10:  Generate  $F_k = \{A \cup B | A, B \in F_{k-1} \text{ and } |A \cup B| = k\}$ 

12:  for each X in  $F_k$  do
13:    //pruning
14:    if all (k-1) subsets of X are in  $F_{k-1}$  then
15:      Do nothing
16:    else
17:      Delete X from  $F_k$ 
18:      Continue with next X
19:    end if

21:    //check if it is frequent
22:    if support(X) < minsup then
23:      Delete X from  $F_k$ 
24:    end if
25:  end for

27: end while

29: return  $\bigcup_i F_i$ 

```

This algorithm first creates all frequent 1-itemsets by counting their support and checking if it is $\geq \text{minsup}$. Afterwards it iterates over all k where k is the number of items in the current itemset. In every iteration it joins every (k-1)-itemsets which only differ in 1 item. As a result, it get all itemsets with k elements as candidates which must then be validated to be frequent.

Consequently, every of these candidates (k-itemsets) is tested if all (k-1) subsets are in F_{k-1} . This make sure that every subset of the current itemset is frequent. At this point the downward closure property which say that every subset of a frequent itemset must also be frequent can be used. If one of the (k-1) subsets of the current candidate is not frequent the candidate itself can not be frequent and can therefore be removed without calculating the support. The fact that the current candidate is created by joining two already frequent itemsets is not sufficient because there can be another subset which is not frequent. Let e.g. a, b, c, d be items and $\{a, b\} \in F_{k-1}$ and $\{a, c\} \in F_{k-1}$ but $\{b, c\} \notin F_{k-1}$. Then after the joining of the itemsets of the last step $\{a, b, c\} \in F_k$ although it can not be frequent due to the *Downward Closure Property*. Therefore, the check in line 14 of algorithm 3.1 would indicate that not all subsets are frequent and remove it from the possible candidates before the support is calculated.

If the candidate satisfies the first test the support can be counted to validate if the itemset is frequent. If it is not, it can also be removed. After this candidate validation all itemsets of F_k are frequent and the while loop can be performed once again until no new (k+1) itemset fulfill the frequency property.

3. Frequent Pattern Mining

The association rule mining can afterwards be done in the usual way as described before in subsection 3.3.1.

The following example shows the performance improvements. As basis the transactional data of table 3.1 is used again, with a single additional transaction, which contains {Bread, Beer, Cola, Chips, Butter, Cake}. The *minsup* still remains 0.25. As a result, there exists 9 transactions. Therefore, all itemsets with an absolute support less than 3 ($\lceil 9 * 0.25 \rceil = \lceil 2.25 \rceil$) are infrequent.

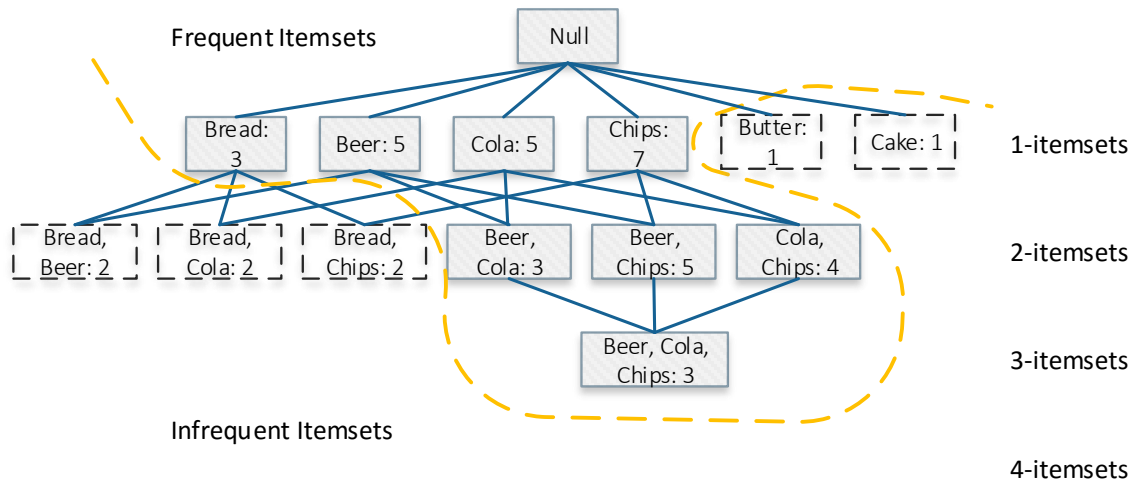


Figure 3.2.: **Execution lattice of *Apriori* algorithm.** Every node represents an itemset with its absolute support. Dashed nodes are infrequent patterns after a join. Edges represent joins. Adapted from [Aggarwal and Han, 2014, p. 26]

Every node in this figure represents an itemset. The solid nodes are frequent itemsets, whereas the dashed ones are infrequent. The itemsets are sorted by their length. As mentioned before each candidate of length k is generated by a join of two of its frequent subsets of size $(k-1)$. Every edge represents such a join. This means each bottom up edge from a k -itemset indicates all $(k-1)$ -subsets of it. All itemsets which do not fulfill the monotonicity property are not shown in this graph. As a result there exists no outgoing edge downwards from an infrequent itemset.

According to the algorithm the support generation only take place if all subsets of size $(k-1)$ are frequent. This means the support is calculated for every node in figure 3.2, because all others are not in the graph. Taking the brute force approach for this $i=6$ items would lead to $2^6 - 1 = 63$ (without null) iterations to calculate the support. In this case the *Apriori* only needs 13.

4. Process Mining

The aim of process mining is to gather process related information like a process model itself. Process mining uses similar to frequent pattern mining the log data to extract this information. [van der Aalst, Wil M. P., 2011, p. 1]

Unlike the log for frequent pattern mining which only requires a set items and their ids, the log for process mining should contain event based information. Every event must at least a refer to a specific task and case and should be totally ordered within its case [van der Aalst, W.M.P. et al., 2003b]. Furthermore, they can have additional attributes like a resource (e.g. a person) or costs [van der Aalst, Wil M. P., 2011, p. 99].

The following table 4.1 illustrates such an event log:

Table 4.1.: Example of an event log

Case	Event type	Activity	Timestamp	User
1	start	a	16.11.2015 9:30	User 1
1	complete	a	16.11.2015 9:45	User 1
1	start	b	16.11.2015 10:30	User 1
1	start	c	16.11.2015 10:45	User 2
1	complete	b	16.11.2015 11:04	User 1
1	complete	c	16.11.2015 11:15	User 2
2	start	a	21.11.2015 9:45	User 3
...

The event type in this example can either be *start* which indicates the start of an event or *complete*. In this example *b* is started before *c* but completed after the start of *c*. Therefore, we can assume that these two activities are executed in parallel. If the completion would be in reverse which means *b* started before *c* but *c* is also completed before *b* the result would be the same.

The log also contains a sequential execution. The activity *a* is started and completed before the next one (*b*) starts.

The resulting process models can afterwards be presented in many different modeling languages. Common representations are:

- **Petri Nets** Petri nets are one of the oldest and best known process model. In addition to that *Petri Nets* are very intuitive. Petri nets consists on transitions which represents events and places. [van der Aalst, Wil M. P., 2011, p. 33]

- **Business Process Modeling Notation (BPMN)** According to [van der Aalst, Wil M. P., 2011, p. 42,43] the *BPMN* has become to one of the most common process model languages. In *BPMN* the events are represented as tasks which can be connected. Every split or join needs a gateway. In addition to that, there exists start and end events.
- **Event-Driven Process Chains (EPCs)** Models in *EPC* notation are similar to the *BPMN*. Here functions represent the activities. Similar to the gateways in *BPMN* there exists connectors for splitting and joining operations. Although only a subset of possibilities is covered by *EPCs*. [van der Aalst, Wil M. P., 2011, p. 44,45]

The following figure 4.1 illustrates the basic example of case 1 in the *BPMN* notation:

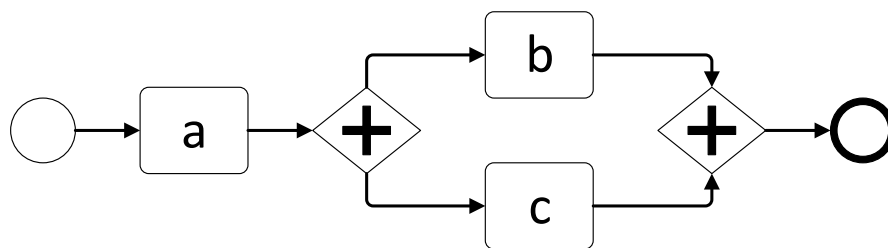


Figure 4.1.: Example process in BPMN notation

In this process model the activity *a* is executed after the start event. Afterwards there is an AND-split gateway which indicates the parallelism of *b* and *c*. This parallelism ends with the AND-join gateway which leads to the end event.

The minimal example process shows the creation of a process model with the information of the event log. Apart from that there exists more application possibilities. According to [van der Aalst, Wil M. P., 2011, p. 9,10] process mining has the following different types:

- **Discovery** The aim of discovery is to create process models out of the event log without additional information.
- **Conformance** The objective for conformance is the opposite. Here process mining is used to validate an existing process model by the event log.
- **Enhancement** Enhancement means that there already exists a process model which should be improved with process mining.

5. Related Work

The later presented approach derive from the area of frequent pattern mining and process mining. Therefore, the research is structured into algorithms for frequent pattern mining and process mining. Finally relevant work to handle these topics in the context of collaborative systems is proposed.

5.1. Frequent Pattern Mining

With [Agrawal et al., 1993] the first paper of this mining problem was introduced. The *AIS* algorithm uses two phases to generate the rules. First all frequent itemsets, which have a specific minimum support, are generated out the transactions. Afterwards association rules are generated, which fulfill a minimum confidence. These represents the in section 3.2 introduced two phases.

The first phase of these algorithms take most of the time to generate the frequent itemsets. Consequently most of the research is done for this part. In the following some algorithms to improve efficiency are shortly introduced:

In 1994 [Agrawal and Srikant, 1994, Srikant and Agrawal, 1995] established the compared to the *AIS* much faster *Apriori* algorithm. This algorithm is mainly the same as the *AIS* but has an improved generation of large itemset, by extending the pruning techniques with an efficient using of the support monotonicity principle. Therefore, does not build all itemsets of length $(k+1)$ out of the frequent ones with length k . Instead it only allows the support calculation for those of length $(k+1)$, if all subsets of it are frequent. This pruning technique makes this algorithm much faster as the *AIS* like tests in [Agrawal and Srikant, 1994] proofed.

In the same year [Agrawal and Srikant, 1994] proposed its *AprioriTID* algorithm. Here the performance can be improved because the database is only scanned once at the start. Afterwards all candidates are intern saved shorter as C_k whereas k is the length of the containing itemsets. These C_k s are then saved in the form $(TID \rightarrow X_k)$ whereat each X_k is an itemset with k elements. For example this means that C_1 contains all itemsets with one element. Afterwards the *Apriori* algorithm is used but instead of scanning the database it uses for each step $(k+1)$ the generated C_k .

The *AprioriTID* works very good in later steps with an higher k but is slower than the normal *Apriori* at the start. There [Agrawal et al., 1996] proposed the *AprioriHybrid* which uses the best of both algorithms. It started with the normal *Apriori* and switches later to the *AprioriTID*.

In [Hidber, 1999] the *Continuous Association Rule Mining Algorithm* (CARMA) has been introduced, which needs at most two database scans. Therefore, it creates a lattice with each potentially frequent itemset in it. Every of those itemsets has a lower and upper bound for its support. These are displayed continuously. The user is able to adapt the *minsup* and *minconf* at runtime, until the shown probably results fit into the requirements. Then the database is scanned once again to get exact values. The advantage of this is, that there is no need to rescan the database each time. This paper also claims that this algorithm is 60 times more memory efficient than the *Apriori*.

The *Direct Hashing and Pruning* (DHP) algorithm [Park et al., 1995] efficiently generates large itemsets and effectively reduce the transactional database size. This can be done by using hash techniques. This algorithm is especially very efficient for large 2-itemsets.

Another popular algorithm the *FP-Growth* was proposed in [Han et al., 2004]. This algorithm uses a frequent pattern mining tree (FP-tree) as data structure to store all needed information. Afterwards the *FP-Growth* is used to mine the frequent patterns. This process only needs two database scans which make it very efficient.

5.2. Process Mining

In [Cook and Wolf, 1998] the three approaches *RNET*, *KTAIL* and *MARKOV* for process discovery are introduced.

RNET This is a statistical approach, which uses the past behavior to characterize its state. Therefore, neuronal networks are used which have to be trained. This is done with the difference of the current and the expected output. According to the author this neuronal approach is robust against noise but not very exact. Even with a perfect input it can not give a proper output for the same stream.

KTAIL *KTAIL* uses an algorithmic approach. Therefore, it uses the behavior of the future to characterize its state. This means the state is defined by the next actions which are performed. To achieve this equivalence classes have to be build. These are classes, which have the same k-length future or in details, which have the same next k activities. Each of these equivalence classes E_i represents a single state. At the end each token a is concatenated with an activity $p \in E_i$. If this concatenation exists, an edge is created from a to this equivalence class.

MARKOV The *MARKOV* model is a hybrid between the statistical and algorithmic algorithms. It uses as well past behavior as future behavior to define its state. Therefore, it creates a probability table for every activity. Afterwards nodes are created out of this table for each activity in the event graph. Every event type is a vertice. If such a vertice does not satisfy a minimal threshold and probability it will be deleted. Afterwards edges are created between the vertices of a sequence. To avoid over-connected vertices every of these vertices is splitted. This can be done by finding disjunct input and output sets of edges. At the end this graph is converted into a state machine. Here all edges of the graph are now vertices and every vertice of the graph is now an edge in the resulting finite state machine.

The author also claims, that *MARKOV* scales well for long event streams, but not for a lot of different event types in contrary to *KTAIL*

The α -algorithm has been introduced in [van der Aalst et al., 2004]. This algorithm uses relations and causalities like after a comes b which is defined by $a \rightarrow_w b$ iff $b >_w a \wedge a \not\prec_w b$, which means that there exists a trace where b comes after a but no one where a comes after b , to create a process model out of the log. However, this algorithm can only create simple components and is not able to handle typical workflow elements like duplicated tasks or short loops how [Medeiros et al., 2003] claims.

The $\alpha+$ -algorithm [Li et al., 2007] is an extension to the α -algorithm to handle duplicate tasks. Therefore, a predecessor / successor table (P/S table) is created to find that duplicated tasks.

The β -algorithm [Wen et al., 2009] uses another approach. It also uses causality information but in contrast to the before mentioned algorithms it does not see an activity as atomic or only use the completion of it as event. For this reasons tasks are seen as time intervals instead of timestamps. Therefore, it uses the event types *START* and *COMPLETE* in relation together to detect explicit parallelism.

5.3. Other Relevant Work

There is several research how to help end users in flexible or ad hoc systems. [Motahari-Nezhad and Bartolini, 2011, Schonenberg et al., 2008] describe how to support end users in flexible systems to find the next step in a process. Therefore, they use the similarity of the current workflow to a successfully completed process to make recommendations for end users. Both also contains metrics to identify similar ones.

[Dorn et al., 2010] try to improve recommendations for possible next steps in ad hoc processes by introducing a self-adjusting recommendation. Here possible recommendations based on the specific user are combined with crowd based recommendations.

[Schönig et al., 2015a, Schönig et al., 2015b] propose a method for recommendations for modelers of those systems with the aim to create resource-aware templates. This is reached by pre-processing the data with frequent pattern mining. Afterwards this already resource-aware data can be used to build up a process model.

In [Robillard and Dagenais, 2008] an algorithm to retrieve task related clusters from change histories in a source code is proposed in the context of tasks for developers. Here different filtering heuristics are briefly described to improve the results.

[Makanju et al., 2009] propose the *IPLoM* (Iterative Partitioning Log Mining) algorithm to cluster event logs. This approach differs significantly from the before proposed frequent pattern mining algorithms. Here text based logs are partitioned by their token count, their token position and for bijections. Every partitioning step is done on the the already existing partitions of the last step. In the end every partition represents a cluster.

5. *Related Work*

Lastly [Eder and Pichler, 2005] describes how to predict execution duration in a probabilistic way. These helps users and modelers to schedule tasks.

Part III.

Conceptual Approach

6. Approach

This chapter describes the datamining process in concept. Thus the results are structured according to the phases of the CRISP-DM process model, which is introduced in section 2.2.1. The conceptual model itself is built on the existing SocioCortex Wiki. Therefore, all perceptions are made on this. The following infographic gives a short overview about the executed steps as part of the CRIPS-DM Model:

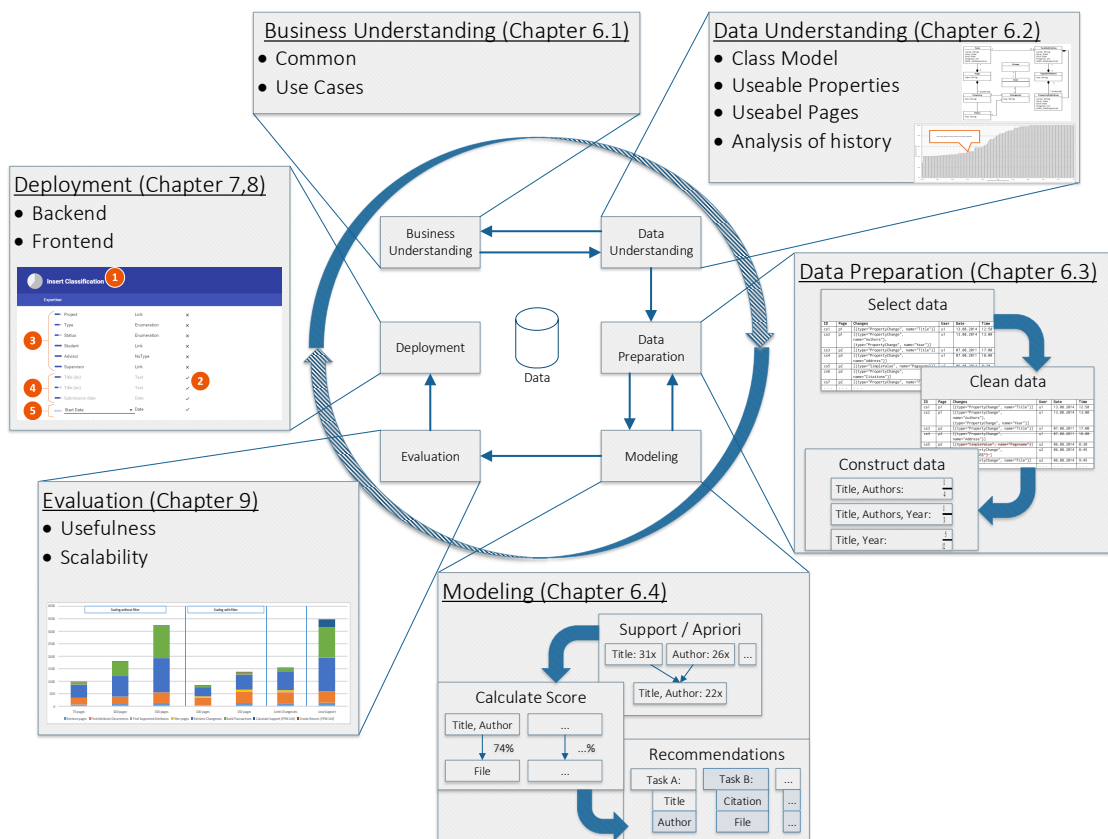


Figure 6.1.: Basic steps of the approach as part of the CRISP-DM model

The first part of this chapter deals with the understanding of the business. Therefore, some information about the system itself and its background are shown. In addition, possible use cases for recommender are illustrated.

Section 6.2 (**Data Understanding**) is aimed to get familiar with the entire data, before the primary data mining process starts. Therefore, a conceptual UML model is shown and

diverse analysis of the data is made to know what data is important and how to set later parameters.

Data Preparation (section 6.3) gives an overview what data have to be selected, how to they should be cleaned and how to construct usable data out of it. These algorithms directly depend on the results of the prior section.

Section 6.4 introduce the algorithms how the recommender creates recommendations for the different use cases out of the cleaned and constructed data of the last step.

The Evaluation and Deployment part can be found in chapter IV.

6.1. Business Understanding

SocioCortex is a collaborative information system developed by the *Software Engineering for Business Information Systems* (sebis) chair of TU Munich. Moreover, it is a collaborative wiki with the capability to store data and knowledge in a semi-structured form. This knowledge can be retrieved with different querying methods and functions. *SocioCortex* itself provides the in [Matthes et al., 2011] proposed hybrid wiki capabilities in combination with different clients. These can e.g. be for end-user-oriented quantitative model analysis (Spreadsheet 2.0) or knowledge-intensive process support (*Darwin*). [SEBIS chair, 2015]

In detail users have the ability to create new pages and fill them with structured and unstructured data. Users themselves don't need to have special modeling skills or previous knowledge to do this [Matthes et al., 2011]. Unstructured data can be simple text as description in the page. Structured data can e.g. be attributes. Attributes are key-value pairs. These values can be text literals (e.g. "Name" -> "John") or links to other sites or users. Attributes which are created by an end-user are called "free attributes". Furthermore, there is the availability to structure the sites by sub sites and create tasks and link attributes to them.

In addition to these flexible creations there is the possibility for special skilled users so called modelers to create predefined types to enable end-users a faster and more consistent way to handle specific circumstances. For instance, modelers are able to create a page type "publication" with the attributes "Authors" and "File" with the task "Link citations". Users are then able to use this definition and can create a new page, which is automatically prefilled with these characteristics.

To facilitate this process for modelers and users there should e.g. be recommendations for a modeler which attribute might be important for a specific task. This leads to the following possible use cases.

6.1.1. Create New Task Definition with Attributes (Use Case 1)

If pages have no predefined tasks it can be helpful for modelers to get recommendations which attributes fit together. The following table 6.1 defines this use case:

Table 6.1.: Use Case 1: Create new task definition with attributes

Name	Create new task definition with attributes	
Actors	Modeler	
Description	A modeler wants to create a new task definitions with a specific name for a specific page type. Before the creation all recommended attributes which are suitable together should be shown. The modeler also wants to link the recommended attributes automatically when he or she click on create. The modeler should also be able to set the attribute type for free attributes (only simple attribute types which don't need any specific settings)	
Precondition	Page type should already exist.	
Main Scenario	Interaction	System reaction
	Modeler choose page type	Existing task definitions with their attributes and their related scores are shown. Recommended attribute groups for new tasks are shown. These attribute groups contain attribute definitions as well as common used free attributes.
	Modeler choose attribute group and enter a task name	
	Modeler choose attribute types (only simple types) for free attributes	
	Modeler click on create	New task is created with given name New attribute definitions are created for all free attributes of pages (with selected simple attribute types) Newly created and existing attribute definitions in an attribute group are linked to newly created task
Post Conditions	Updated task definitions for entire type are displayed. Recommended scores for all attributes are refreshed.	

6.1.2. Remove Existing Task Definition (Use Case 2)

If existing task definitions are unnecessary e.g because end-users do not use them, it can be advantageous to remove these definitions. Else wise it may confuse end-users. The following table 6.2 defines this use case:

Table 6.2.: Use Case 2: Remove existing task definition

Name	Remove existing task definition	
Actors	Modeler	
Description	A modeler deletes a specific existing task definition in a specific page type. All associated elements to this task definitions like attributes should also be unlinked.	
Precondition	Page type with at least one task definition must exist for entire type.	
Main Scenario	Interaction	System reaction
	Modeler choose page type	Existing task definitions with their attributes and their related scores are shown. Recommended attribute groups for new tasks are displayed. These attribute groups contain attribute definitions as well as usually used free attributes.
	Modeler click on delete for specific task definition	Task definition is removed.
		Every attribute definition, which is linked to the specific task definition is unlinked
Post Conditions	Task definitions for entire type are shown (without removed one). Possible formerly associated attributes are recommended elsewhere.	

6.1.3. Add Recommended Attribute to Existing Task Definition (Use Case 3)

Processes may change during time. There can e.g. be new free attributes many end-users utilize. Then it is beneficial to improve existing task definition by adding new attributes to them. The following table 6.3 illustrates this use case:

Table 6.3.: Use Case 3: Add recommended attribute to existing task definition

Name	Add recommended attribute to existing task definition	
Actors	Modeler	
Description	A modeler want, to add a specific recommended attribute to a specific task definition. If the attribute is a free attribute the modeler will have to select a proper type for it. Afterwards a new attribute definition should be created automatically while linking the attribute to the task. If the attribute definition already exists this one will be linked.	
Precondition	Page type with at least one task definition must exist for entire type.	
Main Scenario	Interaction	System reaction
	1. Modeler choose page type	Existing task definitions with their attributes and their related scores are shown. Recommended attribute groups for new tasks are displayed. These attribute groups contain attribute definitions as well as common used free attributes.
	2. Modeler choose an attribute	
	3. Modeler click to add it	Attribute definition is linked to task definition
Extension	2a. Recommended attribute is a free attribute	
	2a1. Modeler select proper simple type for attribute	Enable attribute addition definition
	3a. Recommended attribute is a free attribute	
	3a1. Modeler click to add attribute	Analogous attribute definition to free attribute is created
	3a2.	Created attribute definition is linked to task definition created
Post Conditions	Updated task definitions for entire type are shown.	

6.1.4. Remove Attribute from Task Definition (Use Case 4)

It may turn out that some attributes do not match a specific task. Then it is necessary to remove them from the task definition. The following table 6.4 explain this use case:

Table 6.4.: Use Case 4: Remove attribute from task definition

Name	Remove attribute from task definition	
Actors	Modeler	
Description	A modeler deletes a specific existing attribute definition from a task definition. The attribute definition itself should exist furthermore.	
Precondition	Page type with at least one task definition with at least one attribute definition must exist.	
Main Scenario	Interaction	System reaction
	Modeler choose page type	Existing task definitions with their attributes and their related scores are shown. Recommended attribute groups for new tasks are displayed. These attribute groups contain attribute definitions as well as usually used free attributes.
	Modeler click on delete for specific attribute definition	Attribute definition is unlinked from task definition.
Post Conditions	Updated task definitions for entire type are shown. Possible formerly associated attribute is recommended elsewhere.	

6.2. Data Understanding

The aim of the following section is to become familiar with the entire data before the primary data mining process starts. For a basic understanding of the system a conceptual UML class model is illustrated first. The main aim of use case 1-4 is to get associations between the attributes for a possible prediction what attributes are often used together. The suggestion itself always refer to a single type (a so called `EntityType`) of page.

To get the best results it make sense to check what kind of data is usable for the later recommendation. Therefore, the first step is to check if it is wise to use only `AttributeDefinitions` or also so called free attributes. The second step identifies if the data of all pages of the entire type is reasonable to be used. As last step the `ChangeSets` (the history) of the system will be illustrated. In addition to that possible values parameters of the later recommender are shown.

6.2.1. Class Model

The UML class model in figure 6.2 illustrates a simplified conceptual model of the entire system. Simplified means that only classes, which are important for the recommendation process are mentioned.

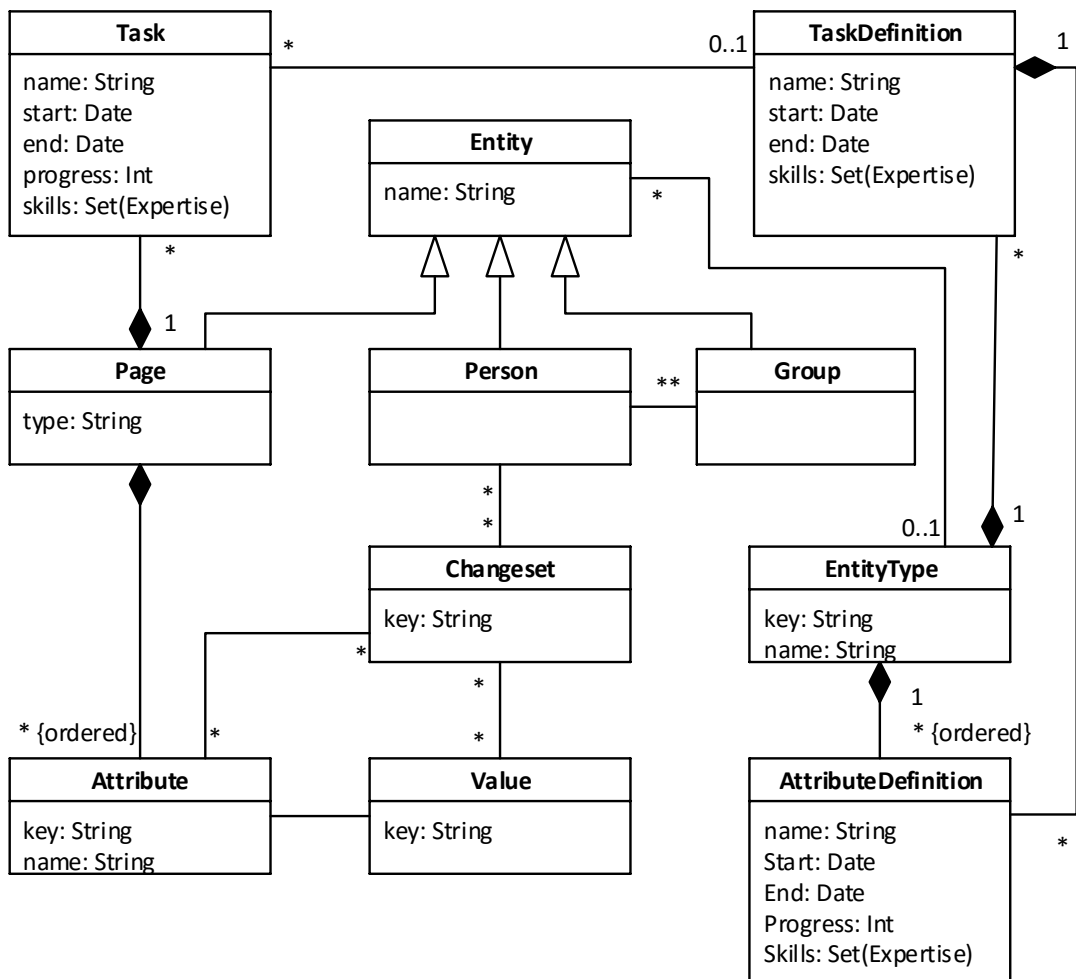


Figure 6.2.: Simplified conceptual class diagram of entire system

The model can be divided into two main parts with some other classes. On the right hand side there are the in 6.1 mentioned predefined classes which can be used to create new pages with predefined characteristics. All classes on the left hand side (including Task) represent the instantiated part of them. Subsequently the main classes are shortly described:

EntityType These are the possible types for entities like pages. Each EntityType can contain an ordered list of AttributeDefinitions.

TaskDefinition TaskDefinitions are patterns how task instances look like. Each task definition is assigned to an EntityType and can have AttributeDefinitions.

AttributeDefinition These are the templates for attributes. Every AttributeDefinition can be in at most one TaskDefinition and 1 EntityType.

Page Each Page has a specific type and can have an ordered list of Attributes and Tasks.

Attribute Attributes are the in subsection 6.1 mentioned attribute instances. An Attribute can have a value. Each Attribute is assigned to a specific Page and can be associated with a Task of this Page. Attributes themselves can be created on the basis of AttributeDefinitions of the EntityType. Additionally, they can be *free attributes*. These free attributes are created by end-users on a single page and do not descend from any AttributeDefinition.

Value A Value can be a simple String or e.g. a link to a Person or Page.

Task A Task is the instance of a TaskDefinition.

Changeset The Changesets are the history of the system. Every Changeset can include one or more changes. Changes can e.g. be Attribute changes and changes of the text or settings of a Page. Every Changeset can explicitly be assigned to a Person and an Entity.

Person A Person is an user who uses the system. It has different attributes and can be in one or more Groups.

Group A Group is an association of users. It can e.g. be used to give all members of it the same access rights for reading or writing specific objects in the system.

6.2.2. Supported Attributes

As mentioned before every page can have two kinds of Attributes: Attributes out of AttributeDefinitions and free attributes. Therefore, the first step is to identify if it is better to use only Attributes out of an AttributeDefinition or also free attributes. The advantage of the attributes with a definition is that every Page of the given type uses it and therefore they are predestined for predefined tasks. Beside of this the later created recommender may have a better performance by using only these attributes because they can directly be accessed from the EntityType. On the contrary some attributes could be evolved to be best practice over time although they are not in the type. Then it would be beneficial to consider them too. But it can also be that free attributes only exist on a single page. Then it makes no sense to include them in the recommendations and it would be enough to use only the AttributeDefinitions. Another possibility is that there exists a bundle of pages with no AttributeDefinitions.

As a consequence the existing data was reviewed to get an answer for this question. To get an overview about the used attributes, the percentages of pages of each type was identified. The following table shows a typical page with some of their attributes. The first column indicates the percentage of pages where the Attribute is used. The second

one includes the name of the Attribute and the last one describes the assignment of every Attribute. The table itself is sorted by the percentages in descendant order.

Table 6.5.: Attribute usages of "Publication".

Percentage of pages where Attribute is used	Attribute	Has AttributeDefinition
99,2 %	Year	✓
98,3 %	Title	✓
95,8 %	Key	✓
95,0 %	Citation	✓
89,9 %	Authors	✓
69,7 %	Address	✓
42,0 %	Type of publication	✓
39,5 %	Published in	×
17,6 %	Research project	✓
10,9 %	File	✓
24,4 %	Acronym	×
4,2 %	Type	×
2,5 %	Team members	×
...

Table 6.5 shows that there exist some free attributes which are used on many pages. In this example "Published in" would be a good choice to be used for the recommender, because it is used in more than a third of all pages. "Acronym" may also be interesting but "Type" and "Team members" are only used in a few Pages.

The results of another page type in table 6.6 indicates a similar result:

Table 6.6.: Attribute usages of "Article"

Percentage of pages where Attribute is used	Attribute	Has AttributeDefinition
100,0%	Key	×
100,0%	Year	×
98,9%	Citation	×
98,9%	Title	×
96,7%	Authors	×
90,0%	Address	×
66,7%	Published in	×
40,0%	Research project	×
7,8%	File	×
...

This example shows that there exist types, where almost all pages have the same attributes. Nevertheless, none of them has already been predefined in the page type. There-

fore, the recommender should be able to use this kind of attributes. However, some free attributes like "File" are not in the majority of the pages.

As a result of this perceptions we can conclude that free attributes take an important part in this system. To reach the best results, they should be used in the later recommender. However, not all free attributes should be considered. In order to filter infrequent free attributes a minimum threshold of pages is needed. This value should be relative to handle different varied numbers of pages. According to the results of the different pages a minimal threshold of 25% of all pages is reasonable.

6.2.3. Supported Pages

The next step is to analyze the pages. The aim of this step is to find out whether all pages are reasonable to be used. Assume there a lot of new pages for the type *Student Project*. Every of these pages have started with *Advisor* and *Supervisor* but have no other attributes changed yet. If these pages are not filtered out it would may falsify the results, because attributes from completed projects like *Final presentation slides* are only considered in less cases. This leads to a very low support of all patterns, which include those attributes in proportion to all patterns.

To handle the problem of incompleteness only pages should be used where the supported attributes are at least a specific minimum percentage filled. Supported attributes are all attributes which are either from a *AttributeDefinition* or free attributes which are in more than the minimum ratio of all pages.

The following figure 6.3 shows the completeness of the pages. The x-axis indicates the level how many supported attributes are used. For the supported attributes a minimum percentage of 25% was used. The y-value represents the number of pages where at least x percent of supported attributes are filled:

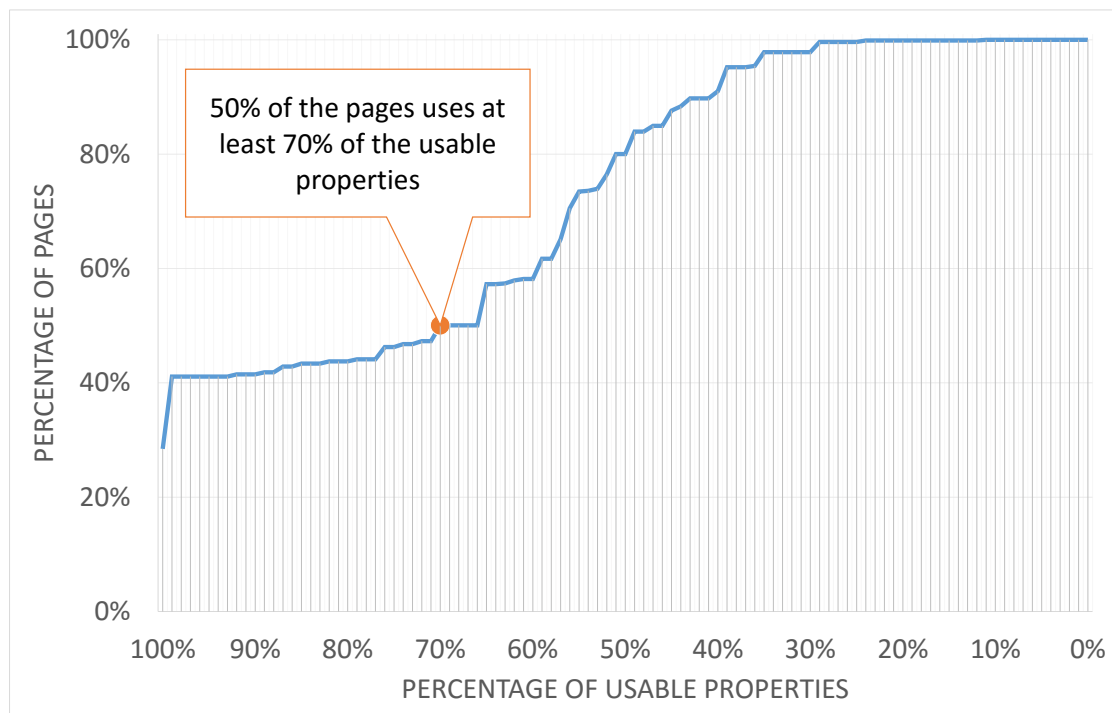


Figure 6.3.: **Completeness of pages.** The x-axis indicates the level how many supported attributes are used. The y-value represents the share of pages where at least x percent of supported attributes are completed.

To get a good trade-off between the existing pages and their completeness at least 50% of the supported attributes should be finished. The entire system does not have a huge amount of pages for all types. As a consequence at least 50% of the pages should be left after the filtering. This area correlates with the minimum completeness of 70%, where about 50% of the pages are left. The exact values for 6.3 can be found in table A.2.

6.2.4. Analysis of Change Histories

As last step of the data understanding phase the change histories are analyzed. Here the first step is to identify different kinds of ChangeSets by their creation and structure. Secondly the time differences between two subsequent ChangeSets are measured.

In the system all attribute or text changes are saved in Changesets. The following example in table 6.7 shows a simplified version of the changeset table:

Table 6.7.: **Simplified log example (Changesets)**. Simplification applies e.g. to unnecessary columns and to truncation of changes. The changes normally also contain the old value and other information. In addition some names like change types are adjusted. All ids (Changeset, Page, User) are formatted with the schema "(idtype)(id)".

ID	Page	Changes	User	Date	Time
cs1	p1	[[{type=AttributeChange, name=Title}]]	u1	13.08.2014	12:58
cs2	p1	[[{type=AttributeChange, name=Authors}, {type=AttributeChange, name=Year}]]	u1	13.08.2014	13:00
cs3	p2	[[{type=AttributeChange, name=Title}]]	u1	07.08.2011	17:00
cs4	p3	[[{type=AttributeChange, name=Address}]]	u1	07.08.2011	18:00
cs5	p2	[[{type=SimpleValue, name=PageName}]]	u2	06.08.2014	8:30
cs6	p2	[[{type=AttributeChange, name=Citations}]]	u2	06.08.2014	8:45
cs7	p2	[[{type=AttributeChange, name=File}]]	u2	06.08.2014	9:45
...

In this simple example you can see some important characteristics of the Changesets. Every Changeset can contain one (e.g. *cs1*) or more changes (*cs2*). These changes have a type which indicates the kind of change and the name what item has been changed. In addition to that every Changeset has explicitly been assigned to a specific user and page. Last but not least every Changeset has a date with time when it was done.

Looking at table 6.7 there exists ChangeSets with multiple changes (*cs2*) and with only a single one (*cs1*). Changes (especially on attributes) can be done in several ways:

Case 1: Batch Job Administrators can change several pages at the same time with a batch job.

Case 2: Edit Mode Users can set a single page in an edit mode. Then this user is able to change several attributes and save all changes afterwards. In this case all changes are saved in a single ChangeSet.

Case 3: Single Change Users can modify attributes individually. This results in a single Changeset for each change.

Looking at the example in table 6.7 the assumption raises that most Changesets contain only one change. This may happen because almost all users modify attributes as described in case 3. As a consequence simple approaches in literature would fail, because they use the changes belonging to a single log entry (Changeset) to identify patterns.

The next step is to determine the importance of this problem. Hence the ratio of Changesets with single attribute changes need to be identified. The following table 6.8 gives a short overview about this ratio:

Table 6.8.: **Share of Changesets with single/multiple Changes.** The table includes the changesets of all pages.

Changesets with single Change	83,93 %
Changesets with multiple Changes	16,07 %

According to this table less than $\frac{1}{5}$ of all Changesets contain multiple changes. That means mostly all Changesets contain only a single change. Therefore, using the logs without preparation or grouping of the Changesets would ignore case 3 and leads to inadequate results.

Nevertheless, in table 6.7 the Changesets cs1 and cs2 have been done within two minutes and could belong together. In common there exists the possibility to group the changes of Changesets, which have been done in a short time interval, to get rid of this problem and to get usable transactions. The key is to find an optimal interval. The following figure 6.4 gives an overview what time differences exist between subsequent Changesets within the same page.

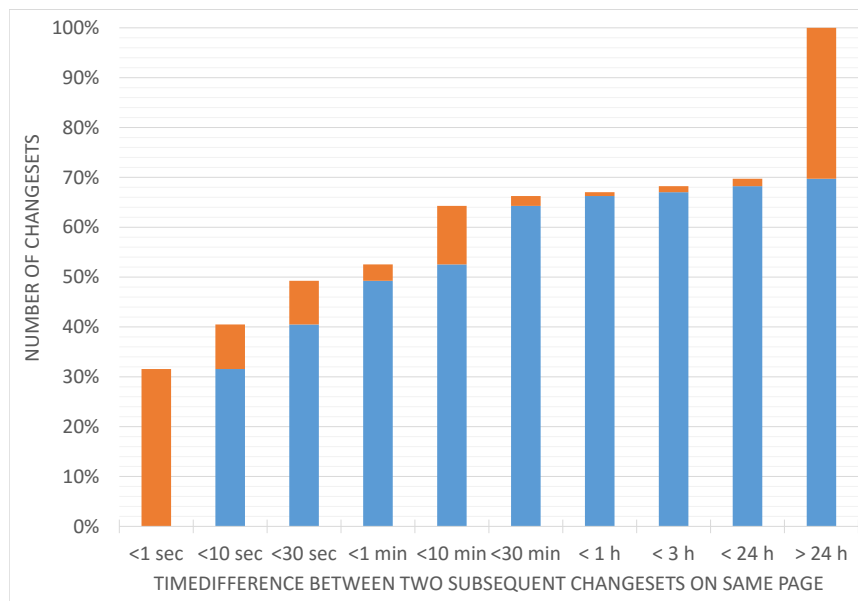


Figure 6.4.: **Time differences between subsequent changesets** within the same page. It contains as well Changesets with only one change as Changesets with multiple changes. The bars itself show the cumulated share of them within the given interval. The orange part of each bar indicates the additional number of Changesets in that time interval according to the last one.

This analysis shows that nearly a third of all Changesets are done within one second and can therefore be seen as concurrent changes. It also shows that after 10 respectively 30 minutes almost none new change happened. That means the limit is reached there. To sum this up changes on a page are typically done within in 30 minutes. This suits to the conception. There a typical user may do one or more tasks or prepare something for a specific topic and afterwards record his knowledge and results in the tool at one time. The person may correct some values afterwards at the same day but usually he or she does the next changes sometimes different (> 24 hours). A really surprising result is that $\frac{2}{3}$ of all changes have been done in 24 hours. That means that a a lot of pages are probably only edited once and there are no changes afterwards. The complete table with the absolute and relative values can be found in A.1.

6.3. Data Preparation

This section corresponds to the third part of the CRISP-DM model, the data preparation phase. This phase deals with the selection, the cleaning and the preparation of the data for the later modeling phase.

The preparation builds up on the results of the last chapter. There the considerations for the parameters have been done. The fact that there exist as much parameters even for the data preparation provides high flexibility for administrators with different data sources and can also be used to increase the performance of the system. The following parameters are used:

Max. number of pages Absolute value between 0 and ∞ . This parameter is an upper bound for the number of pages in the first iteration (to find the supported attributes).

Min. percentage for supported attributes Ranges from 0 to 1. This parameter is used for the free attributes. It reflects the lower bound of pages (in percent) which must contain the specific free attribute belonging to the supported attributes. A lower value leads to more supported attributes.

Min. completeness of page Ranges between 0 and 1. It is used as lower bound for the supported attributes of a page. A value of 0 means all pages are used.

Extended min. completeness of page Ranges between 0 and 1 with the condition "Min. completeness of page" > this. It is used as upper bound for the lower bound for the supported attributes of a page. It only comes into operation if "Min. completeness of page" filters as many pages that *Min. number of pages after filter* is not achieved. Then as many pages which fulfills this lower bound are added until it is reached.

Min. number of pages after filter Ranges between 0 and ∞ . Indicates how many pages should be left in minimum after filtering to get reasonable results afterwards.

Max. number of pages after filter Absolute value between 0 and ∞ , but should be \leq "Max. number of pages". It is an upper bound of pages to limit the time intensive procedure of catching the Changesets for the filtered pages afterwards.

Max. number of Changesets Absolute value between 0 and ∞ . It is use as upper bound for the retrieving of the Changesets. On the one side this parameter has a huge impact on the performance of the whole algorithm and should not be infinity. On the other side the algorithm itself need a sufficient amount of Changesets to get good results.

Max. time interval for Changeset grouping Absolute value between 0 and ∞ in seconds. It is used for the in 6.2.4 described time intervals between two subsequent Changesets.

6.3.1. Select

The selection of required data is the first step of data preparation phase. All we have in this step are the listed parameters and a specific page type.

1. Retrieve pages for type

The first step is to retrieve all pages for the type. This can be parametrized with *Max. number of pages* to get the newest x number of pages. This can be done with a simple database query and won't be explained more. These pages are used as basis to determine the supported attributes. For the subsequent algorithms the result of this step is called pages.

2. Find attribute occurrences

The next step is to identify the number of occurrences of the free attributes. This can be done by the following simple algorithm which count the free attributes in every page of pages:

Algorithm 6.1 Find attribute occurrences

```
1: Input: Collection pages for the pages of type
2: Output: 2-D array with occurrences (O) of attributes (A) as (A -> O)

4: for each page in pages do
5:   for each attribute in page.attributes do
6:     Increment occurrence of attribute in (A -> O)
7:   end for
8: end for
```

3. Find supported attributes

This step is to determine supported attributes. As reminder: Supported attributes are all attributes specified in the type and free attributes which occur in at least x pages whereas x is the parameter for *Min. percentage for supported attributes*:

6. Approach

Algorithm 6.2 Find supported attributes

```
1: Input: 2-D array attributeOccurrences (A → 0)
2: parameter minPercSuppAttr for [Min. percentage for supported attributes]
3: Output: Collection S[] of supported attributes

5: Add all predefined attributes of page to the supported attributes

7: //Add free attributes to supported attributes
8: for all attributes in attribute occurrences do
9:   currPercentage =  $\frac{\text{numberOfPages} * \text{attributeOccurrences}[\text{attribute}]}{100}$ 

11:   if currPercentage ≥ minPercSuppAttr then
12:     Add attribute to supported attributes
13:   end if
14: end for
```

The algorithm 6.2 first adds all predefined attributes of the type to the supported attributes and afterwards add all free attributes which satisfy the *Min. percentage for supported attributes*.

4. Filter pages

The aim of this step is to filter the existing pages of the type to get all pages which are at least minCompleteness ($\hat{=}$ *Min. completeness of page*) filled. This is necessary to get rid of the in subsection 6.2.3 described problem of incompleteness. In addition there are possibilities to prevent the filtering of too many pages. Therefore, if less than minNumberPagesAfterFilter ($\hat{=}$ *Min. number of pages after filter*) are left after filtering the algorithm try to do it once again with minCompletenessExt ($\hat{=}$ *Extended min. completeness of page*). To get an upper bound (because of time) a upper limit for this function can be set with maxNumberOfPagesAfterFilter ($\hat{=}$ *Max. number of pages after filter*). Although this step semantically belongs to the next subsection (Clean data) it is illustrated at this position, because the results are important for the next step.

Algorithm 6.3 Filter pages

```
1: Input: Collection pages for the pages of type
2: Parameter minCompleteness for [Min. completeness of page]
3: Parameter minNumberPagesAfterFilter for [Min. number of pages after filter]
4: Parameter minCompletenessExt for [Extended min. completeness of page]
5: Parameter maxNumberOfPagesAfterFilter for [Max. number of pages after filter]
6: Output: Collection filteredPages with all pages after the filter

8: List<Page> filteredPages

10: // Try filtering with minCompleteness
11: for each page in pages do

13:   currPerc =  $\frac{\text{Number of supported attributes in page}}{\text{Number of all supported attributes}}$ 

15:   if currPerc ≥ minCompleteness then
16:     add page to filteredPages
17:   end if

19:   //terminating condition (max number of pages reached)
20:   if filteredPages.size ≥ maxNumberOfPagesAfterFilter then
```

```

21:     return filteredPages
22:   end if
23: end for

25: //terminating condition (min number of pages reached)
26: if filteredPages.size ≥ minNumberPagesAfterFilter then
27:   return filteredPages
28: else
29:   empty filteredPages
30: end if

32: // Try filtering with minCompletenessExt
33: for each page in pages do

35:   currPerc =  $\frac{\text{Number of supported attributes in page}}{\text{Number of all supported attributes}}$ 

37:   if currPerc ≥ minCompletenessExt then
38:     add page to filteredPages
39:   end if

41: //terminating condition (max number of pages reached)
42: if filteredPages.size ≥ maxNumberOfPagesAfterFilter then
43:   return filteredPages
44: end if
45: end for

47: //terminating condition (min number of pages reached)
48: if filteredPages.size ≥ minNumberPagesAfterFilter then
49:   return filteredPages
50: else
51: // not reached -> return all pages without any filter
52:   return pages
53: end if

```

The result of this step is called `filteredPages`.

5. Retrieve Changesets

The last step of this section is to get Changesets of the filtered pages. Therefore, a simple database query can be used. This query should be extended that the resulting list of changesets is ordered by page and by time in descending order to get the newest ones. Because the number of Changesets has a huge effect on the execution time of the whole algorithm it must have the limit [Max. number of Changesets].

6.3.2. Clean Data

The algorithms for the cleaning of the data can be found in 6.3.1 and in 6.3.3. So the algorithm will not be described here explicitly.

Assume that the pages (p1) and (p2) reach the minimal completeness and the supported attributes are: *Title*, *Authors*, *Year*, *Address* and *File*. Beginning with the table 6.7 the following cleanings are made in the Changesets before the modeling starts:

Table 6.9.: Cleaned log.

ID	Page	Changes	User	Date	Time
cs1	p1	[[{type=AttributeChange, name=Title}]]	u1	13.08.2014	12:58
cs2	p1	[[{type=AttributeChange, name=Authors}, {type=AttributeChange, name=Year}]]	u1	13.08.2014	13:00
cs3	p2	[[{type=AttributeChange, name=Title}]]	u1	07.08.2011	17:00
cs4	p3	[[{type=AttributeChange, name=Address}]]	u1	07.08.2011	18:00
cs5	p2	[[{type=SimpleValue, name=PageName}]]	u2	06.08.2014	8:30
cs6	p2	[[{type=AttributeChange, name=Citations}]]	u2	06.08.2014	8:45
cs7	p2	[[{type=AttributeChange, name=File}]]	u2	06.08.2014	9:45
...

1. **Filter sites with less than *Min. completeness of page percent* filled:** The page (p3) does not reach this lower bound. As a result all Changesets belonging to this page are filtered out. In this example Changeset 4 (cs4) is affected. See point 4 of subsection 6.3.1 for more details.
2. **Filter all changes which do not modify an attribute:** The later algorithm only needs changes with the type *AttributeChange*. In this example Changeset (cs5) has no such change and is therefore been affected.
3. **Filter all changes which are no supported attribute changes:** The attribute *Citations* is a free attribute and does not reach the parameter *Min. percentage for supported attributes*. Therefore, all changes with this attribute (only in Changeset (cs6)) are sorted out from the associated Changeset.

6.3.3. Construct Data

The aim of this subsection is to solve the in subsection 6.2.4 mentioned problem that most of the Changesets only contain a single change. Without the construction of reasonable groups five nearly concurrent changes on the same pages could lead to five different Changesets. This problem can be prevented by grouping them into transactions. In addition to that the algorithm should not only group them by the time difference but also by the groups of the acting users.

To sum up we already have retrieved all necessary data and have done some cleaning. The result of this algorithm will be grouped sets mapped to their count. This map speeds up the later frequent pattern mining algorithm, because they must not be counted repeatedly. Additionally, the count can be better weighted in relation to the groups.

The first step of the algorithm has the parameter timeInterval1 ($\hat{=}$ *Max. time interval for Changeset grouping*) which indicates the maximal time difference between two Changesets

to belong together. The Changesets of the pages are sorted in descendant order of the timestamp and already cleaned which means they are not empty and only contain necessary changes.

Algorithm 6.4 Group changesets to transactions

```

1: Input: Collection filteredPages for already filtered pages
2:   Parameter timeIntervall for [Max. time interval for Changeset grouping]
3: Output: 2-D Array transactionRatio which contains a weighted count C of each
   ↪ Transaction T (T -> C)
4: for each page in filteredPages do
5:   create new 2-D Array transactionBuffer which later contains the transactions T of
   ↪ each group G (G -> T)
6:   create new 2-D Array transactionRatioBuffer which later contains the ratio R of each
   ↪ group G (G -> R)
7:   create new valCount (number of changesets in current interval)

9:   lastDate = page.changesets[0].timestamp - timeIntervall

11:  for each Changeset cs in page.changesets do
12:    currentDate = cs.timestamp

14:    if currentDate is before lastDate then
15:      // out of intervall -> new transaction
16:      // add buffers to result
17:      constructTransaction(transactionBuffer, transactionRationBuffer, transactionRatio
   ↪ , valCount)
18:    else
19:      // in intervall -> same Transaction, nothing to do
20:    end if

22:    //assign weighted changes to group
23:    for each group in cs.person.groups do
24:      Add all cs.changes to transactionBuffer[group]
25:      Add  $\frac{1}{\text{Number of groups of user of cs}}$  to transactionRatioBuffer[group]
26:    end for

28:    Increment valCount

30:    lastDate = cs.timestamp
31:  end for

33:  //next page: add buffers to result
34:  constructTransaction(transactionBuffer, transactionRationBuffer, transactionRatio,
   ↪ valCount)
35:  }
36: end for

```

The algorithm 6.4 iterates through all pages and their Changesets. For every Changeset all changes are assigned to the groups of the associated user and weighted. After all changes in a page (line 34) and if the current Changeset extends the given time interval (line 17) the currently buffered transaction is created. Listing 6.5 illustrates how the second step (constructTransaction) works:

6. Approach

Algorithm 6.5 Construct transactions

```

1: Input: 2-D Array transactionBuffer with transactions T of each group G (G -> T)
2:   2-D Array transactionRatioBuffer with the ratio R of each group G (G -> R)
3:   2-D Array transactionRatio which contains the weighted count C of each Transaction T
   ↪ (T -> C)
4:   Counter valCount which indicates the number of changesets which are combined
5: Output: updated transactionRatio

7: for each group of transactionBuffer do
8:   var currentTransaction = transactionBuffer[group]
9:   Divide associated ratio in transactionRatioBuffer[group] with valCount
10:  Add transactionRatioBuffer[group] to existing transactionRatio[currAttrs]
11: end for

```

There the question arises why a weighting according the groups is necessary. Imagine user A is in two groups (g1, g2) and perform some changes (Title, Authors). User B is in six groups and also modify some attributes (Title, Year). No group of them is overlapping and they are in the same time interval. Then if we not weight them and increase the count for every transaction of every group, user A would only create two transactions but user B would create six transactions. This means changes of user B weight three times as much as changes of user A. With the in listing 6.5 proposed weighting, every transaction will only receive the count $\frac{1}{|usergroups|}$ normalized by the number of Changesets. After the second part of the algorithm the count of both attribute groups will be increased in sum by one. That is because every of the six groups of user B received $\frac{1}{6}$. If these two users would have one overlapping group (g2) the weight of this group would be summed up. Figure 6.5 illustrates a minimal example to show this case.

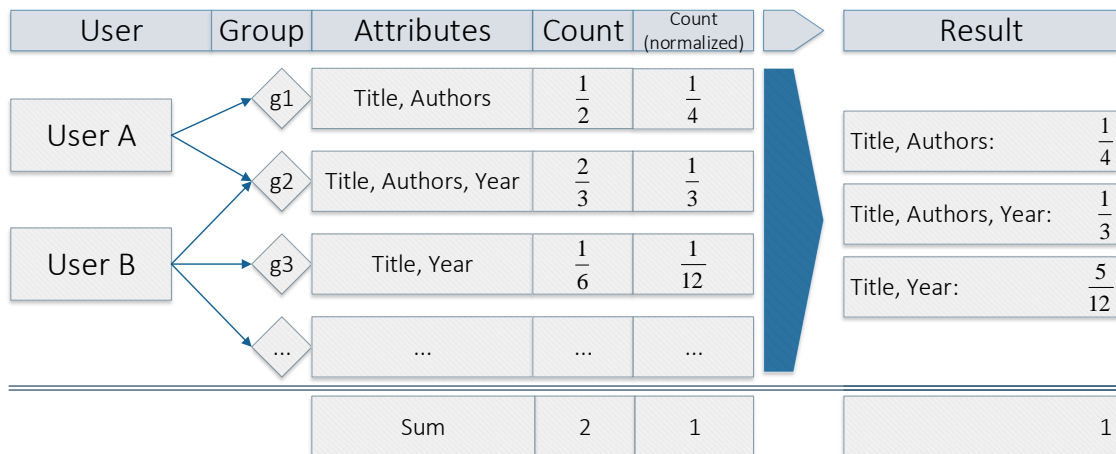


Figure 6.5.: **Example of group creation.** User A changed [Title, Authors] and is in the two groups (g1) and (g2), user B changed [Title, Year] and is in the two groups (g2) - (g7). All changes are in same time interval.

The graphic shows the users with their groups. Right of them there are the attribute groups (transactionBuffer in algorithm) with their count (transactionRatioBuffer). In addition to that the resulting map over all transactions is shown beneath "result". According to the algorithm the specific count for the groups of user A is $\frac{1}{2}$ because he has two

groups. The specific count for each group of user B is $\frac{1}{6}$. Therefore, the attribute changes of group (g1) only get the $\frac{1}{2}$ of user A . The changes of group (g2) get $\frac{1}{2} + \frac{1}{6} = \frac{2}{3}$ and so on. The next step is to normalize these values. Without this step the transaction would get as many counts as Changesets are combined (two in this case). This contradicts with the aim, that every transaction should only count as one. Therefore, these values are normalized by dividing their temporal count by the number of transactions which have been combined. The resulting attribute groups are afterwards the already mentioned sum of each different set of attributes e.g. for [Title, Year]: $\frac{1}{12} + \frac{1}{12} + \frac{1}{12} + \frac{1}{12} + \frac{1}{12} = \frac{5}{12}$.

6.4. Modeling

As a result of the previous sections we already have a fully cleaned 2-dimensional array, which maps every transaction to a specific count for all filtered pages. The next step is to perform a frequent pattern mining algorithm for them.

In subsection 5.1 many different approaches are shown. The performance benefit for these algorithms justified by less database scans take no effect in this case. This results from the need to combine the Changesets to transactions. Consequently, we already have direct access to them similar to the *AprioriTID*. For this reasons the *Apriori* algorithm has been chosen to prevent calculating all combinations. Therefore, the first step is to calculate the support and to create groups which fulfill the min support. Afterwards groups with the highest score can be built. The subsequent algorithms have the following parameters:

Min. support Ranges from 0-1. It is the lower bound for the support of an itemset (attribute group). The base of this value is the sum of all attribute occurrences. Attribute groups with a lower support than this are sorted out.

Min. score for Group Recommendation Ranges from 0-1. Is used for group recommendations. Attributes are only recommended to existing groups if they have at least this score. In addition to that its used as a minimum for complete group suggestions.

Max. adjustment factor Ranges from 0-1. The attributes of a page are in a user specified order. Therefore, attributes with a similar position can be ranked higher to each other. This value specifies the amount the score can be increased or lowered.

6.4.1. Supported Attribute Sets

The first step is to find the frequent attribute sets. For this purpose, the in subsection 3.3.2 introduced *Apriori* algorithm is used with the given min. support. Assume that the resulting transactions are stored in a 2-dimensional Array supportMap which maps all frequent transactions (which satisfy min support) to their support.

6.4.2. Score

An important step is the calculation of the score. The score is a mixture of the confidence with a small adjustment by the order of the attributes on the page. The calculation

6. Approach

of the confidence is not shown here explicitly. It can be found in section 3.2. The result is called confidence in the algorithm. The maximal adjustment is parametrized with maxFactor ($\hat{=}$ *Max. adjustment factor*). Let maxDistance be the max distance between two *AttributeDefinitions* in the defined order. The maximal distance is therefore the number of *AttributeDefinitions* in the page type. Free attributes are not ordered and therefore no minimal distance can be found for them. The adjustment for the order is only calculated for the "nearest" *AttributeDefinition* to the tested attribute.

Algorithm 6.6 Find minimum distance

```
1: Input: Transaction origin which contains attributes
2:   Attribute attribute which should be tested
3: Output: Real number  $\text{minDistance}$  which indicates the minimal distance of the tested
    $\rightarrow$  attribute to the transaction

5: if attribute is free attribute OR  $\text{maxDistance} < 2$  then
6:   return MAX_VALUE
7: end if

9: for each Attribute attr in origin do
10:  if attr is free attribute then
11:    //no distance measurement possible
12:    continue
13:  end if

15:   $\text{minDistance} = \min(\text{minDistance}, \text{distance}(\text{attr}, \text{attribute}))$ 
16: end for
```

The algorithm first checks whether the tested attribute is a free attribute and if there exists at least two *AttributeDefinitions* in the type. This is done by checking the maxDistance . If there is only one *AttributeDefinition*, MAX_VALUE is returned. Alternatively, the attribute would always be next the other, which results with a positive adjustment. Afterwards the minimum distance (minDist) between all the attributes and the tested attribute is calculated. With this value the adjustment can be calculated with the following equation:

$$\text{adjustment} = \begin{cases} 0, & \text{if } \text{minDist} = \text{MAX_VALUE} \\ \text{maxFactor} - \left(\frac{2 * \text{maxFactor} * (\text{minDist} - 1)}{\text{maxDist} - 1} \right), & \text{otherwise} \end{cases} \quad (6.1)$$

As a result the adjustment is inside $[-\text{maxFactor}, \text{maxFactor}]$.

The following example illustrates this process.

Assume the parameters $\text{maxFactor} = 0.2$ and $\text{minDistance} = 2$ and we have seven *AttributeDefinitions* in the current type, which leads to $\text{maxDistance} = 7$. Therefore, $\text{minDist} \neq \text{MAX_VALUE}$ and we are in the *otherwise* part. The resulting equation with inserted values look like: $0.2 - \left(\frac{2 * 0.2 * (2 - 1)}{7 - 1} \right) = 0.13$.

The result can be divided into several cases:

1. $\text{minDistance} = 1$

This represents the best possible case, because then the attribute is directly aside the group. As a result the part in brackets will return 0 and the adjustment is the max possible value maxFactor .

2. $minDistance = maxDistance$

This illustrates the worst case, because then the attribute has the maximal possible distance in the order of the group. The following equation shows this circumstance:

$$adjustment = maxFactor - \left(\frac{2 * maxFactor * (maxDistance - 1)}{(maxDistance - 1)} \right)$$

$$adjustment = maxFactor - (2 * maxFactor)$$

$$adjustment = -maxFactor$$

The result is therefore the minimal possible value $-maxFactor$

3. $minDistance = average(= 4)$

If the distance is the average distance, which is in the before mentioned example $4(= \frac{1+7}{2})$, the part in brackets will be $\frac{2 * maxFactor * (4-1)}{(7-1)} = maxFactor$. Therefore, there is no adjustment ($adj. = maxFactor - maxFactor$) and the result is the same.

To get a more meaningful score for all cases, the average confidence of both directions is used. This ensures that the attribute fits to the group as well that the group fits to the attribute. Assume the group already contains the items A and B and the score for C ($\{A, B\} \Rightarrow \{C\}$) should be calculated. Then the average of the confidence from ($\{A, B\} \Rightarrow \{C\}$) and ($\{C\} \Rightarrow \{A, B\}$) is used for this calculation.

With the adjustment the score can be computed with:

$$score = max(min(avgConf * (1 + adjustment), 1), 0) \quad (6.2)$$

This formula ensures that the score is in the bounds $[0, 1]$.

6.4.3. Group Recommendations

The aim of this group recommendation algorithm is to show which attribute fits to existing task definitions (Add Recommended Attribute to Existing Task Definition (Use Case 3)) and what attributes fit together to create a new task definition (Create New Task Definition with Attributes (Use Case 1)). For the suggestion of new tasks only attributes, which are not recommended to an existing one are used. Therefore, the algorithm for use case 1 depends on use cases 2, 3 and 4 and is explained together with them. The part where the use case 2, 3 and 4 ends is marked with **Breakpoint**.

Algorithm 6.7 Find group recommendations

- 1: **Input:** Collection existingTA with the existing Transactions
- 2: Collection supportedAttributes with the supported attributes of type
- 3: Parameter minScore for [Min. score for Group Recommendation]
- 4: **Output:** Collection result with possible TaskRecommendations

- 6: Add all existingGroups to result
- 7: Create Collection remainingAttributes and add all attributes from the
 - ↪ supportedProperties which are not already in an existingGroup

- 9: Set boolean inTask = true **for** all attributes in result, =false **for** all attributes in
 - ↪ remainingAttributes

6. Approach

```
11: //check if all supportedAttributes are already in a task definition
12: if remainingAttributes.size == 0 then
13:   return
14: end if
15: //if there is no existing group add a new group with one attribute
16: if result is empty then
17:   Breakpoint: return if only attribute recommendations wanted
18:   result.add(remainingAttributes.next)
19: end if

21: while remainingAttributes.size > 0 do
22:   Init maxScore with 0
23:   Init pointer maxScoreGroup
24:   Init pointer maxScoreAttribut

26:   //find attribute/group combination out of remaining attributes with highest score
27:   for each Attribute currAttribute in remainingAttributes do
28:     for each AttributeGroup currGroup in result do
29:       currScore = getScore for currAttribute in currGroup
30:       //test if current score is better than previous maxScore and if new group is
           ↪ frequent
31:       if currScore > maxScore and supportedAttributes contain {currGroup ∪ currAttr} do
32:         buffer maxScoreAttribute, maxScoreGroup and maxScore with currAttribute /
33:         currGroup / currScore
34:       end if
35:     end for
36:   end for

38:   if maxScoreAttribute found then
39:     //add maxScoreAttribute to group if minScore is reached, else create new group
40:     if maxScore ≥ minScore then
41:       Add maxScoreAttribute to maxScoreGroup
42:     else
43:       Breakpoint: return if only attribute recommendations wanted
44:       Add new group to result which contains maxScoreAttribute
45:     end if

47:     Remove maxScoreAttribute from remainingAttributes
48:   else
49:     Breakpoint: return if only attribute recommendations wanted
50:     Add new group with remainingAttribute.next and remove this one from
           ↪ remainingAttributes
51:   end if
52: end while
```

Algorithm 6.7 first initializes the values and adds some termination conditions until line 19. After that it try to assign all remaining attributes to the best matching group. There first the best group / attribute combination is searched. Afterwards it tries to add the best matching attribute if the calculated score is high enough ($> \text{minScore}$) to the associated group. If only possible attribute recommendations should be made (Use Case 3 from subsection 6.1.3) then the marked **Breakpoints** in line 17, 43 and 49 should be activated as termination conditions. Then no new groups are created. If all use cases should be served these have to be deactivated. Consequently, new groups are instantiated whereby all attributes of them are flagged with `inTask=false`.

6.4.4. Individual Score

In the previous subsections the structure for the recommendation of new TaskDefinitions and AttributeDefinitions have been created (use cases 1,3). For the both use cases *Remove*

Attribute from Task Definition (Use Case 4) and *Remove Existing Task Definition (Use Case 2)* the user needs a proper indicator which one he should remove and which one not. This indicator, the already mentioned *Score* can also be helpful for the other 2 use cases to show e.g. which attribute is reasonable to add. The following algorithm 6.8 shortly illustrates how this can be done. The input *taskRecommendations* represents the output of *Find group recommendations* (Algorithm 6.7).

Algorithm 6.8 Calculate scores

```

1: Input: Collection taskRecommendations with all TaskRecommendations (containing as well
   ↪ existing TaskDefinitions as recommended one)
2: Output: Updated collection taskRecommendations

4: for each taskRecommendation in taskRecommendations do
5:   Create new collection attributesFromTask and add every attribute in attributeGroup
   ↪ where inTask==true

7:   for each attribute in taskRecommendations.attributes do
8:     if attributesFromTask.size > 0 then
9:       Calculate and set score of attribute for attribute in attributesFromTask
10:    else
11:      Calculate and set score of attribute for attribute in taskRecommendations.
   ↪ attributes
12:    end if
13:  end for
14: end for

```

As mentioned this algorithm can be used for all four use cases. If only the existing groups are wanted, the result of 6.4.3 can be used until the Breakpoints occur. Else the whole result can be put in. The algorithm differs between attributes which are already in a TaskDefinition and the recommended ones. Therefore, all attribute recommendations only refer to already existing attributes in the task definitions. If it is an empty TaskDefinition or if it is a recommended group then obviously it has no attributes which are in the task and therefore the score refers to all (recommended) attributes of this group.

Part IV.

Technical Approach

7. Front End

This chapter illustrates the visible result of the shown concepts. Initially, the position of the client in the *SocioCortex* eco system is shown in order to understand the background. Subsequently the user interface is described by the underlying use cases.

7.1. The Client in the Context of the SocioCortex Eco System

The SocioCortex (see section 6.1 for explanation) server will have different data and content sources and is able to deal with different clients like the normal "SC Generic Web Client" and the "SC Web IME". The *SocioCortex* server itself offers a RESTful API to provide the clients with the persistent data. Figure 7.1 illustrates the structure:

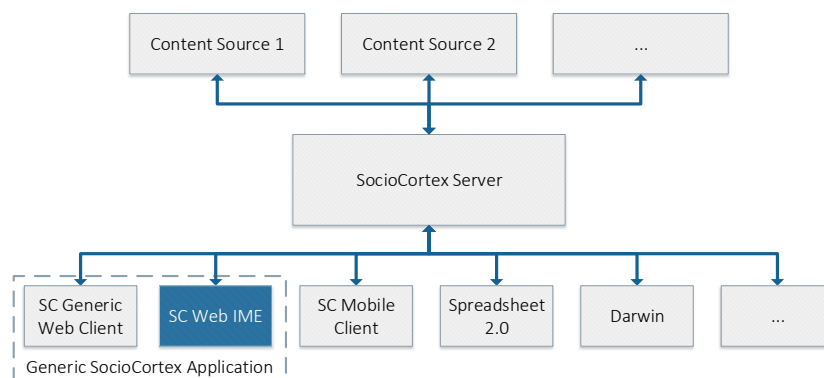


Figure 7.1.: **The SocioCortex eco system.** Adapted from [SEBIS chair, 2015]

The front end implementation of this thesis is integrated into the existing *SC Web IME* which is marked blue in the graphic. This *Integrated Model Environment (IME)* for social enterprise applications helps even non-modeling experts to create domain specific data models. The client itself builds on AngularJS¹ and is presented in Material Design². [Schalkamp, 2015]

AngularJS is a JavaScript framework, which is originally designed to build one-page web applications. This framework provides code templates and commands for an easy usable data binding in HTML pages and to do things like filtering on them. Material Design is only a styling guideline from Google and no concrete implementation. As implementation Angular Material³ is used. Angular Material is a lightweight component

¹<https://angularjs.org>

²www.google.com/design/spec/material-design

³<https://material.angularjs.org>

suite including mainly UI components which are based on material design and work with AngularJS.

All generic clients are built on *scAngular*, which already provides an abstraction for the REST API of the server. This includes e.g. the CRUD (create, retrieve, update, delete) operations of the SocioCortex server. The response of this calls is in JSON format.

All methods and variables which are shown in this chapter refer to the front end controllers of the application.

7.2. User Interface

The following figure 7.2 shows the page. The site can be split into the navigation on the left hand side and the content on the right one.

The content can be divided into three different recommendation types which contain the four use cases. Every use case is explicitly explained in the subsections 7.2.1.

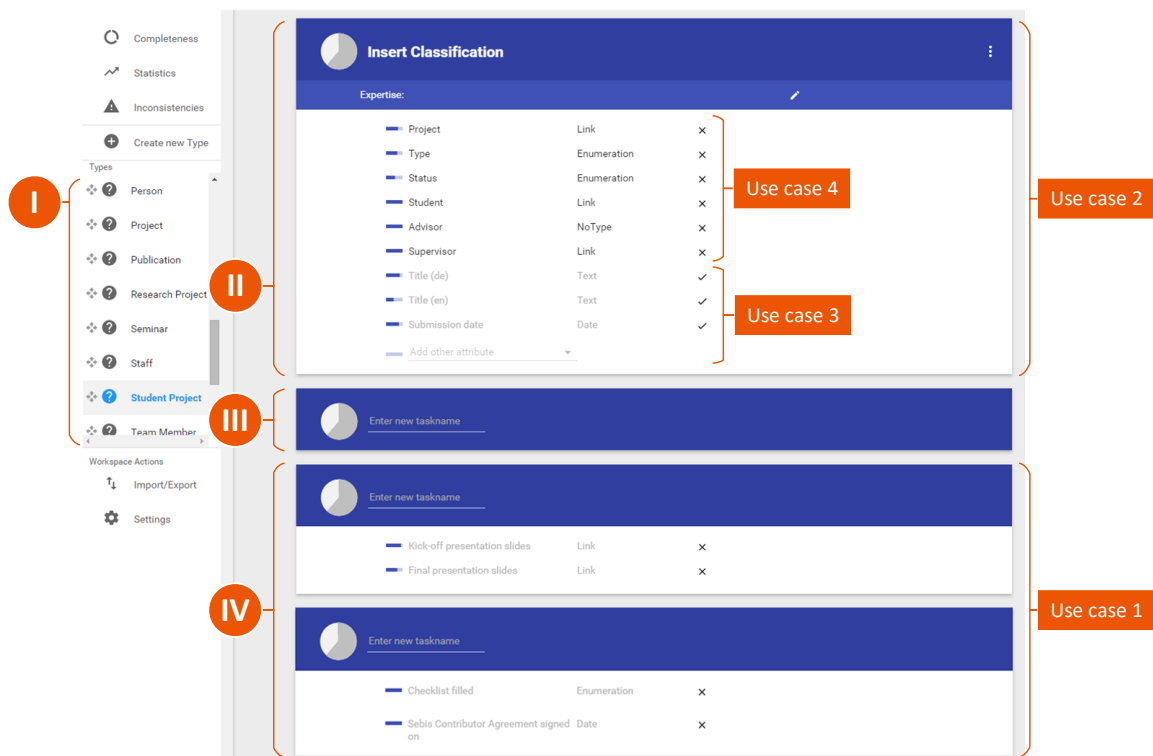


Figure 7.2.: Overview Frontend

Every type has the following basic elements:

- I This section lists all EntityTypes related to the current workspace. After one of them has been selected, the associated content is shown on the right side.

- II In the first block the existing TaskDefinitions are shown. This includes as well already assigned AttributeDefinitions as the recommended ones. Further details can be found in subsection 7.2.2, 7.2.3 and 7.2.4.
- III This block allows new TaskDefinitions without any recommendations.
- IV The last block allows a user to create new TaskDefinitions including recommend attributes. In subsection 7.2.1 more details can be found.

The following subsections describe the in section 6.1 introduced use cases step by step. Therefore, every use case will be summed up in the beginning. Afterwards all associated parts of the user interface are explained.

7.2.1. Create new Task Definition with Attributes (Use Case 1)

The first use case deals with the creation of a new TaskDefinition with recommended attributes. Therefore, suitable attributes are shown with the score indicator:

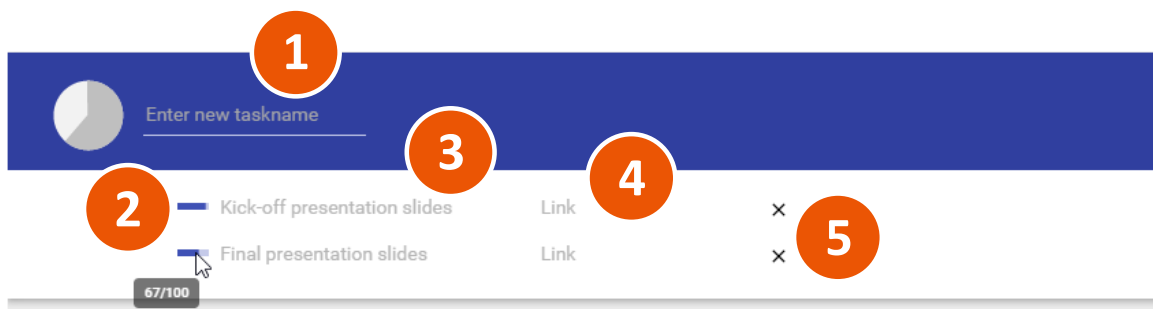


Figure 7.3.: Recommended TaskDefinition with attributes

Each of this kind of blocks illustrate a TaskRecommendation with the in section 7.3.2 shown structure. In details the page snippet has the following elements:

- ① In the top the new name of the TaskDefinition has to be inserted.
- ② Every line represents a single RecommendedAttribute. The fill state of the linear indicator shows the score of each of them. If a user hovers on an indicator the value of it is shown in a range from 0 to 100.
- ③ Next to the indicator, the name of the attribute is shown.
- ④ Fourthly, there is the kind of the attribute. If it is a free attribute, there exist no such kind. Consequently, the user has the availability to choose one in a select box. If there exists at least one free attribute where the user has not chosen a kind it can not be created.
- ⑤ Lastly there is the availability to remove a RecommendedAttribute.

After the user adds a name for the TaskDefinition a create button appears on the right hand side and the new TaskDefinition can be created. Therefore, the createTask(...) method of the controller is called with the new task name, the associated attributes and the type of the page. This method is explicitly explained in subsection 7.3.2.

There is also the possibility to create a task without attributes. Then the user has to use either (III) of figure 7.2 or to remove all recommended attributes before he creates it.

7.2.2. Add Recommended Attribute to Existing Task Definition (Use Case 3)

The maybe most common use case is to add recommended attributes to an existing task definition. Therefore, the TaskRecommendations for existing TaskDefinitions are positioned in the top of the page how (II) of figure 7.2 illustrates. Here the user got recommendations for possible attributes and the possibility to add them with one click. All information about the basic structure of this element can be found in subsection 7.2.1. The following figure 7.4 gives an overview about extended features of this element:

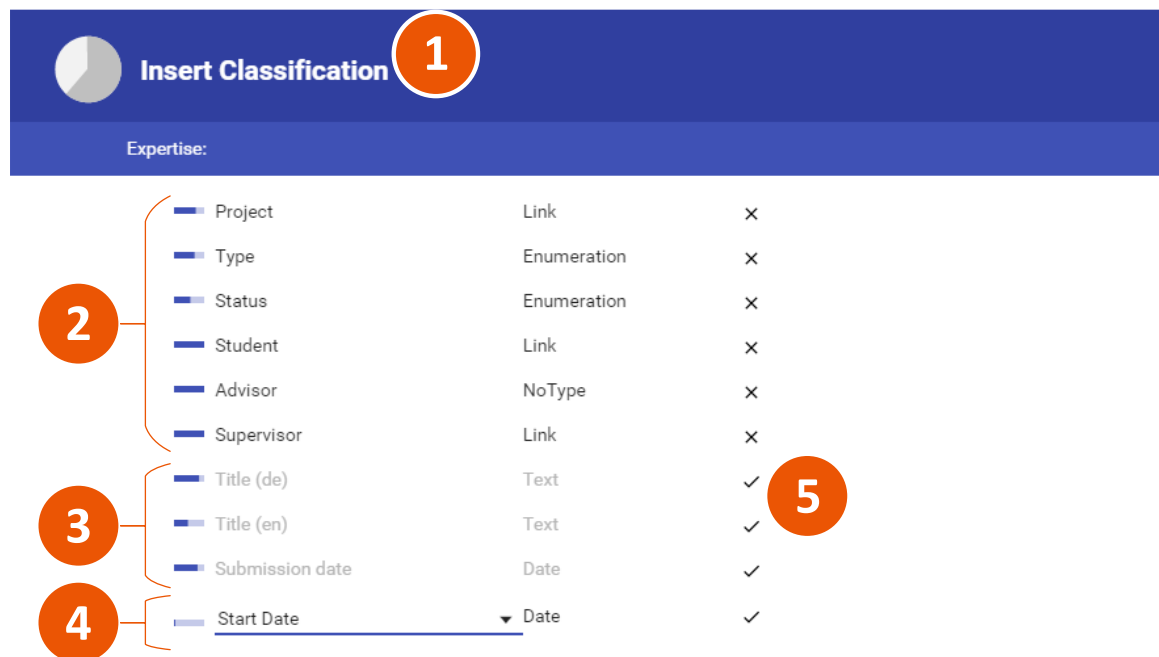


Figure 7.4.: Existing TaskDefinition with recommendations

- (1) In the top the name of the existing TaskDefinition is shown.
- (2) The first part contains all already associated attributes. These attributes are emphasized with their color to show this state.
- (3) The second part are the recommend ones. Here the font color indicates that these are only recommendations and that they are not linked with the TaskDefinition.

- ④ Lastly there is the possibility to add other attributes, which are not recommended to the TaskDefinition. Here every attribute, which does not belong to any Task-Definition and is not already recommended to the current one can be selected. If the chosen attribute is a free attribute the kind has to be selected afterwards, too. The possible attributes in the selection menu are sorted by their score.
- ⑤ In the fourth column the buttons to add and remove specific attributes are shown. This can either be a cross (×) to delete already associated attributes or a checkmark (✓) to create the recommended ones.

After a user has clicked to add the specific attribute to the given TaskDefinition the `addGenericAttributeToTask(...)` method is called with the specific attribute. Details about this method can be found in subsection 7.3.2.

7.2.3. Remove Attribute from Task Definition (Use Case 4)

The probably second most common use case is to remove attributes from an existing task definition. This use case can be done in the block of the existing TaskDefinitions (see figure 7.4 for more information). There the user can check the score of every existing attribute. If it is beneath a certain value it can be deleted by clicking on the cross ⑤. Afterwards `removeAttributeFromTask()` is executed which simply calls the related REST API. After this the TaskRecommendations are recalculated under consideration of the now not associated attribute.

7.2.4. Remove Existing Task Definition (Use Case 2)

Sometimes users want to remove existing TaskDefinitions. This can e.g. either be if there is no need for the task or the attributes completely do not fit together. This can be checked by the score of every attribute. Then the user can hover over the button in the top right. Afterwards the remove button ① appears:



Figure 7.5.: Remove of an existing TaskDefinition

As soon button ① is clicked the method `removeTask()` is executed. Then the specified TaskDefinition is deleted for this type of page. In addition to that every attribute which was linked to it is unlinked. Afterwards all TaskRecommendations are recalculated under consideration of the now not associated attributes.

7.3. Technical Structure

The front end implementation can be divided into the structure of the files and the used AngularJS controller. To get a short overview about the realizations subsequently the new created or changed files are shortly introduced. For reasons of clarity only most important ones are explained:

7.3.1. File Structure

`type.task.html`

This newly created file is the template of the content part of the page from figure 7.2. The template itself is written in HTML with additional AngularJS specific elements and expressions, which are rendered dynamically. In details it is a so called partial which is included into `type.html` with `ng-view`.

`type.task.controller.js`

This file is the AngularJS controller related to the template. The controller is used for expressions and contains the variables. These variables are bound to specific elements in the template where they are dynamically rendered. All used methods of the new page are included in this file.

`ime.routing.config.js`

Here the configuration for every route can be defined. Therefore, it was extended by the new view. This contains the route with the before mentioned template and controller.

`app.css`

The `app.css` includes all the custom styles of the page. It was extended with custom design styles for the page.

7.3.2. Controller

The controller contains the variables and methods which are used. Subsequently the most important ones are mentioned as outline:

`data`

The `data` variable holds all the results from the backend. This contains as well the existing `TaskDefinitions` as the recommended ones.

`loadingActive`

This variable indicates if the frontend is in a loading state. Then the use of the recommendations is deactivated.

`refreshData()`

This method reloads all calculations and recommendations.

`addGenericAttributeToTask()`

Adds an attribute to an existing task. The subsequent pseudo code shows simplified this process:

Algorithm 7.1 Add generic attribute to TaskDefinition

```

1:  if attribute has id then
2:    // attribute already has AttributeDefinition
3:    addAttributeToTask(attribute)
4:  else
5:    // attribute is free attribute
6:    if attribute has no name || attribute has no kind then
7:      show error message
8:    else
9:      var attributeDefinition = createAttributeDefinition(attribute)
10:     addAttributeToTask(attribute)
11:    end if
12:  end if

```

This algorithm checks, if the attribute is a free attribute or not. If it is an existing AttributeDefinition it can simply be added to the TaskDefinition. In the other case it is first checked if the user the user has chosen a kind for the attribute. Then the free attribute can be converted to an AttributeDefinition by calling the REST API. Afterwards the attribute is added to the TaskDefinition like in the first case.

`removeAttributeFromTask()`

Remove a specific attributeDefinition from an existing task.

`createTask()`

The `createTask()` method of the controller creates a new TaskDefinition with all recommended attributes. It is called with the new task name, the associated attributes and the type of the page. The following pseudo code illustrates the basic steps:

Algorithm 7.2 Create task definition with attributes

```

1:  var taskDef = createTaskDefinition(taskname, type)
2:  for each attribute in attributes do
3:    var attr = addGenericAttributeToTask(task.id, attribute.name, attribute.kind,
4:    ↪ attribute.id)
5:  end for

```

Firstly the TaskDefinition itself is created. Afterwards every associated attribute is added to it. Explanations how the appending works in detail can be found in the `addGenericAttributeToTask` method.

`removeTask()`

Removes the specified taskDefinition. It also unlinks all associated AttributeDefinitions.

For performance reasons all recommendations and calculations for all use cases are done at the same time. This can be done by calling `refreshData()`. This method calls the REST API of the SocioCortex server. The result has the following structure shown in listing 7.3:

```
existing: List<TaskRecommendation>
recommended: List<TaskRecommendation> (without task)
```

Listing 7.3 Structure of resulting recommendation

The list existing contains all TaskRecommendations which are already defined as TaskDefinition. These elements are represented like (II) in figure 7.2. The recommended list contains all suggested attribute groups. These are represented as (IV).

Every TaskRecommendation has the subsequent structure:

```
taskDefinition: TaskDefintion (optional, if exist)
attributes: List<RecommendedAttribute>
otherAttributes: List<RecommendedAttribute>
```

Listing 7.4 Structure of TaskRecommendations

TaskRecommendations serve as container for the associated TaskDefinition, (attribute taskDefinition) and their attributes, whereat taskDefinition is optional. Every suggested TaskRecommendation (all from recommended of the result) does not have one. The otherAttributes contains all non-recommend attributes with their score. TaskDefinition itself contains amongst others its name and id. The RecommendedAttributes serve as container for each attribute and have the following structure (listing 7.5):

```
score: int
inTask: boolean
attributeDefinition: AttributeDefinition
```

Listing 7.5 Structure of RecommendedAttribute

The score is the in subsection 6.4.2 introduced mixture of the confidence with the adjustment. In the result it is normalized as integer within a range from 0 to 100. The boolean inTask indicates if the current attribute already belongs to the TaskDefinition. The attributeDefinition is the existing AttributeDefinition of the system. It contains a name, kind and id. If it is a free attribute, which means no AttributeDefinition exists for the given attribute, it will only contain its name.

8. Backend

In this chapter the technical implementation of the shown concepts will be illustrated as result of the done analysis. This implementation is an extension of the existing SocioCortex server. More information about the classification in the SocioCortex eco system can be found in section 7.1.

In the beginning the entire structure and the flow of the implemented algorithms are shortly explained in section 8.1. The two main classes, the `BasicRecommendationUtil` and the `FPMUtil`, are introduced in subsection 8.2 and 8.3. Lastly the finally used parameters as result of the analysis and evaluation are shown in section 8.4.

8.1. Structure

To get an overview about the implementation first the main classes are shortly explained with their features. Afterwards it is shown how they are basically used.

The implementation covers the following main classes:

`BasicRecommendationUtil`

The `BasicRecommendationUtil` is the basic class for the now existing and maybe later extended recommendations based on the history. It contains all steps for the data preparation introduced in section 6.3. This includes the selecting of all data from the database, the cleaning of the data and the construction of the transactions out of the `Changesets`.

`FPMUtil`

The `FPMUtil` uses the results of the `BasicRecommendationUtil` for the modeling steps of section 6.4. It contains the calculations for the support and the score of the groups. Additionally, it can work independently from the used data layer.

`TaskRecommendationHandler`

The handler is used to deal with the REST calls. There the methods of the `BasicRecommendationUtil` are called.

`TaskRecommendation`

This represents the view class for the recommended tasks. The structure has already been introduced in listing 7.4 of subsection 7.3.2.

`RecommendedAttribute`

This view class represents a single attribute. This includes the score and the associated `AttributeDefinition`, if there exists one. In addition to that this class has information about its type, if it is a recommendation or an existing attribute of the task. See listing 7.5 for more information.

RecommendedAttributeSerializer

The RecommendedAttributeSerializer serializes the RecommendedAttributes.

The following figure 8.1 shows how they are basically used:

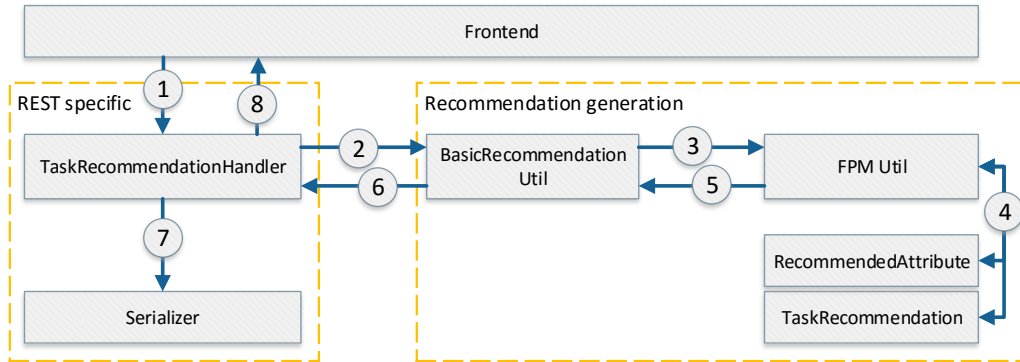


Figure 8.1.: Process in Backend

At the beginning of the process the front end calls the REST API for the type of the page (1). The call lead to the TaskRecommendationHandler. This handler initializes the BasicRecommendationUtil (2) with the parameters shown in subsection 8.4 and executes the method to get the task recommendations. During the initialization the BasicRecommendationUtil select, clean and construct the data. Afterwards it calls the FPMUtil to get the recommendations (3). This util creates out of a count map where every different combination of attributes is associated with their count, a support map with the in section 3.3.2 introduced *Apriori* algorithm. Afterwards the view models for TaskRecommendations and RecommendedAttributes can be instantiated with the calculated groups (4) and be return to the BasicRecommendationUtil (5). Here the view models are associated with their proper TaskDefinitions and AttributeDefinitions (6). The handler subsequently uses the serializer (7) to transform the view models into JSON format so it can be sent back to the front end (8).

8.2. BasicRecommendationUtil

The BasicRecommendationUtil represents the basis for the recommendations based on history. Therefore, it contains all steps of the data preparation phase of section 6.3. The util also contains standard values for the parameters of the preparation and modeling phase which are introduced in section 6.3 and 6.4. The util has two different constructors. One for using the standard parameters and one to set them to specific values. This one is especially essential for testing or evaluation purposes.

The first step is to select and clean the entire data. To do this the retrieveData() method calls the following sub functions. The subsections with the theoretical explanations can be found in the brackets.

getPagesForType() Retrieve all pages in descendant order of their publish date (Subsection 6.3.1.1).

getAttributeOccurrences() Identifies the number of occurrences of the free attributes (Subsection 6.3.1.2).

getSupportedAttributes() Determine supported attributes as combination of the existing AttributeDefinitions and the usually used free attributes. (Subsection 6.3.1.3).

filterPages() Filters the already retrieved pages by their completeness prevent the problem of incompleteness and to limit the time for retrieving the Changesets afterwards (Subsection 6.3.1.4).

getChangesetsForPages() Retrieve the changesets of the already filtered pages from the database. (Subsection 6.3.1.5).

Afterwards the transactions can be build out of this data with getTransactions(). The result is a map which associate each occurring combination of attributes in a transaction with their count. The associated algorithm how the grouping of the attributes and the calculation of the count works can be found in subsection 6.3.3.

The returned recommendations of the FPMUtil is independent from the existing TaskDefinitions. Therefore, the view models need to be associated with their proper TaskDefinitions and AttributeDefinitions. The following listing 8.1 shows how this association works:

Algorithm 8.1 Find groups

```

1: Init FPMUtil with standard parameters
2: //call recommendation with the existing groups
3: List<TaskRecommendation> taskRecommendations = fpmUtil.getGroups(existingGroups.values)

5: for each TaskRecommendation tr in taskRecommendations do
6:   for each RecommendedAttribute attr in tr.attributes do
7:     Find associated AttributeDefinition and link it to attr
8:   end for
9:   for each RecommendedAttribute attr in tr.otherAttributes do
10:    Find associated AttributeDefinition and link it to attr
11:   end for

```

```
12: end for
14: for each TaskDefinition taskDefinition in type.taskDefinitions do
15:   for each TaskRecommendation tr in taskRecommendations do
16:     if tr.attributes contain all taskDefinition.attributes and taskDefinition is not
17:       ↪ empty then
18:       Link taskDefinition with tr
19:       break
20:     end if
21:   end for
22:   if no TaskRecommendation is found then
23:     create new TaskRecommendation with no attributes and link taskDefinition to it
24:     add it to taskRecommendations
25:   end if
26: end for
```

This algorithm first iterates over all RecommendedAttributes of every TaskRecommendation and search for a proper AttributeDefinition. If one exists it can be linked to it. Secondly the TaskDefinitions are linked to their view model (TaskRecommendation). The second case (no TaskRecommendation is found) is for TaskDefinitions without attributes. There the FPMUtil can not return any attributes or recommendations. Therefore, an empty TaskRecommendation without attributes is created for the view to remain consistent and to show the counterpart of every TaskDefinition.

8.3. FPMUtil

The FPMUtil uses the prepared data of the BasicRecommendationUtil. The prepared and constructed data is independent of the beneath data layer.

Subsequently, the most important methods are shortly explained. Analogously to the explanation of the BasicRecommendationUtil the declaration of the beneath algorithms can be found in the subsections in brackets:

calculateSupport() This is the initial step of the FPMUtil. Out of the countMap, which associate every occurring combination of attributes in a transaction with their count, the supportMap is calculated and created with the *Apriori* principle. (Subsection 6.4.1)

getAllGroups() This is the main method, which is called from the BasicRecommendationUtil. In this function the generation of the recommendation is done like subsection 6.4.3 explains.

getSupportBuffered() Calculate the support of a transaction. If support has been calculated before it returns the existing value from the supportMap instead.

getConfidence() This method is overloaded. It returns the confidence of a single attribute or a set of attributes according to an existing set. The formula for the confidence can be found in equation 3.3.

getScore() The resulting score is a mixture of the confidence with a small adjustment by the order of the attributes of the page. This method is also overloaded. Therefore, the score can either be calculated for an attribute or a RecommendedAttribute against an origin set. (Subsection 6.4.2)

8.4. Used Parameters

As result of the analysis of the data in 6.2 the following standard values are set for the parameters:

Table 8.1.: Used parameters

Parameter	Range	Value
Data Preparation		
Max. number of Pages	$0 - \infty$	200
Min. percentage for supported properties	$0 - 1$	0.25
Min. completeness of page	$0 - 1$	0.7
Ext. min completeness	$0 - 1$	0.35
Min. number of pages after filter	$0 - \infty$	50
Max. number of pages after filter	$0 - \infty$	100
Max. number of Changesets	$0 - \infty$	3000
Max. time interval for Changeset grouping	$0 - \infty$	1800
Modeling		
Min. support	$0 - 1$	0.02
Min. score	$0 - 1$	0.3
Max. adjustment factor	$0 - 1$	0.2

The values for the maximal number of pages before and after filtering as well as the limit of Changesets have been chosen as result of the evaluation in chapter 9. The values for the minimal and extended minimal completeness of the pages as well as that 50% of the pages should be left after filtering can directly be taken from the results of the data analysis in subsection 6.2.2. The maximal time interval 30 minutes (1800 seconds) results from the outcome of subsection 6.2.4. There the perception has been made that after 30 minutes almost no new change happens within the same day. The values for the parameters of the modeling part arise from the evaluation.

9. Evaluation

The evaluation of the implemented algorithms is splitted into two basic groups. First the usefulness of the recommendation is analyzed in section 9.1. Afterwards the impact of the parameters in the proposed algorithms is tested in terms of scalability in section 9.2.

The testing data originate from the existing productive platform of the SEBIS chair of the TU Munich. This system uses a similar data layer to the in section 6 introduced one. A small part of the data was anonymized and ported to fit in the new database. The amount of available data can be extracted from the following table 9.1:

Table 9.1.: **Scope of available data**

Template	# Pages	# ChangeSets
Student Project	166	4365
Research Project	47	1286
Publication	119	1247
Article	90	1104
Team Member	21	988
Course	17	614

9.1. Usefulness of Results

The aim of this thesis is to help modelers to create task templates without prior experience or domain specific knowledge. Therefore, the user has an indicator, the so called score, which indicates how good specific attributes fit to an specific group.

The first result analysis is for types, which have no TaskDefinitions yet. As example the *Student Project* type is used. *Student Project* is a type, where e.g. bachelor theses can be processed. Here different user groups have different kinds of tasks. This evaluation belongs to the first use case (Create new task definition with attributes). Table 9.2 illustrates the recommended results.

Table 9.2.: **Recommendation without existing tasks.** The cross in "In task" indicates that no attribute has been associated to a task yet.

In task	Score	Attribute
Recommended Task		
×	0.4	Project
×	1.0	Advisor
×	1.0	Supervisor
Recommended Task		
×	1.0	Title (de)
×	1.0	Title (en)
×	0.8	Student
×	0.7	Start Date
Recommended Task		
×	0.5	Submission date
×	0.3	Thesis PDF

The recommendation offers three possible new tasks. Each of those tasks contain at least two attributes. The cross in "In task" for every attribute indicates that no attribute has been associated to a task yet. This confirms with the initial position of the type. Every attribute is specified by its name. In addition to that, they are ordered inside every recommendation by their existing user defined order in the type.

In details the results are reasonable. Every score is above 30%, which indicates that they fit well together.

Semantically they also fit together. The first TaskRecommendation (*Project, Advisor, Supervisor*) contains the typical attributes an advisor creates, before the student project starts. Out of this recommendation a new task "Setup assignment" can directly be created.

As soon the project is assigned to a student the titles (*en* and *de*) as well as the student itself and the start date of the project can be inserted. This fits to the second recommended task.

Lastly when the project has been completed the *Thesis PDF* can be uploaded and the *Submission Date* can be set to the current date. This completely fits to the last recommended task.

The next test concerns how the recommender interact with existing TaskDefinitions. As basis some tasks have been created and adapted. The results of the table 9.3 can be used for all four use cases.

Table 9.3.: **Recommendation based on existing tasks.** Associated attributes are marked with a checkmark, recommended ones with a cross. All attributes are colored according their kind in the recommendation. All black and red attributes already belong to a task. The red color indicates a low score. The green colored attributes are recommended attributes which should be added.

In Task	Score	Attribute
Setup assignment		
✓	0.3	Project
✓	0.6	Advisor
×	0.8	Supervisor
Enter organizational aspects		
✓	0.6	Status
✓	0.7	Student
✓	0.4	Submission date
×	0.6	Start Date
Enter title		
✓	1.0	Title (de)
✓	0.9	Title (en)
✓	0.1	Thesis PDF

The basic structure of table 9.3 is similar to table 9.2. Differences are the in the caption defined colors, which indicate the kind of recommendation for every attribute.

The first two TaskDefinitions contains recommendations for attributes, which should be added. This belongs to use case 3 (Add recommended attribute to existing task definition).

The first recommendation proposes to add a the attribute *Supervisor*. This fits to the already done perceptions of the first case in table 9.2. There the advisor changes this attributes before the project starts.

The next suggestion to add *Start Date* is reasonable, too. In a lot of cases at the start of a project both dates, the start and the submission date, are already known and therefore filled out. Because the process starts at this point of time *Status* is set on start and the student is assigned.

In the last TaskDefinition, *Enter title* is suggested to be removed (Use case 4). This make sense. The upload of the final thesis has to be done at the end of the project, whereas the entering of the titles happen at the start. Therefore, this attribute does not belong to the entire task.

9.2. Scalability

The aim of this section is to test the scalability of the proposed algorithms. Therefore, a test system is built up with different parameters to limit the number of ChangeSets or pages to see their impact. To get proper results a certain number of ChangeSets are needed. Consequently, this evaluation is focused on the type with the most available data (*Student Project*).

The other page types can not be used because they have too less ChangeSets to identify the scalability properly.

All tests are executed on a machine with a Core i5-4510u with 8gb ram. Each test case has been done 10 times. The results indicating the average of these values. Table 9.4 shows the experimental setup with the used parameters for each test case

Table 9.4.: **Used Parameters for scalability tests.** Every column represents a test case. Every row specifies the parameter for test case. The most important parametric values for every case are marked.

Case	1	2	3	4	5	6	7
	Without filter			With filter		Without filter	
Characteristic	75 pages	100 pages	150 pages	100 pages	150 pages	Limit Change-sets	Low Support
Max. number of Pages	50	100	150	100	150	150	150
Min. percentage for supported properties	0.25	0.25	0.25	0.25	0.25	0.25	0.25
Min. completeness of page	0.7	0.7	0.7	0.7	0.7	0.7	0.7
Ext. min completeness	0.35	0.35	0.35	0.35	0.35	0.35	0.35
Min. number of pages after filter	75	100	150	50	50	150	50
Max. number of pages after filter	75	100	150	100	100	150	100
Max. number of Changesets	∞	∞	∞	∞	∞	2000	∞
Max. time interval for Changeset grouping	1800	1800	1800	1800	1800	1800	1800
Min. support	0.02	0.02	0.02	0.02	0.02	0.02	0.002
Min. score	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Max. adjustment factor	0.2	0.2	0.2	0.2	0.2	0.2	0.2

Every test case has a minimal percentage of 0.25 for supported attributes. These have no impact on scalability. In addition to that the minimum and extended completeness of a page (to filter pages) and the time interval for the grouping of the ChangeSets remain the same. The time interval has been chosen according to the analysis of subsection 6.2.4. The first three test cases show the scalability for a fixed ascending numbers of pages without any limitation for the ChangeSets. Thus they use a fixed number of pages at start. Afterwards they got the same number as minimal and maximal number of pages after filtering. As a result, no filtering is done and all pages remain.

The aim for the next two cases (4, 5) is similar. They show the execution time for a different number of pages at the start. In contrast to the previous cases the page filter to ensure completeness are activated. As a consequence the ChangeSets are only retrieved for this 50 till 100 pages with a proper minimal completeness.

The sixth case, *Limit ChangeSets* uses the fixed number of pages (150) setup of the third case, but limits the ChangeSets.

The last one can directly be compared with the third one. It has the same setup for the preparing of the data. The difference is the lower minimal support rate of $\frac{1}{10}$.

The following figure 9.1 shows the resulting differences. Exact values can be found in table A.3.

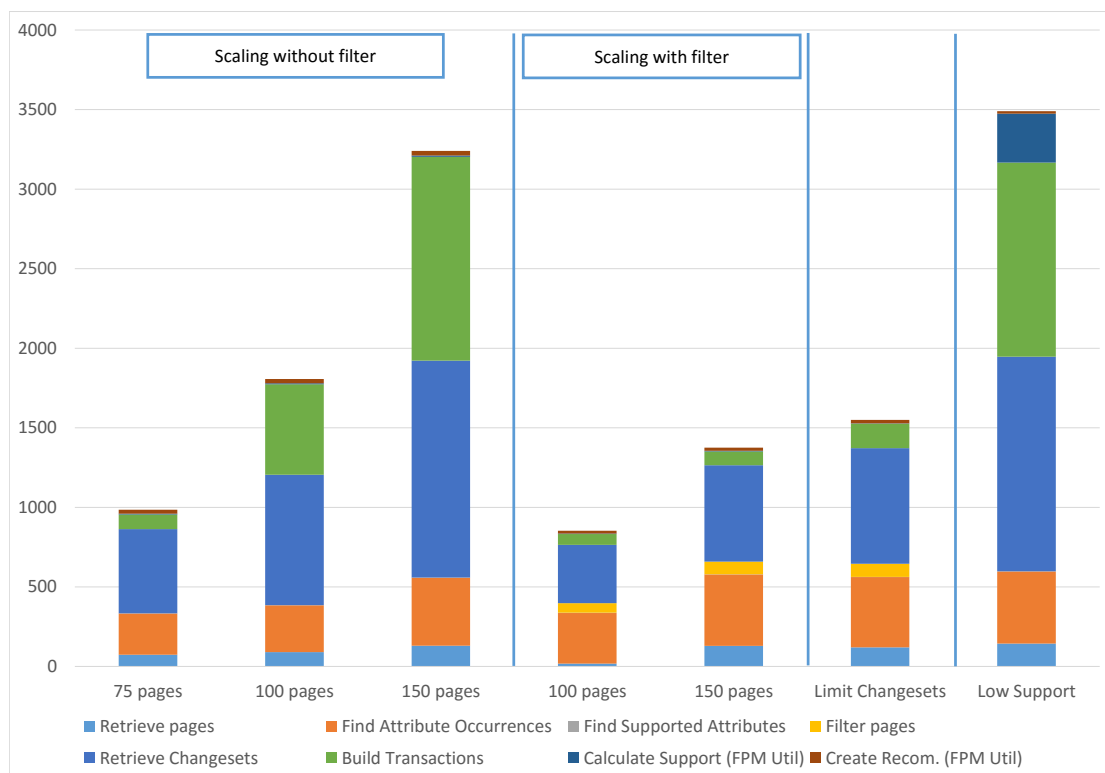


Figure 9.1.: Existing TaskDefinition with recommendations

On the one hand there are many similarities. All cases show that finding the supported attributes as well as the calculation of the support (except last case) and the creating of the recommendation has nearly no impact on the overall execution time. In general, the filtering algorithms uses almost no time in relation to the other parts. If no filtering is needed no time is used for it. The retrieving of the data to find the attribute occurrences, to get the ChangeSets and to create the transactions takes the greatest part in all cases. Reasons for this are that the finding of the attribute occurrences and the building of transactions also contains database accesses. The first one needs to find the associated attributes of each page. The second one needs to retrieve the groups of all users, which are included in the ChangeSets.

On the other hand the cases have some differences. The first three cases indicate, that the execution time directly depends on the number of pages. Starting with 75 pages (case 1) the overall execution time raises about 700 to 800 ms every additional 25. Reasons for this are the direct dependence for retrieving the ChangeSets. Case 1 (75 pages) results in only 1664 ChangeSets. Case 2 (100 pages) already contain 2226 and case 3 (150 pages) has 3376. This means every additional 25 pages starting from 75 results in about 560-575 more ChangeSets which takes in average 280 milliseconds.

The fourth and fifth case uses the same page number of pages, but allows filtering to get the best of them. From the 100 pages at start in case 4 there are 52 left, which have 1162 ChangeSets. Case 5 (150 cases) leads to 76 pages after filter and to 1516 ChangeSets. Consequently, the time for getting the ChangeSets and to build transactions out of them drop significantly.

Case 6 (Limitation of ChangeSets to 2000) for 150 pages leads to a similar time as case 5 with the same amount of pages. However, the number of ChangeSets is a lot higher.

The last case has the same setup as the third one. The only difference is the lower minimal support rate of $\frac{1}{10}$. In total this lower support rate leads to only 250 ms more execution time. This corresponds about $\frac{100 \cdot 250}{3240} \approx 8\%$. Nevertheless the time to calculate the support tree raises from about 5ms to 308ms. This is an increase of $\frac{100 \cdot 303}{5} = 6060\%$ even though the rate is only $\frac{1}{10}$ of the other one. The problem for finding a suitable minimal support is a well-known problem in literature.

9.3. Impact of Parameters

In the following tables 9.5 and 9.6 the recommended tasks according to the parameters in table 9.4 can be found.

Table 9.5.: Recommended tasks (Part 1)

Case 3			Case 5		
In task	Score	Attribute	In task	Score	Attribute
Existing			Existing		
✓	1.0	Title (de)	✓	1.0	Title (de)
✓	0.9	Title (en)	✓	0.9	Title (en)
✓	0.1	Thesis PDF	✓	0.1	Thesis PDF
Existing			Existing		
✓	0.5	Project	✓	0.3	Project
✓	0.8	Advisor	✓	0.6	Advisor
	0.7	Supervisor		0.8	Supervisor
Recommended			Recommended		
	0.8	Status		0.8	Status
	0.9	Student		1.0	Student
	0.6	Start Date		1.0	Start Date
				0.7	Submission date

Table 9.6.: Recommended tasks (Part 2)

Case 6			Case 7		
In task	Score	Attribute	In task	Score	Attribute
Existing			Existing Task		
✓	1.0	Title (de)	✓	1.0	Title (de)
✓	0.7	Title (en)	✓	0.9	Title (en)
✓	0.0	Thesis PDF	✓	0.1	Thesis PDF
Existing			Existing Task		
✓	0.5	Project		0.4	Status
✓	0.8	Advisor		0.3	Student
	0.7	Supervisor		0.4	Start Date
Recommended Task			Existing Task		
	0.8	Status		0.4	Submission date
	1.0	Student	✓	0.5	Project
	0.8	Start Date	✓	0.8	Advisor
				0.7	Supervisor
Recommended Task			Recommended Task		
				0.2	Final presentation slides
				0.5	Kick-off presentation slides

The initial setup were two existing TaskDefinitions. The first has the attributes *{Title (de), Title (en), Thesis PDF}*. The second one has the attributes *{Project, Advisor}*.

The resulting recommendations are very similar. All cases recommend *Supervisor* to the second existing *TaskDefinition*. Only the scores differ. Additionally, case 3, 5 and 6 recommend a new *TaskDefinition* with the attributes $\{Status, Student, Start Date\}$, which is similar to the result shown in table 9.3. Case 5 shows that the filtering of pages by their completeness helps to get the whole process. Therefore, the *Submission date* is added to the recommendation. This possible task illustrates the initial creation of the organizational aspects.

Case 7 differs from the other cases. The lower *minsup* allows the suggestion of possible attributes to the first task. This task represents all possible organizational aspects, which the advisor creates in the beginning of the thesis. Additionally, another recommendation can be found in case 7, which no other case offer. The suggestion contains the slides of the *Final presentation* and the *Kick-off presentation*. Here the problem occurs, that the given data is incomplete. A lot of the attributes, which have been edited and filled out on most of the pages have only a few *ChangeSets*. This leads to the fact that these attributes do not fulfill the minimal support and can therefore not be recommended. Setting the *minsup* down avoid this problem.

Taking these results into consideration case 5 has the best combination between the execution time and the usefulness of the resulting recommendations. It need only half of the time as case 3, which starts with the same amount of pages. In addition to that case 5 only uses mostly completed processes, which reflect the process most.

Part V.

Conclusion and Outlook

10. Conclusion and Outlook

This chapter summarizes the thesis in section 10.1 and gives a short outlook of possible future improvements to extend the recommendation system in 10.2.

10.1. Summary

The objective of this thesis was to create a concept how to help modelers of collaborative systems to create process templates for frequently performed tasks and to prototypically implement and evaluate these tools. This goal has been achieved vividly with the shown prototype. To accomplish this the whole process has been structured in a standardized way within the CRISP-DM model.

In the following the basic steps are briefly summarized:

First possible use cases have been defined, to find suitable applications for the recommendations. These cases describe possible interactions with their associated system reactions.

Afterwards, the underlying data has been analyzed. In the beginning, attributes has been identified, which are reasonable used. This leads to the perception that all predefined AttributeDefinitions as well as user defined free attributes are needed, because some of them have been evolved to be "best practice" and are used in almost all cases. Another perception was that not all free attributes should be taken into consideration, because many of them are only used in very few pages.

The next step concerned the problem of incompleteness. In order to find the best patterns, cases have to be completed. Alternatively, in newly created pages the problem can occur that only the first steps have been completed so far. This would lead to faulty results. Therefore, an analysis has been done to find the optimal percentage of completeness to keep as many instances (pages) as possible without losing too much significance.

Lastly there was an analysis of the histories themselves. For this purpose, the first step was to identify how they have been created and if the kind of creation has an impact on their expressiveness. The result was that most of the history entries (over $\frac{4}{5}$) only contain a single change. This leads to the conclusion that normal pattern mining algorithms, which only search the entire database may fail, because nearly no patterns exist, if only a single item is changed each time.

On the contrary another analysis of the history shows, that almost $\frac{2}{3}$ of all changes on the same page have been done within 30 minutes after the last change. This has led to the con-

clusion that these nearly concurrent changes need to be grouped to get appropriate results.

Having completed the analysis, concepts were developed to select, clean and construct usable information out of the data based on the previous perceptions.

The first step of this concept is to retrieve the data and to find supported attributes for the given type. The supported attributes contain all predefined attributes of the type in addition to the in common used free attributes.

In the next step, the retrieved data has been cleaned. Therefore, all Changesets that belong to pages, which are incomplete up to a certain degree were filtered out. Additionally, all unnecessary changes, such as text changes and changes of non-supported attributes were filtered out, too.

Apart from that a proper construction algorithm has been developed, which creates group related and weighted transactions based on the change history. Out of these transactions a method has been introduced to find suitable patterns as well as the score which indicates how well specific attributes fit together.

To show that this concept works in practice, it has been successfully implemented into the *SocioCortex* eco system. This includes both the implementing of the underlying algorithms to the *SocioCortex* server and to the *IME* client. The features of the client are based on the use cases. As a result, the client is able to satisfy them.

The implementation on the server has been splitted into a preparing and construction part for the data (*BasicRecommendationUtil*) and into a modeling part (*FPMUtil*), which creates the recommendations. These recommendations can therefore be done in a way, which allows the recommendation util to be independent from the underlying data structure. Additionally, the *BasicRecommendationUtil* can be used and extended for other recommendations based on history.

Finally, an evaluation for the usefulness of the recommendations and their scalability has been done. This evaluation results, that the recommendations are able to help even unexperienced users to create templates. Here the problem is, that the provided data was incomplete. Therefore, a lot of attributes, which have been edited and filled on almost all pages have only a few changes. This lead to the fact, that these attributes do not fulfill the minimal support and can not be recommended to a group. Additionally, only a few different types have currently been provided. For this reasons the usefulness can not be clearly stated for each case.

Nevertheless, the proposed approach has a limitation. Until now there is no availability to retrieve the order of the tasks or to create a whole process model.

10.2. Outlook

The proposed approach strongly focuses on the analysis of the data and finding proper patterns and not to recommend a whole process model. However, the implemented part currently provides an interface for other history based recommendation techniques to use the prepared data.

Possible next steps for the modeler part are to use these built patterns and prepared data to perform process mining. There a possible objective is to find a complete process model. These includes e.g. the order of the tasks. The proposed `BasicRecommendationUtil` already provides all required data to perform this task and can directly be extended. This can be done within the creation of the transactions. There the prepared data of all `ChangeSets` as well as the associated users and their groups are available.

Other possibilities are the mining of organizational aspects. These can include the relationships between users or the groups of users, who usually do similar activities.

Furthermore, recommendations to improve the end user experience would be advantageous. Assuming end users want to distribute tasks among themselves. Without recommendation techniques they have to analyze the previous cases, to find out who has done the task last time. With staff mining these users could automatically get recommendations about possible users or user groups. This can help end users to distribute tasks more effectively. Another possible recommendation can be the normal duration of tasks. This can help users to dispose the time, which is necessary for the whole process.

Appendix

A. Appendix

Table A.1.: Time differences between two subsequent ChangeSets

	Additional (absolute)	Sum (absolute)	Additional (in %)	Sum (in %)
<1 sec	2747	2747	31,56%	31,56%
<10 sec	778	3525	8,94%	40,49%
<30 sec	763	4288	8,77%	49,26%
<1 min	286	4574	3,29%	52,54%
<10 min	1022	5596	11,74%	64,28%
<30 min	173	5769	1,99%	66,27%
<1 h	66	5835	0,76%	67,03%
<3 h	105	5940	1,21%	68,24%
<24 h	130	6070	1,49%	69,73%
>24 h	2635	8705	30,27%	100,00%

Table A.2.: Completeness of pages

Completeness	# Pages	Pages in sum	Completeness	# Pages	Pages in sum
100%	100	28%	63%	4	57%
99%	0	41%	62%	2	58%
98%	0	41%	61%	0	58%
97%	0	41%	60%	28	58%
96%	0	41%	59%	0	62%
95%	0	41%	58%	27	62%
94%	0	41%	57%	43	65%
93%	3	41%	56%	23	71%
92%	0	41%	55%	1	73%
91%	0	41%	54%	3	74%
90%	3	41%	53%	20	74%
89%	0	42%	52%	28	76%
88%	8	42%	51%	0	80%
87%	0	43%	50%	31	80%
86%	4	43%	49%	0	84%
85%	0	43%	48%	8	84%
84%	0	43%	47%	0	85%
83%	3	43%	46%	21	85%
82%	0	44%	45%	6	88%
81%	0	44%	44%	11	88%
80%	3	44%	43%	0	90%
79%	0	44%	42%	0	90%
78%	0	44%	41%	10	90%
77%	17	44%	40%	33	91%
76%	0	46%	39%	0	95%
75%	4	46%	38%	0	95%
74%	0	47%	37%	2	95%
73%	4	47%	36%	19	95%
72%	0	47%	35%	0	98%
71%	22	47%	34%	0	98%
70%	0	50%	33%	0	98%
69%	0	50%	32%	0	98%
68%	0	50%	31%	0	98%
67%	0	50%	30%	14	98%
66%	57	50%	29%	0	100%
65%	0	57%	28%	0	100%
64%	1	57%

Table A.3.: Executing times of algorithm. All units are in milliseconds

Case	1	2	3	4	5	6	7
	Without filter			With filter		Without filter	
Characteristic	75 pages	100 pages	150 pages	100 pages	150 pages	Limited Change-sets	Low Support
Retrieve pages	74	90	131	18	129	120	144
Find Attribute Occurrences	259	294	427	318	449	442	453
Find Supported Attributes	1	1	1	2	1	1	1
Filter pages	0	0	0	60	80	83	0
Retrieve Change-sets	529	820	1363	366	606	726	1349
Build Transactions	92	569	1281	68	87	154	1220
Calculate Support (FPM Util)	5	6	8	3	4	2	308
Create Recom. (FPM Util)	25	27	29	18	19	22	15
Sum	985	1807	3240	853	1375	1550	3490

Bibliography

- [Aggarwal, 2015] Aggarwal, C. C. (2015). *Data mining: The textbook*. Springer, Cham.
- [Aggarwal and Han, 2014] Aggarwal, C. C. and Han, J., editors (2014). *Frequent pattern mining*. Springer, Cham.
- [Agrawal et al., 1993] Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216. ACM, Washington, D.C., USA.
- [Agrawal et al., 1996] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. (1996). Fast Discovery of Association Rules. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in knowledge discovery and data mining*, pages 307–328. American Association for Artificial Intelligence, Menlo Park, CA, USA.
- [Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules. *Proceedings of the 20th VLDB Conference Santiago, 1994*:487–499.
- [Cleve and Lämmel, 2014] Cleve, J. and Lämmel, U. (2014). *Data Mining*. Oldenbourg, München.
- [Cook and Wolf, 1998] Cook, J. E. and Wolf, A. L. (1998). Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249.
- [Dorn et al., 2010] Dorn, C., Burkhart, T., Werth, D., and Dustdar, S. (2010). Self-adjusting Recommendations for People-Driven Ad-Hoc Processes. In Hull, R., Mendling, J., and Tai, S., editors, *Business Process Management*, volume 6336 of *Lecture notes in computer science*, pages 327–342. Springer Berlin Heidelberg.
- [Eder and Pichler, 2005] Eder, J. and Pichler, H., editors (2005). *Probabilistic calculation of execution intervals for workflows: Temporal Representation and Reasoning, 2005. TIME 2005. 12th International Symposium on*.
- [Fayyad et al., 1996] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996). *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, Menlo Park, CA, USA.
- [Gantz and Reinsel, 2010] Gantz, J. and Reinsel, D. (2010). The Digital Universe Decade – Are You Ready?
- [Gantz and Reinsel, 2012] Gantz, J. and Reinsel, D. (2012). THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East.

- [Han et al., 2004] Han, J., Pei, J., Yin, Y., and Mao, R. (2004). Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach: Data Mining and Knowledge Discovery. *Data Mining and Knowledge Discovery*, 8(1):53–87.
- [Hand et al., 2001] Hand, D. J., Mannila, H., and Smyth, P. (2001). *Principles of data mining. Adaptive computation and machine learning*. MIT Press, Cambridge, Mass.
- [Herlocker et al., 2004] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53.
- [Hidber, 1999] Hidber, C. (1999). Online association rule mining. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 145–156. ACM, Philadelphia, Pennsylvania, USA.
- [Karthikeya and Ravikumar, 2014] Karthikeya, T. and Ravikumar, N. (2014). A Survey on Association Rule Mining. *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 3, Issue 1.
- [Klahold, 2009] Klahold, A. (2009). *Empfehlungssysteme: Recommender Systems ; Grundlagen, Konzepte und Lösungen*. Aus dem Programm IT-Management und -Anwendungen. Vieweg + Teubner, Wiesbaden, 1. aufl. edition.
- [Larose and Larose, 2015] Larose, D. T. and Larose, C. D. (2015). *Data mining and predictive analytics*. Wiley series on methods and applications in data mining. Wiley, Hoboken, NJ, 2. ed. edition.
- [Li et al., 2007] Li, J., Liu, D., and Yang, B. (2007). Process Mining: Extending a-Algorithm to Mine Duplicate Tasks in Process Logs. In Chang, K.-C., Wang, W., Chen, L., Ellis, C., Hsu, C.-H., Tsoi, A., and Wang, H., editors, *Advances in Web and Network Technologies, and Information Management*, volume 4537 of *Lecture notes in computer science*, pages 396–407. Springer Berlin Heidelberg.
- [Makanju et al., 2009] Makanju, A. A. O., Zincir-Heywood, A. N., and Milios, E. E. (2009). Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1255–1264. ACM, Paris, France.
- [Matthes et al., 2011] Matthes, F., Neubert, C., and Steinhoff, A. (2011). Hybrid Wikis: Empowering users to collaboratively structure information. *6th International Conference on Software and Data Technologies (ICSOFT)*, 2011.
- [Medeiros et al., 2003] Medeiros, A. d., van der Aalst, W.M.P., and Weijters, A. (2003). Workflow Mining: Current Status and Future Directions. In Meersman, R., Tari, Z., and Schmidt, D., editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture notes in computer science*, pages 389–406. Springer Berlin Heidelberg.
- [Motahari-Nezhad and Bartolini, 2011] Motahari-Nezhad, H. R. and Bartolini, C. (2011). Next Best Step and Expert Recommendation for Collaborative Processes in IT Service

- Management. In Rinderle-Ma, S., Toumani, F., and Wolf, K., editors, *Business Process Management*, volume 6896 of *Lecture notes in computer science*, pages 50–61. Springer Berlin Heidelberg.
- [Naisbitt, 1986] Naisbitt, J. (1986). *Megatrends: 10 Perspektiven, die unser Leben verändern werden*, volume 01/7235 of *Heyne-Sachbuch*. Heyne, München, 2. aufl. edition.
- [Park et al., 1995] Park, J. S., Chen, M.-S., and Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. *SIGMOD Rec*, 24(2):175–186.
- [Ricci et al., 2011] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2011). *Recommender Systems Handbook*. Springer US, Boston, MA.
- [Robillard and Dagenais, 2008] Robillard, M. P. and Dagenais, B. (2008). Retrieving Task-Related Clusters from Change History. In *15th Working Conference on Reverse Engineering (WCRE)*, pages 17–26.
- [Runkler, 2010] Runkler, T. A. (2010). *Data Mining: Methoden und Algorithmen intelligenter Datenanalyse*. Computational intelligence. Vieweg + Teubner, Wiesbaden.
- [Schalkamp, 2015] Schalkamp, J. (2015). *An Integrated Modeling Environment for Social Applications*. Master’s Thesis, Technische Universität München, München.
- [Schonenberg et al., 2008] Schonenberg, H., Weber, B., van Dongen, B., and van der Aalst, Wil (2008). Supporting Flexible Processes through Recommendations Based on History. In Dumas, M., Reichert, M., and Shan, M.-C., editors, *Business Process Management*, volume 5240 of *Lecture notes in computer science*, pages 51–66. Springer Berlin Heidelberg.
- [Schönig et al., 2015a] Schönig, S., Cabanillas, C., Jablonski, S., and Mendling, J. (2015a). Mining the Organisational Perspective in Agile Business Processes. In Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., and Ma, Q., editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 214 of *Lecture Notes in Business Information Processing*, pages 37–52. Springer International Publishing, Cham.
- [Schönig et al., 2015b] Schönig, S., Gillitzer, F., Zeising, M., and Jablonski, S. (2015b). Supporting Rule-Based Process Mining by User-Guided Discovery of Resource-Aware Frequent Patterns. In Toumani, F., Pernici, B., Grigori, D., Benslimane, D., Mendling, J., Ben Hadj-Alouane, N., Blake, B., Perrin, O., Saleh Moustafa, I., and Bhiri, S., editors, *Service-Oriented Computing - ICSOC 2014 Workshops*, volume 8954 of *Lecture notes in computer science*, pages 108–119. Springer International Publishing.
- [Schönig et al., 2012] Schönig, S., Zeising, M., and Jablonski, S. (2012). Adapting Association Rule Mining to Discover Patterns of Collaboration in Process Logs. In Pu, C., Joshi, J., and Nepal, S., editors, *8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 531–534.
- [SEBIS chair, 2015] SEBIS chair (2015). SocioCortex - A Social Information Hub. <https://wwwmatthes.in.tum.de/pages/13uzffgw1h8z4/SocioCortex>. Last accessed on Dec 01, 2015.

- [Shearer, 2000] Shearer, C. (2000). The CRISP-DM Model: The New Blueprint for Data Mining. *Journal of Data Warehousing*, 2000(5 (4)):13–22.
- [Srikant and Agrawal, 1995] Srikant, R. and Agrawal, R. (1995). Mining Generalized Association Rules. *Proceedings of the 21st VLDB Conference*, 1995:407–419.
- [Srivastava et al., 2000] Srivastava, J., Cooley, R., Deshpande, M., and Tan, P.-N. (2000). Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. *SIGKDD Explorations*, Vol. 1, Issue 2:12–23.
- [Toedt, 2014] Toedt, M. (2014). *Data revolution: How Big Data will change the way of doing business*. epubli, Berlin, 1st ed. edition.
- [van der Aalst et al., 2004] van der Aalst, W., Weijters, T., and Maruster, L. (2004). Workflow mining: discovering process models from event logs: Knowledge and Data Engineering, IEEE Transactions on. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1128–1142.
- [van der Aalst, Wil M. P., 2011] van der Aalst, Wil M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg.
- [van der Aalst, W.M.P. et al., 2003a] van der Aalst, W.M.P., Stoffele, M., and Wamelink, J. (2003a). Case handling in construction. *Automation in Construction*, 12(3):303–320.
- [van der Aalst, W.M.P. et al., 2003b] van der Aalst, W.M.P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. (2003b). Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267.
- [Wen et al., 2009] Wen, L., Wang, J., van der Aalst, W., Huang, B., and Sun, J. (2009). A novel approach for process mining based on event types: Journal of Intelligent Information Systems. *J Intell Inf Syst*, 32(2):163–190.