JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Klaus Peter Brandner, BSc

Submitted at
Institute for Business Informatics Software Engineering

Supervisor
Ass.Prof. Dipl.-Ing. Dr. Rainer Weinreich

September 2018

A Recommender System for Software Architecture Decision Making

Master Thesis
to obtain the academic degree of
Master of Science
in the Master's Program
Business Informatics

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.
This thesis is identical to the electronically transmitted text document.

.....	.....
date	signature

Abstract

The quality and success of a software product highly depends on its software architecture. Inappropriate decisions during the architectural design of a software system are often hard to reverse and might lead to costly and time-intensive changes later on. Therefore, software architects are required to make proper design decisions early on in the architectural design process. The goal of this thesis is the development of a recommender system for software architecture design decisions. The thesis starts by presenting basic concepts and terms of software architecture, software architecture decision making, and decision models. It then presents fundamentals of recommender systems including different kinds of recommender systems. The main part of the thesis is the presentation of the developed recommender system for software architecture decision making based on decision models. This includes a presentation of the main requirements of the system, of its conceptual realization, and of its implementation.

Kurzfassung

Die Qualität und der Erfolg eines Softwareprodukts hängen stark von seiner Softwarearchitektur ab. Falsche Entscheidungen bei der Gestaltung der Architektur eines Softwaresystems sind oft schwer rückgängig zu machen und können später zu kostspieligen und zeitintensiven Änderungen führen. Daher sind Softwarearchitekten gefordert, frühzeitig im Architekturdesign richtige Entwurfsentscheidungen zu treffen. Das Ziel dieser Arbeit ist die Entwicklung eines Empfehlungssystems für Entwurfsentscheidungen zur Gestaltung der Softwarearchitektur. Die Arbeit beginnt mit der Erklärung grundlegender Begriffe und Konzepte in den Bereichen Softwarearchitektur, Entwurfsentscheidungen für Softwarearchitekturen und Entscheidungsmodelle. Anschließend werden grundlegende Aspekte von Empfehlungssystemen vorgestellt, einschließlich verschiedener Arten von Empfehlungssystemen. Der Hauptteil der Arbeit ist die Präsentation des entwickelten Empfehlungssystems zur Unterstützung der Entscheidungsfindung im Architekturdesignprozess auf Basis von Entscheidungsmodellen. Dazu gehört eine Darstellung der wesentlichen Anforderungen an das System, seiner konzeptionellen Umsetzung und seiner Implementierung.

Contents

Chapter 1

Introduction

Software architecture is a major research topic in the field of software engineering. The interest in software architecture increased significantly in the last two decades [1]. The architecture of a software system mainly refers to the set of design decisions that were made in the development and later adaptations of the system. A proper software architecture is essential for high-quality and successful software products [2]. Taylor et al. [2] highlight the relation between the quality of the architectural design and the quality of software products. They state that high-quality software products with a poor design are to a high degree uncommon. Babar et al. [3] state that in the design process of a software architecture, the software system as a whole gets decomposed to deal with the complexity of a system and to make it manageable. The decomposed parts of the system are responsible for solving a user's problem. However, there does not exist a recipe for this method which makes the design process of a software architecture a creative one. Therefore, the quality of the architectural design process highly depends on the people involved in this process.

Making the right decisions in the architectural design process of a software system early on is crucial to avoid costly and time-intensive changes later. Babar et al. [3] even state that changing a design decision at a later stage can not only be costly but may also be impossible to do. To overcome this issue, this thesis examines the use of a recommender system in the decision-making process of software architecture design. The idea is to make recommendations to software architects so that they make decisions that fit best to the requirements for the software system. It is important to highlight that the accuracy of the recommender system depends on the documented architectural knowledge like requirements and already made design decisions. However, the recommender system should be designed in a way that it provides recommendations even if there is little known and documented about a software system. Thereby, the recommender system should provide recommendations early on in the architectural design process and becomes more accurate when more architectural knowledge is documented.

1.1 Goal of the Thesis

Software architecture is all about making design decisions that fulfill the requirements of the customer and other stakeholders. Making the right decisions, however, can be a challenging task and depends on the qualification and experience of the software architect [3]. This thesis demonstrates how a recommender system can support software architects in the decision-making process. The goal of the thesis is the development of a recommender system that uses architectural knowledge like requirements and documented design decisions to make recommendations for upcoming design decisions. Furthermore, the recommender system should incorporate the documented architectural knowledge of other software systems. Using this information, software architects will get recommendations based on how other software architects decided in certain design areas. The goal of the thesis can be reached by answering two research questions. First, it needs to be clarified what types of recommender systems exist and how they work. This is done by conducting a literature analysis on recommender systems. Secondly, this thesis demonstrates how these recommender systems can be applied in the context of software architecture. To answer this research question, a recommender system was developed for a tool for managing architectural knowledge that has been developed by the software architecture group (lead R. Weinreich) at the Department of Business Informatics - Software Engineering at the Johannes Kepler University Linz.

1.2 Structure of the Thesis

This thesis is structured into four chapters. Chapter 2 deals with terms and concepts that are necessary for the understanding of the following chapters. Next, an overview of prominent recommender systems is presented in Chapter 3. This chapter analyzes different kinds of recommender systems and explains how different recommender methods work. Chapter 4 deals with the requirements of the recommender system and presents a concept for recommendation strategies to fulfill these requirements in the context of software architecture. Thereby, this chapter focuses on the data that is used to make recommendations, the recommender methods that are applied, how the different recommender methods can be combined, and the results of the recommender system. Chapter 5 demonstrates how the results of the recommender system are displayed in the user interface of the application. Thereby, the chapter shows the information that is presented to the user supporting the decision-making process. Also, this chapter deals with architecture and implementation aspects. It shows the top-level structure of the developed recommender system and explains key-components and the interactions between them.

Chapter 2

Terms and Concepts

This chapter explains terms and concepts that are relevant for the understanding of the following chapters. The first section deals with the term software architecture. Therefore, this section clarifies the term software architecture by comparing and discussing various definitions of that term. Furthermore, this section explains what architectural knowledge is and why it is important to document it. The second section introduces architecture profiles as a means to document architectural knowledge. Finally, the last section deals with decision models. This section shows how decision guidance models are used to support the decision-making process of software architectures. Additionally, it presents a possibility to document the decisions in terms of a decision documentation using decision models.

2.1 Software Architecture

In the early 1990, the first definitions of the term software architecture emerged. Ever since, the definitions evolved as the view on software architecture changed. In 1995, Shaw et al. [4] defined the term software architecture as follows:

> "The architecture of a software system defines that system in terms of computational components and interactions among those components." [4]

This definition shows, that the components of a software system and the interactions between these components refer to its architecture. However, this is a quite narrow view on software architecture as more aspects play an important role in the design of a software architecture nowadays. The second definition was made by Bass et al. [5] in 2003:

> "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [5]

In contrast to the first definition by Shaw et al. [4], this definition shows that a software system may have multiple structures that are of interest. Therefore, the whole software architecture can not always be visualized by only one point of view on the system. Furthermore, Bass et al. [5] state that a software architecture also contains the externally visible properties of software components. However, the result of the software architecture design process is still solution-oriented. Hence, the software architecture shows how the system looks like. A more recent definition was made by Taylor et al. [2] in 2010.

> "A system's architecture is the set of principal design decisions made during its development and any subsequent evolution." [2]

In this definition, the view on software architecture changed from a solution-oriented view to a decision-oriented view. Therefore, it became more important to see why a certain software system is the way it is in contrast to how the system looks like [1]. For this purpose, the software architecture refers to all design decisions made in the lifecycle of a software system. Also, Bosch [6] had a similar view on software architecture in 2004.

> "The key difference from traditional approaches is that we do not view a software architecture as a set of components and connectors, but rather as the composition of a set of architectural design decisions." [6]

The definitions by Taylor et al. [2] and Bosch [6] show that the made design decisions and the rationale behind these decisions are often more interesting than the actual solution. Therefore, it is important to document both the design decisions and the rationales during the architectural design process.

Generally speaking, architectural knowledge is produced during the architectural design process. According to Babar et al. [3], important parts of the architectural knowledge are the solutions as well as the design decisions that lead to these solutions. However, also assumptions, the context, and other factors that lead to the system as it is are parts of the architectural knowledge. In this regards, Weinreich and Groher [1] include the requirements, project-generic knowledge like the experience of the people involved, patterns, and architectural tactics in the architectural knowledge. As mentioned earlier, the rationales behind the design decisions are also an important part of the architectural knowledge. Babar et al. [3] state that these rationales serve two purposes. First, they show the reason for a certain design choice. Second, they show the alternatives that have been identified and evaluated.

A problem that is present oftentimes in practice is that design decisions are not well documented or not documented at all. In most cases, the solution may be documented, but not the rationale behind these solutions [3]. However, as mentioned before, the design decisions are often more interesting than the actual solutions. This can lead to the situation in which this architectural knowledge resides only in the head of the software architect. If the software architect leaves the company or the architectural knowledge is not accessible due to other reasons, this knowledge can get lost [3]. This has a significant impact on the subsequent evolution of the software system as design decisions made in the past are no longer comprehensible. Babar et al. [3] examined the reason behind the lack of documented architectural knowledge and found three main reasons. First, a design decision could be obvious to the software architect or the software architect does not see the design decision as such. For example, based on the experience, a software architect always follows a certain pattern and sees no need for documentation in this case. Second, the software architect is aware of a design decision but leaves it undocumented. Weinreich and Groher [1] identified that time and cost are the main factors for this behavior. Finally, the software architect does not document a design decision intentionally. Tactical company reasons or personal reasons of the software architect are main drivers for this case. For example, the software architect could keep design decisions undocumented to protect his position in the company or organization.

Since a software system will evolve over time, it might be necessary to replace certain design decisions. Therefore, it is important to keep the architectural knowledge up-to-date to make it manageable and maintainable [3]. The need for the management of architectural knowledge led to the development of architecture knowledge management approaches. Many different tools have been developed for this purpose. Examples can be found in [7].

To sum up the term software architecture, nowadays, the term mainly refers to the set of design decisions made in the architectural design process of a software system. According to Babar et al. [3], the software architecture serves three purposes. On the one hand, it is used for communication purposes among stakeholders. On the other hand, the software architecture captures early design decisions and, therefore, the global structure of the system. Finally, the software architecture serves as an abstraction of the whole software system to make it readable by people and to make certain concepts reusable.

2.2 Architecture Profile

As explained in the previous section, the documentation of the architectural knowledge is a crucial task to avoid vaporization of relevant knowledge over time. However, a complete documentation of a software system can become long winded and, as a consequence, is never read [8]. Architecture haikus have been proposed to deal with this issue. Keeling [8] defines architecture haiku as follows:

> "An architecture haiku is a quick-to-build, uber-terse design description that lets you distill a software-intensive system's architecture to a single piece of paper." [8]

Therefore, architecture haikus restrict the software architecture documentation to the absolute necessary to include only the most essential design decisions and rationales. Software architects are forced to aim attention at the most important aspects of a software system and leave out superfluous details. The aim is to have a software architecture documentation that fits on one piece of paper. Keeling [8] highlights an example to shorten the documentation of a software architecture by saying that it can be assumed that the reader is in possession of certain knowledge which makes it unnecessary to document decisions in full detail. For example, it can be assumed that the meaning of the word "layer" in software architectures will be known by the reader and does not need to be explained in more detail. However, keeping the software architecture documentation short and compact can be challenging since oftentimes it is easier to write a long documentation instead of a short documentation focusing only on the most relevant aspects. This can be compared with the prominent saying: "I write you a long letter because I did not have time to write you a short one.". Nevertheless, Fairbanks [9] recommended that the architecture haiku should include following information:

- a solution description
- trade-offs
- quality attribute priorities
- architectural drivers
- design rationales
- constraints
- architecture styles
- diagrams

Based on the architecture haiku approach, a lean web-based architecture knowledge management tool has been developed by the software architecture group (lead R. Weinreich) at the Department of Business Informatics - Software Engineering at the Johannes Kepler University Linz. The tool is called Architecture Knowledge Base (AKB) and extends the basic concept of the architectural haiku. Using this tool, stakeholders can create, edit, and share architecture descriptions in an architecture haiku manner for their projects. These architecture descriptions are called architecture profiles. An example of an architecture profile describing the architecture of eBay is shown in Figure 2.1.

Figure 2.1 shows that architecture profiles are divided into two parts. The left part shows attributes, constraints and trade-offs and the right part shows design decisions which are divided into general decisions, styles, patterns, and tactics. Also, a rationale can be

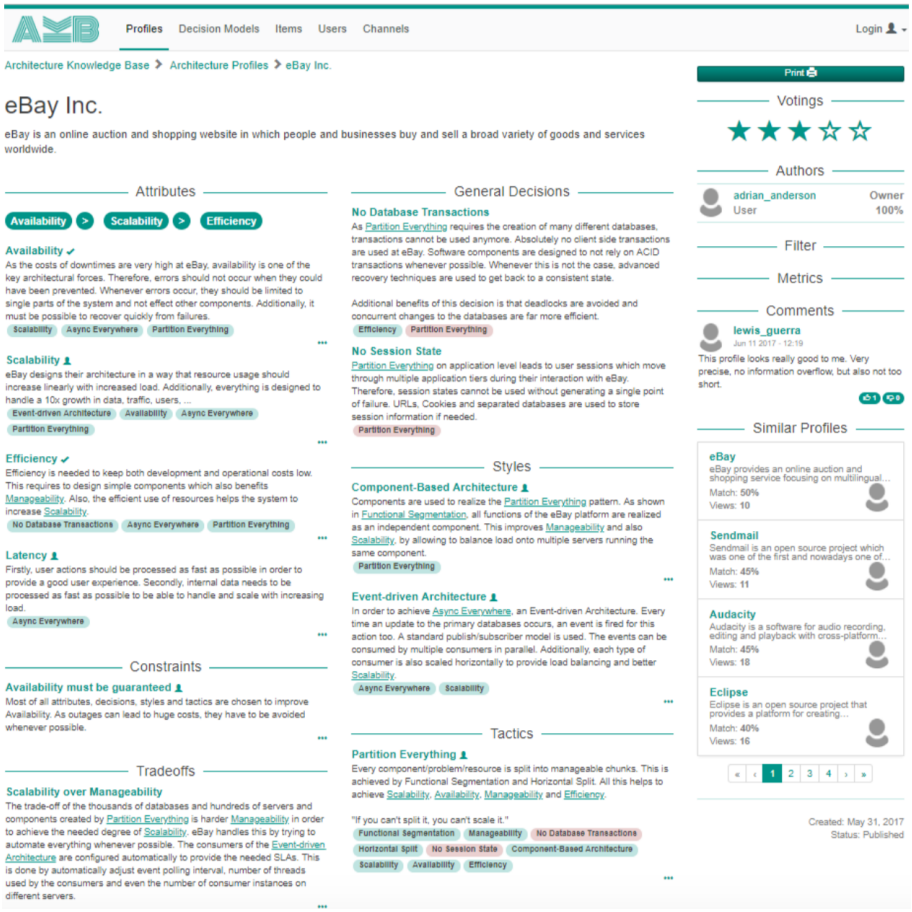


Figure 2.1: Excerpt of an architecture profile for eBay

provided to each design decision. It can be said that the left part shows attributes, constraints and trade-offs that are influenced by or result from the design decisions on the right side of the architecture profile. Furthermore, green, red, and gray boxes indicate relations between elements. The color refers to the kind of impact which can be either positive, negative, or neutral. As Figure 2.1 shows, the general decision "No Database Transactions" has a positive impact on the attribute "Efficiency" in the case of eBay.

As Keeling [8] explains, for large and complex software systems, it is possible to create multiple architecture haikus describing subcomponents of the whole system. In this case, he mentions that it might also be helpful to have a top-level architecture haiku of the whole system to have a common understanding among stakeholders. The AKB tool supports this idea by offering the ability to hierarchically organize architecture profiles. The way in which architecture profiles are hierarchically organized is left to the user.

Furthermore, users can add properties to architecture profiles in form of tags. These properties can be used to show the application context of the architecture profile, for example. In case of a microservice architecture, one would add the tag "Microservice"

to the architecture profile. Properties are especially useful when multiple architecture profiles are used to describe a software architecture. In this case, properties may determine which design decisions are documented in this architecture profile. For instance, a sub-architecture profile describing all design decisions for monitoring will have the tag "Monitoring".

2.3 Decision Models

In contrast to architecture profiles which represent project-specific architecture knowledge, decision models provide project-generic architecture knowledge. Decision models show design options in a specific design space and can be reused for new architecture profiles. In this context, a design space refers to a specific area of a software architecture in which decisions can be made, e.g., Monitoring, Caching, Versioning. In other domains, decision models have been successfully used for decision making and the documentation of decisions. Furthermore, they have been used for design space exploration. This section examines the use of decision models in software architecture and presents a meta-model developed by Haselböck et al. [10] to create decision models for microservice architectures. These decision models have been incorporated into the Architecture Knowledge Base (AKB) tool presented in the previous section. Thus, they can be used to create architecture profiles.

One of the first approaches for decision models for human-computer interaction was developed by MacLean et al. [11, 12]. They created the semiformal notation QOC. QOC stands for questions, options, and criteria. The aim of the QOC approach is the analysis and representation of a design space. Thereby, the analysis of a design space is conducted by defining questions which refer to main design issues, possible options to answer these questions, and criteria to make the options assessable and comparable. Nowadays, the QOC approach by MacLean et al. [11, 12] serves as a basis for decision models in software architecture [10].

Zimmermann and Miksovic [13] introduce decision models in the context of service-oriented architecture. They focused on creating models that should serve as guidance when making decisions by showing possible issues and solutions. These guidance models can be reused for design decisions in multiple projects since they provide project-generic knowledge. Therefore, Zimmermann and Miksovic describe guidance models as "reusable assets" [13]. Similarly, Lewis et al. [14] presented decision models in the context of cyber-foraging systems. They also aimed to create guidance for the architectural design process and the subsequent evolution of these systems. The decision models by Lewis et al. [14] can be compared to the guidance models by Zimmermann and Miksovic [13]. Both concepts include the distinction between problem-space and solution-space modeling. The problem-space comprises the requirements for the system and the solution-space the possible design options.

Haselböck et al. [10, 15, 16] examined decision models for microservice architectures. They developed a meta-model as a basis to create decision models. The meta-model can be compared to those of Zimmermann and Miksovic [13] and Lewis et al. [14]. As mentioned before, the decision models were incorporated into the Architecture Knowledge Base (AKB) tool that has been developed by the software architecture group (lead R. Weinreich) at the Department of Business Informatics - Software Engineering at the Johannes Kepler University Linz. Similar to architecture profiles presented in the previous section, decision models can be created, edited, and shared. Decision models can be used in the architectural design process as guidance and for documentation in multiple architecture profiles. Hence, they are project-generic assets that can be used by stakeholders when making decisions for a software architecture.

2.3.1 Decision Guidance

This section focuses on the use of decision models for decision guidance. For this purpose, decision models include concerns, aspects, design options, components, technology options, and any relationships between these elements. Concerns refer to questions in the QOC approach by MacLean et al. [12] and to requirements in other decision models, e.g., the decision models by Lewis et al. [14]. Therefore, concerns represent issues that have been identified in a specific design space and that are addressed by possible design options. A design option refers to an option in the QOC approach and can be either a general decision, an architectural style, a pattern, or a tactic. Design options can be extended by showing required components, on the one hand, and technology options that can be used to implement them, on the other hand. Finally, aspects refer to the criteria in the QOC approach. Design options can have positive, negative, or both positive and negative influences on aspects. This information is used to make design options assessable and comparable. An example of a decision model for service discovery in microservice architectures is shown in Figure 2.2.

Here, the decision model is presented as a graph and shows the two design options server-side discovery and client-side discovery. The concerns that have been identified are displayed on the left side and show which design option addresses these concerns. At each design option, there is a list of aspects and the influence the design option has on this aspect. Aspects that are positively influenced have a $+$ symbol in front of it, aspects that are negatively influenced a $-$ symbol, and aspects that are both positively and negatively influenced by the design option have a $+-$ symbol in front of it. For example, server-side discovery has a positive influence on reusability, but a negative influence on performance. Client-side discovery, on the other hand, has a negative influence on reusability, but a positive influence on performance. In a simple assumption, based on this information, one would select the design option server-side discovery if reusability is more important than performance. Additionally, components that are required by the design options and technology options that implement these components are shown on the right side. Components and technology options may have an impact on the decision making since

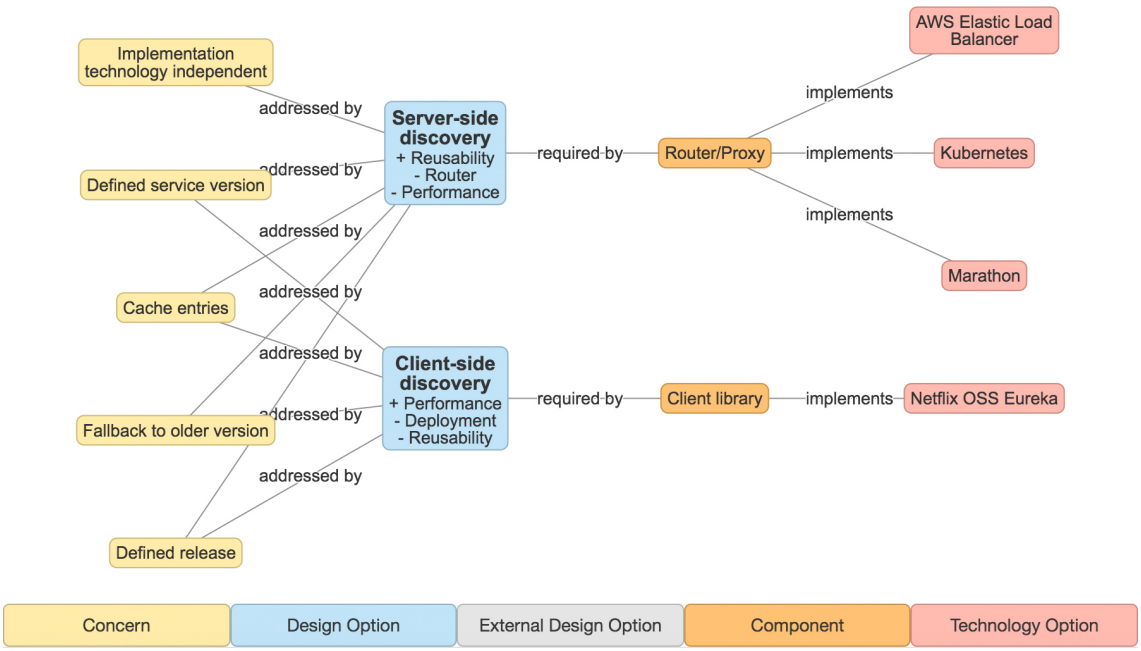


Figure 2.2: Decision Guidance Model for Service Discovery

decisions could be made based on the components and technologies already in use in the software system. Decision models also provide the possibility to display dependencies to other decision models. For this purpose, it can be defined if a design option of another decision model excludes or requires a specific design option in this decision model. The design options of other decision models refer to external design options and are also visualized in the graph.

As mentioned before, decision models can be created and edited in the AKB tool. Besides the representation of a decision model as a graph, decision models are represented in a textual form similar to architecture profiles. An example of the decision model for service discovery is shown in Figure 2.3.

Just like in architecture profiles, decision models are also separated into two parts. The left part containing concerns and aspects represents the problem-space, and the right part containing design options, components, and technology options represents the solution-space. Also, like in architecture profiles, relationships between elements are shown as boxes and the color indicating the impact on the element. Decision models can also be hierarchically organized to provide a better understanding of the design space. In addition, properties of a decision model can be added to the model in the form of tags. These properties are used to classify decision models. For example, properties can refer to the application context of decision models. Thereby, a decision model that can be used to make a decision in the context of monitoring may get the tag "Monitoring".

In a decision model hierarchy, it might be recommended to use decision models in a

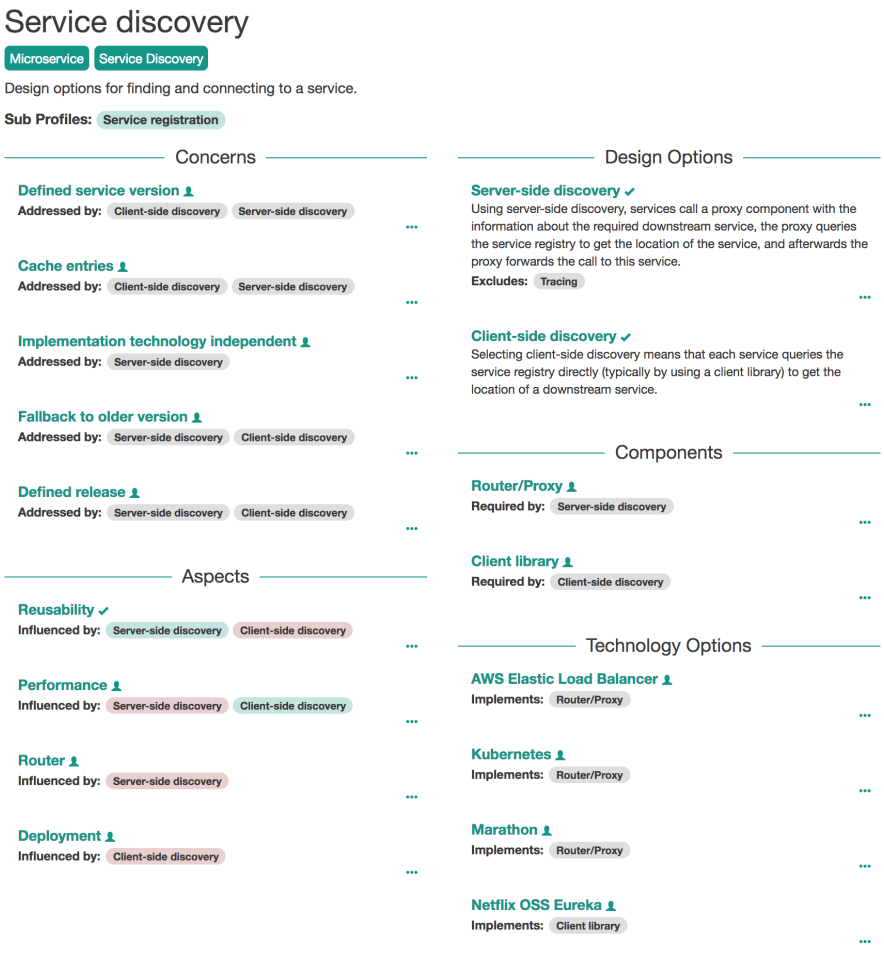


Figure 2.3: Decision Guidance Model for Service Discovery

specific order to make design decisions. For example, for microservice API management, it will make sense to make design decisions in API specification before API support. Design processes have been introduced to show the order in which decision models in a hierarchy should be used. An example of a design process for microservice API management is shown in Figure 2.4.

The design process shows the start- and end-node of the process as well as decision models of the hierarchy which are also displayed as nodes. The design process can be modeled by displaying decision models in parallel and sequentially. The purpose of the design process can be summarized as a guidance for the sequence in which decision models should be used in a hierarchy.

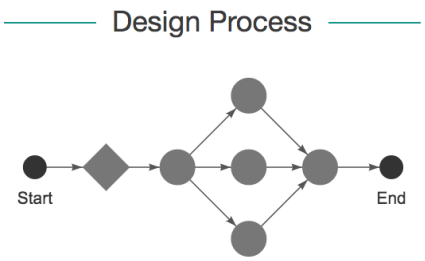


Figure 2.4: Design Process for Microservice API Management

2.3.2 Decision-Making and Documentation

As mentioned before, decision models are successfully used for documentation in other domains. This section presents the decision-making and documentation process using the decision models by Haselböck et al. [10] introduced in the previous section. It is important to highlight that decisions and the documentation of these decisions are always for a specific architecture profile. Therefore, the corresponding architecture profile needs to be selected before documentation.

Decision-Making

The decision-making process mainly consists of evaluating and comparing different options and selecting the most promising options. Decision models in the AKB tool show which concerns are addressed by a design option and what impact a design option has on certain aspects, e.g., performance, scalability. Using this information, stakeholders can evaluate and compare the different design options in the decision model. To support this process even further, the tool provides functionality to preselect design options to highlight the elements affected by the design option. An example is shown in Figure 2.5.

The figure shows that the design option server-side discovery was selected. Each element that has no relationship to the design option is greyed out, like the aspect deployment and the component client library. Alternatively, any other element can be selected to show their impact on elements. An example is shown in Figure 2.6 where the concern "Implementation technology independent" was selected.

Here, the decision model highlights which design options address the selected concern. In this case, the concern is addressed by server-side discovery. Instead of concerns, also aspects, components, and technology options can be selected to see which design options affect these elements. This functionality should help to make design options better comparable and consequently support the decision making.

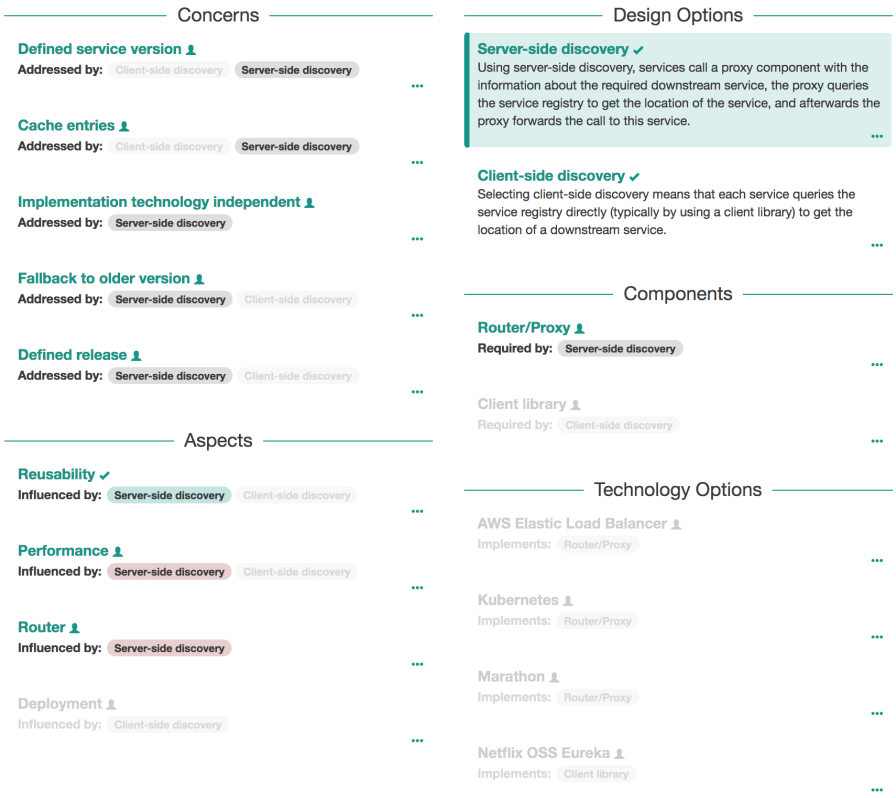


Figure 2.5: Preselection of a Design Option

Decision Documentation

After the user experimented with the decision model by preselecting and comparing design options and other elements, the tool provides functionality for documentation once a decision was made. First, the user selects the architecture profile for which the documentation should be created. During the documentation, the user selects the chosen design options if they were not already selected in the experimentation phase. As discussed in Section 2.1, the rationales behind the decisions play an important role in the documentation. Therefore, the user can provide a rationale for each design option to describe why it was chosen or not chosen. In the tool, this is done by providing a textbox at each design option as shown in Figure 2.7a.

Besides, concerns and aspects can be rated during the documentation process to show their importance for the decision. Therefore, a rating scale from one to five is displayed at each aspect and concern as shown in Figure 2.7b.

The figure shows that reusability has been rated with four stars and performance only with two stars. In order to justify the rating, the user can provide a rationale at each element.

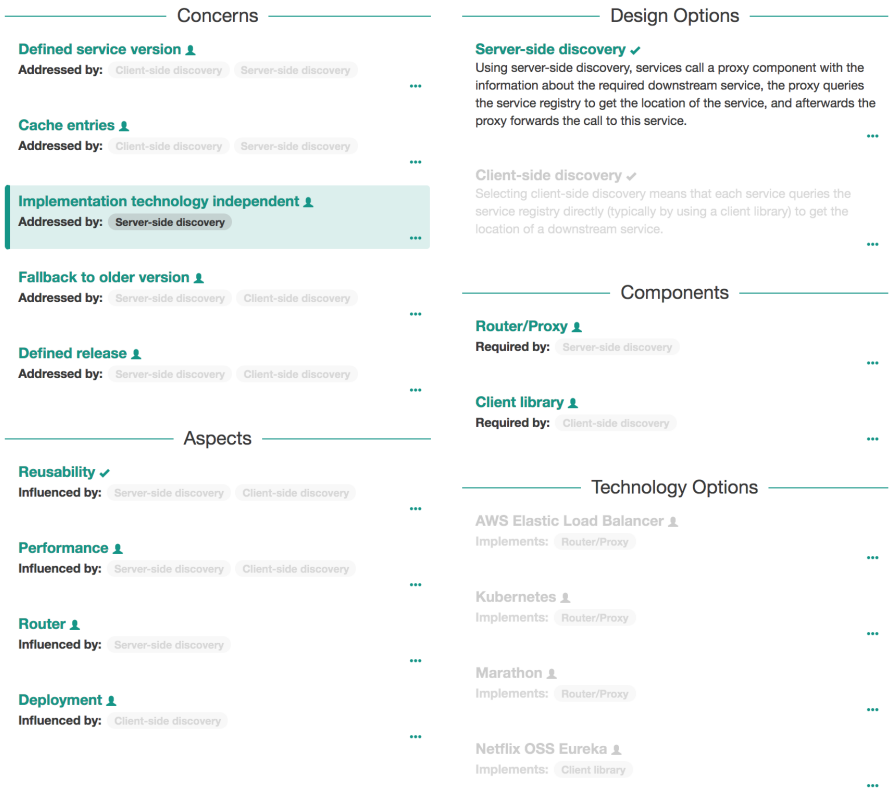


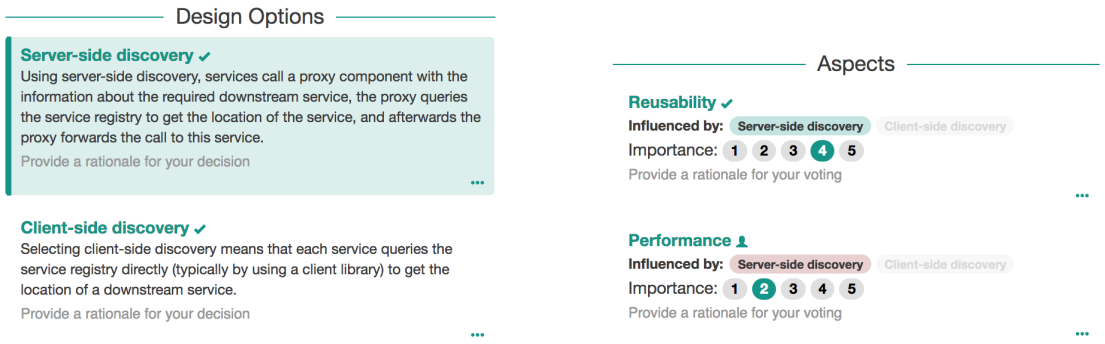
Figure 2.6: Preselection of a Concern

To provide a better overview of which decision models have already been used for decision making, the design process of a hierarchy shows additional information as soon as an architecture profile gets selected. For this purpose, the nodes in the design process representing decision models are colored regarding their status. Decision models that were already used for documentation in an architecture profile are colored green and decision models where the documentation is still unfinished are colored yellow. Unused decision models maintain the gray color. Figure 2.8 demonstrates the design process of an architecture profile regarding the Microservice API Management design space.

The figure shows that the first decision model in the design process was already used for documentation. The decision documentation in the second decision model is still unfinished and, hence, is colored yellow. The remaining decision models were not used for documentation in an architecture profile yet and, therefore, are colored in gray.

2.4 Discussion

The last sections of this chapter presented the AKB application with architecture profiles and decision models in particular. An architecture profile utilizes the architectural



(a) Design Options (b) Rating of Aspects

Figure 2.7: Decision Documentation

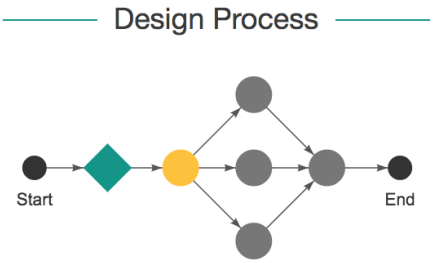


Figure 2.8: Design Process for Microservice API Management Documentation

haiku approach for software architecture documentation. Therefore, architecture profiles describe project-specific architectural knowledge and show the most important design decisions as well as rationales for these decisions.

Decision models, on the other hand, represent project-generic architectural knowledge in various design spaces, e.g., monitoring, service discovery, and fault tolerance. Decision models can be used for decision guidance and decision documentation. For decision guidance, decision models show design options in a design space, concerns that are addressed by these design options, the influences a design option has on certain aspects, and components and technology options that can be used to implement the design options. This way, software architects can evaluate the design options to make the best decision for a software system. Decision models can also be used for decision documentation. Thereby, a software architect may select an architecture profile in which the design decision should be documented. After that, the software architect selects the chosen design options and saves them into the architecture profile. The selected design options will be displayed as a design decision in the architecture profile. Additionally, the software architect can provide rationales and can assess the importance of aspects and concerns for this decision.

The next chapter focuses on answering the first research question of this thesis. For this purpose, the chapter serves as an introduction to recommender systems. The chapter

presents various types of recommender systems and describes how they work. This introduction forms the basis for the development of a recommendation system within the context of software architecture.

Chapter 3

Recommender Systems

Most internet users stumbled upon a recommender system even without them consciously noticing it. Just take for example Amazon, where a user gets recommendations based on the products that the user bought or viewed in the past. Another example would be Netflix, where a user receives movie recommendations based on the movies that the user watched in the past. All of these web applications use recommender systems with the goal to recommend items that might be interesting to a user.

Recommender systems are a relatively new research area in information systems compared to other fields like databases or search engines. In the mid-1990, recommender systems established itself as an autonomous research area [17]. Ever since they emerged in the early 1990's, these systems provide personalized recommendations and, thus, changed the marketing sector and helped web services to make better content available to their customers [18]. The main cause for the development of recommender systems was the rapid growth of information that became available through the internet. Regarding Amazon, for example, it would be impossible for a user to go through the whole product catalog to find the desired product. Therefore, besides search engines, recommender systems are used to narrow down relevant products and recommend them to a particular user [19].

Before discussing the use of a recommender system to support software architecture decisions, this chapter provides a general overview of different categories of recommender systems from a theoretical point of view. First, this chapter presents terms and concepts for a better understanding of the following sections. After that, the chapter deals with possibilities to categorize recommender systems. This is followed by the four main categories of recommender systems, namely collaborative-filtering, content-based, knowledge-based and hybrid recommender systems. Collaborative-filtering systems use the opinion and behavior of existing users to recommend items to the active user. The active user refers to the person for whom the system makes recommendations. For instance, the user-based collaborative-filtering method aims to find users that have a similar opinion or behavior

as the active user and then recommends items they liked in the past. Content-based recommender systems, on the other hand, use information about the items that the active user liked in the past to recommend items. If a user always liked action movies, it is most likely that the user will be interested in other action movies he or she has not seen yet. Knowledge-based recommender systems use explicitly provided information about the preferences of a user to make recommendations. For example, when buying a computer, a user could specify the desired amount of RAM, storage and the display size and the recommender system tries to find items that match these requirements the best. Finally, hybrid recommender systems are a combination of multiple recommender systems with the goal to produce better results than single recommender systems.

3.1 Terms and Concepts

Before introducing different categories of recommender systems, some basic terms and concepts need to be clarified. First of which is the term recommender system. Jannach et al. [20, p. 1] state that a recommender system is a software system that determines which items or products to present to a particular user. Ricci et al., on the other hand, define recommender systems as "software tools and techniques providing suggestions for items to be of use to a user." [17, p. 1]. The term item refers to the artifacts that the recommender system presents to the user, such as books, movies or articles [17, p. 1]. In general, there are two main use cases for recommender systems according to Jannach et al. [20, p. 3]. On the one hand, recommender systems are used to lead the user to certain actions like buying a particular product or watching a particular movie. For example, if a user bought a camera on a web-shop, the recommender system could present memory cards to lead the user to another transaction. On the other hand, recommender systems can serve as a filter function since only a certain number of items that might be interesting to the user are presented instead of the whole dataset. In other words, recommender systems can be used to deal with information overload [20, p. 3]. For example, a web-shop usually does not display all products but rather only the products that are most relevant to a particular user. Zhou and Luo [21] also state that recommender systems are an effective method to deal with information overload. In their view, internet applications like search engines or professional document indexing are important tools to filter information. However, unlike recommender systems, they lack personalized considerations.

In general, recommender systems can be distinguished into personalized and impersonalized recommender systems [20, p. 1]. Personalized recommender systems present recommendations that are tailored to a particular user based on the taste or past behavior [20, p. 1]. This means that every user sees a different list of recommendations. In contrast to personalized recommender systems, impersonalized recommender systems provide recommendations independently of any particular user. According to Jannach et. al [20, p. 1], impersonalized recommender systems provide recommendations that

many users are interested in, for example, top-selling products of a web-shop. The focus in the following sections lies on personalized recommender systems. Since the aim of this thesis is the development of a tool to support users when they make decisions for particular software architectures, it will not make sense to recommend decision models regardless of the type of software architecture. In case of a microservice architecture, the recommender system should only present items that are relevant in a microservice context. Thus, impersonalized recommender systems are not useful for this purpose.

3.2 Recommender System Categories

The number of categories differs significantly in the literature when it comes to recommender systems. The main criterion to assign a recommender system to a category are the knowledge sources that are used in the respective algorithms [22]. Some examples of knowledge sources are product databases, user rating databases or user demographic databases according to Burke [22]. In contrast, Tarus et al. state that "Recommender systems are classified according to the technique used in the recommendation." [23]. This means that not the types of knowledge sources determine the category of a recommender system, but rather how the knowledge sources are used in the respective algorithms. This shows that different criteria can be used to classify recommender systems.

Jannach et al. [20] classify recommender systems into collaborative-filtering, content-based, knowledge-based and hybrid recommender systems. They chose the types of knowledge sources that are used in the algorithms to classify recommender systems. As mentioned earlier, hybrid recommender systems are a combination of multiple recommender systems and, thus, are handled as a separate category. The knowledge source for collaborative-filtering recommender systems are the opinions and behaviors of a group of users to make recommendations to a particular user [20, p. 13]. Content-based recommender systems, on the other hand, use item descriptions and characteristics as well as user ratings as knowledge sources [20, p. 51]. Finally, knowledge-based recommender systems use the user's requirements and item descriptions to make recommendations [20, p. 82].

In addition to the four aforementioned categories, Burke [24] and Aggarwal [25] see demographic recommender systems as an additional category. These recommender systems use the demographic data of users to make recommendations. However, Jannach et. al [20, p. 125-126] see demographic recommender systems as a sub-category of collaborative-filtering recommender systems. This can be justified by the fact that demographic data is used as additional knowledge to identify similar users which is another variant of the collaborative-filtering approach [20, p. 126]. Additionally, Burke [24] defines utility-based recommender systems as a separate category. This category, however, can be seen as a sub-category of knowledge-based recommender systems according to Jannach et al. [20, p. 125]. Also Aggarwal [25, p. 18] states that utility-based approaches can be subordinated to knowledge-based recommender systems. According to

Tarus [23], many other more advanced recommender system approaches exist. For example, context-aware, trust-aware, fuzzy-based, social network-based, group-based, and ontology-based techniques. However, all of these approaches can be subordinated to one of the main categories of recommender systems.

It can be concluded that all of the different categorizations can be summarized into four main categories, namely, collaborative-filtering, content-based, knowledge-based, and hybrid recommender systems. In the next sections, these categories are explained in more detail. Furthermore, different algorithms are presented for each category. As for hybrid recommender systems, different techniques will be covered to show possibilities to combine multiple recommender systems to deliver better results.

3.3 Collaborative-Filtering Recommender Systems

Collaborative-filtering algorithms were the first recommender systems and are still the most popular and most used methods nowadays [26]. Especially in the field of online retail sites, collaborative-filtering recommender systems became prominent to promote products that might be of interest to a particular user to boost sales. In general, collaborative-filtering methods can be used to either calculate how much a user will like a particular item or to provide a list of items that the user might be interested in [20, p. 13]. The original idea of these recommender systems was to find users with similar behavior or opinions in the past as the active user and recommend items that they liked in the past [17, 20]. This idea has evolved over the last decades and lead to different variations of collaborative-filtering. However, all collaborative-filtering methods use the opinions or behavior that existing users had in the past to make recommendations [20, p. 13]. In the context of recommender systems, ratings are a means to express opinions or behavior of users. A commonly used method to get information about the opinion of a user for a particular item is to let the user explicitly rate the item [20, p. 22]. For example, by providing a rating scale from one to five stars for each item. Aggarwal [25, p. 10] declares that 5-point, 7-point, and 10-point rating scales are most common. Jannach et al. [20, p. 22] also state that the past behavior of a user can be used to implicitly determine the opinion of a user for a particular item. For example, if a user bought an item in the past, it is most likely that the user likes the item. Additionally, to implicitly determine the rating of a user for an item the system could also use the browsing behavior. In this case, the rating for an item would be positive if the user tried to get more details about the item.

To make recommendations collaborative-filtering methods only use the ratings that are either explicitly provided by the users or implicitly determined by the system [22]. Therefore, collaborative-filtering methods do not need any information about the users or products. The only input that these recommender algorithms need is the so-called rating matrix [20, p. 13]. Table 3.1 shows an example of a rating matrix.

User/Item	Item 1	Item 2	Item 3	Item 4
Active User	3	2	4	**?**
User 2	4	3	4	4
User 3	2	5	2	2
User 4	1	2	4	2

Table 3.1: Rating Matrix

In this example, a five-star rating was used to determine the opinion of users for items explicitly. Table 3.1 shows that the active user rated item 1 with three stars and item 2 with two stars. Item 4 has a question mark instead of a rating for the active user which indicates that the active user has not rated the item yet. The recommender system should calculate a forecast value for Item 4 for the active user using collaborative-filtering methods to determine if the user will like the item or not.

As mentioned earlier, different variants of collaborative-filtering emerged over the last decades. The classic and earliest method is user-based collaborative-filtering [20, p. 13]. The user-based method represents the original idea to find users with similar tastes in the past as the active user and recommend items that they liked. However, user-based collaborative-filtering has its drawbacks when it comes to large databases since the ratings of all users and items need to be taken into account when the system determines the items to recommend to the active user [20, p. 18]. This lead to another variant of collaborative-filtering, namely, the item-based method. According to Jannach et al. [20, p. 18], the idea of this method is to find similar items and not similar users. The recommender system then calculates a prediction for a particular item based on how the active user rated the similar items. The reason why item-based collaborative-filtering is more suitable for large databases is that the system can calculate the similarity between items offline. In general, calculating the similarity between users is also possible offline. However, Sarwar et al. argue that "a few additional ratings may quickly influence the similarity value between users." [27], but the similarity between items is more stable. Model-based methods are the third approach and deal with the scalability problem even further [20, p. 26]. Model-based methods process the rating matrix offline to create models using machine learning methods [25, p. 9]. At the time when the system makes recommendations, only the model is used instead of the rating database which leads to a significant benefit regarding the performance [25, p. 71]. The next sections examine the user-based, item-based and model-based method in more detail.

3.3.1 User-based Collaborative-Filtering

As explained before, the user-based method aims to identify similar users and uses their ratings to calculate predictions for items the active user has not rated, yet [25, p. 34]. First, the system needs to calculate the similarity between the active user and other users. Similarity measures are a means of computing the similarity between users [28]. Sureka and Mirajkar [29] explain that a similarity measure is a function whose result expresses the similarity between two elements. Since the quality of the recommendations depends on the selection of the right users, a suitable similarity measure is crucial for the user-based method [26]. Herlocker et al. [30] suggest using the Person coefficient since it produces better results for comparing users than other similarity measures. However, an empirical study by Sureka and Mirajkar [29] shows that a mix of different similarity measures delivers the best results.

Regarding explicit ratings, users often rate items differently even though they have the same opinion about it [31]. Some users hardly ever give one or five stars on a 5-point rating scale, and others do this regularly. Therefore, Ren et al. [31] suggest taking the average rating of a user into account. The Pearson coefficient includes this aspect. To demonstrate the formula as presented in Jannach et al. [20, p. 14] let $I = \{i_1, ..., i_m\}$ be the set of items in the database. $r_{a,i}$ expresses how the user a rated item i and $\overline{r_a}$ the average rating of user a. The similarity of user a and user b is now calculated using the $sim(a, b)$ formula shown in (3.1).

$$sim(a,b) = \frac{\sum_{i \in I}(r_{a,i} - \overline{r_a})(r_{b,i} - \overline{r_b})}{\sqrt{\sum_{i \in I}(r_{a,i} - \overline{r_a})^2}\sqrt{\sum_{i \in I}(r_{b,i} - \overline{r_b})^2}} \tag{3.1}$$

The Pearson correlation returns a value ranging from +1 which indicates a strong positive correlation to -1 which indicates a strong negative correlation [20, p. 15]. When applying the formula on the rating matrix shown in Table 3.1, following similarity values result as presented in Table 3.2.

	Similarity Value
User 2	0.853
User 3	-0.853
User 4	0.653

Table 3.2: Result of the Pearson coefficient formula

The result shows that User 2 is very similar to the active user. Also, User 3 is quite similar to the active user with a Pearson coefficient of 0.653. However, User 4 shows a high dissimilarity to the active user with a negative Pearson coefficient of -0.853. After using a similarity measure to calculate the similarities between the active user and other

users, the most similar users need to be selected. Jannach et al. [20, p. 17-18] present two options to determine similar users. Similar users are also called neighbors of the active user. First, a particular threshold can be set for the similarity value to select all users that are above that threshold. However, it is difficult to find a proper threshold that will return a suitable number of neighbors. A high threshold might return not enough or even no neighbors at all, and a low threshold could return too many neighbors which has a negative effect on the recommendations. The second option is a fixed number of neighbors. Other than a threshold, this option will deliver the right amount of neighbors but faces the problem that the list of neighbors could also include users who have low similarity to the active user. Herlocker et al. [30] state that a number of neighbors below ten negatively influences the results of the recommendations. They suggest that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" [30].

The next step of the user-based method is to calculate a prediction for all items the user has not rated, yet. To determine the prediction for item i, the user-based method uses the ratings of the similar users for that item [20, p. 15]. Jannach et al. [20] and Aggarwal [25] suggest the same formula to calculate a prediction for item i and the active user a. Let $N = \{b_1, ..., b_n\}$ be the set of neighbors for user a and $sim(a, b)$ the similarity between the active user and the neighbor b. The formula also takes the average rating of users into account with $\overline{r_a}$ being the average rating of the active user and $\overline{r_b}$ being the average rating of the neighbor b. The rating of a neighbor b for item i is expressed by $r_{b,i}$. Now, the formula to calculate the prediction is shown in Formula (3.2).

$$pred(a, i) = \overline{r_a} + \frac{\sum_{b \in N} sim(a, b) * (r_{b,i} - \overline{r_b})}{\sum_{b \in N} sim(a, b)} \tag{3.2}$$

In the example, User 2 and User 4 who are both more similar to the active user than User 3 could represent the neighborhood N. Their rating for Item 4 and their similarity to the active user are taken into account to calculate a prediction for Item 4 for the active user. The $pred(a, i)$ formula would return a value of 3.033 in this case. The result is slightly above three because User 2 has rated Item 4 with four stars and is more similar to the active user than User 4. In other words, the active user will rate Item 4 with three stars based the ratings of similar users. If the system should recommend a list of items, it will apply the $pred(a, i)$ formula to all items the active user has not rated, yet [20, p. 16]. After that, the system will recommend the items with the highest prediction values.

3.3.2 Item-based Collaborative-Filtering

As mentioned before, in contrast to the user-based method, the item-based method aims to find similar items and computes a prediction based on the ratings of the active user for these items. The item-based method is more suitable for large databases since the

recommender system can calculate the similarity between items offline [20, p. 18]. Sarwar et al. [27] explain that the similarity between users can change quickly even if users rate just a few new items. Therefore, the similarity measure cannot be applied offline. The similarity between items, on the other hand, is more stable over time.

The item-based method uses the rating matrix as input as well. The recommender system looks for items that were rated similarly to determine similar items [20, p. 18]. For example, in the rating matrix in Table 3.1 Item 4 was rated by users with $\{3, 2, 5\}$. The item-based method compares these ratings to the ratings of other items like Item 1 which was rated by users with $\{4, 2, 1\}$. Herlocker et al. [30] suggested using the Pearson coefficient to determine the similarity between users. However, the item-based method usually uses the cosine similarity [20, p. 19]. Both Jannach et al. [20] and Aggarwal [25] argue that an adjusted cosine similarity performs best for comparing items since it also takes the average rating of users into account. Jannach et al. [20, p. 19] propose a formula that defines $U = \{u_1, ..., u_n\}$ as a set of users. Then, $r_{u,a}$ expresses the rating of user u for item a and $\overline{r_u}$ the average rating of user u. The formula for the similarity $sim(a, b)$ between two items a and b is shown in Formula (3.3).

$$sim(a,b) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}} \tag{3.3}$$

Like the Pearson coefficient, the adjusted cosine similarity returns a value ranging from $+1$ which indicates a strong positive correlation to -1 which indicates a strong negative correlation [20, p. 19]. When applying the adjusted cosine similarity again on the rating matrix shown in Table 3.1 to find similar items to Item 4, similarity values as shown in Table 3.3 result.

	Similarity Value
Item 1	0.764
Item 2	-0.917
Item 3	0.118

Table 3.3: Result of the adjusted cosine formula

The result shows that Item 1 is the most similar item to Item 4 with an adjusted cosine similarity of 0.764. Item 2 is very dissimilar to Item 4 with an adjusted cosine similarity of -0.917 and Item 3 has a moderate similarity to Item 4 with 0.118. Both Jannach et al. [20] and Aggarwal [25] propose a formula to calculate a prediction for an item based on similar items. Let $I = \{i_1, ..., i_n\}$ be the set of similar items and $r_{u,i}$ the rating of the active user u for item i. The prediction $pred(u, p)$ for an item p is calculated as shown in Formula (3.4).

$$pred(u,p) = \frac{\sum_{i \in I} sim(i,p) * r_{u,i}}{\sum_{i \in I} sim(i,p)} \tag{3.4}$$

If the two most similar items to Item 4 represent the neighborhood, Item 1 and Item 3 would be used to calculate a prediction for Item 4, in the example. The result of the $pred(u,p)$ formula would return a value of 3.133. Even though one would expect a value of 3.5 since the active user rated Item 1 with three stars and Item 3 with four stars, the similarity to Item 1 is higher. Thus, the three stars have a higher impact on the prediction. Therefore, the item-based method comes to the same result as the user-based method which is that the active user will rate Item 4 with three stars.

3.3.3 Model-based Methods

Both the user-based and item-based method are also described as memory-based methods since the ratings are kept in memory to make recommendations [20, p. 26]. Model-based methods, on the other hand, use the rating database to create a model in advance to the recommendation phase. At the time of computing recommendations, only the model is used to make predictions [20, p. 26]. Also, Aggarwal [25, p. 71] states that user-based and item-based methods do not create a model to make recommendations but rather use some preprocessing steps to target performance issues. He also mentions that the model-building phase is completely separated from the actual recommendation phase since the model is created offline in advance. Models are created using machine-learning techniques that can be either supervised or unsupervised. Different methods exist to create a model. Aggarwal [25, p. 74-134] presents different approaches such as decision and regression trees, rule-based collaborative filtering, Naive Bayes collaborative filtering, and latent factor models.

From a theoretical point of view, memory-based methods deliver better results than model-based methods since they use all available data to calculate recommendations. However, as mentioned earlier, these methods face problems when it comes to large databases. The item-based method targets this problem by computing similar items in advance to the recommendation phase. However, since only the model is used to make recommendations, model-based methods perform best for large databases [20, p. 26].

3.3.4 Strengths and Weaknesses

One main advantage of collaborative-filtering recommender systems is the fact that the ratings of users for items are sufficient to make recommendations. There is no further information about the users or items required. Consequently, the system can also recommend different types of items. For example, if users who bought similar books as the active user watched a particular movie, the system could recommend the movie to

the active user. Also, collaborative-filtering recommender systems improve over time as more ratings become available. Finally, the domain of the application does not have any impact on the recommender system which means that no domain knowledge is needed for the implementation [24].

Since the comparison of users is based on the ratings alone, it is obvious that this is not possible if the active user has not rated any items yet. Also, if the active user has only rated a small number of items, the selected similar users are not meaningful. This problem is known as the cold-start or ramp-up problem [24, 32]. New items or items with few ratings face the same problem as the system will hardly ever or never recommend these items [32]. Jannach et al. [20, p. 26] state that the combination of the collaborative-filtering method and other types of recommender systems can solve both of these problems. Possible techniques to combine multiple recommender systems are presented in Section 3.6. Additionally, Burke [24] also highlights the "gray sheep" problem for collaborative-filtering methods. If a user has a unique behavior or opinion about items, the recommender system will not find any users that are somewhat similar. Thus, the quality of the recommendations will be low. Another problem identified by Burke [24] is the fact that it is hard for users to change their opinions. For example, if a user always liked action movies, but is now more interested in science-fiction, the recommender system will still recommend action movies because of the ratings made in the past. As Jannach et al. [20, p. 23] explain, in a real-world scenario, users will only rate a small number of all items in the database. However, it is difficult to provide good recommendations if only a small number of ratings is available. This issue is also called data sparsity problem and can be addressed when combining collaborative-filtering methods with other types of recommender systems. Aggarwal [25, p. 9] states that many model-based methods address data sparsity better than memory-based approaches.

3.4 Content-based Recommender Systems

As explained in the previous section, collaborative-filtering methods use only the ratings of users for items as the knowledge source. Therefore, collaborative-filtering recommender systems do not require any information about users or items. In contrast, content-based methods use both the ratings and the description of items to make recommendations [20, p. 51]. First, content-based methods use the ratings of a user to identify items the user liked in the past. Then, the content of these items is used to find similar items which are then recommended to the user [17, p. 11]. In contrast to the item-based collaborative-filtering method that uses the ratings of users to find similar items, content-based methods use the description of items for this purpose. Content-based methods create a so-called user profile that includes the preferences of a user [20, p. 51]. For example, in a scenario where the system should recommend movies, a user profile could hold the genres and favorite actors of movies the user liked in the past. Finally, the system compares the information stored in the user profile with the attributes of the

items the active user has not rated yet [17, p. 73]. After that, the system recommends the items with the best match.

The following sections introduce the procedure of content-based recommender systems which includes three steps [17, 25]. In the first step, the system preprocesses the content of items to prepare them for the next steps. The goal of this step is to bring the content of items in a uniform structure. This step is especially necessary for unstructured data such as texts [17, p. 75]. In the second step, the system uses the preprocessed item information of the items the user liked in the past to create a user profile. Usually, content-based methods use classification or regression modeling to create a user profile. In the final step, the system uses the user profile to filter items that might be interesting to a particular user. It is important to highlight that the recommender systems can perform the first two steps offline. The system uses only the user profile to filter relevant items in the recommendation phase [25, p. 142]. Each of these steps will be explained in more detail in the following sections.

3.4.1 Preprocessing Data

Lops et al. [17] state that the purpose of this step is the representation of the content of items. In other words, the goal is the analysis of the content and bringing it in a structured form for further processing. For example, a movie could be represented by a list of features like the genre, actors and year of release [20, p. 52]. Aggarwal describes the first step of content-based recommender systems as "preprocessing and feature extraction." [25, p. 142f]. For example, feature extraction techniques determine important words in unstructured data like texts. Furthermore, he explains that it is obvious that the content of items varies between different domains. A movie and an article will have different contents. Therefore, the preprocessing and feature extraction step differs between applications.

Usually, content-based methods extract keywords from unstructured content of items like texts [17, 25, 20]. As explained before, the system applies feature extraction techniques for this purpose [17, p. 75]. Jannach et al. [20, p. 54] presents a possible technique which is term-frequency inverse-document-frequency TF-IDF. Yun-tao et al. [33] define TF-IDF as follows:

> "The term (word) frequency/inverse document frequency (TF-IDF) approach is commonly used to weigh each word in the text document according to how unique it is. In other words, the TF-IDF approach captures the relevancy among words, text documents and particular categories." [33]

Therefore, when applying TF-IDF on a text, the most relevant words get extracted. Term-frequency stands for the number of occurrences of a word in a text. The idea behind

the inverse-document-frequency is that words that often appear in texts of different items are less useful when it comes to segregating texts [20, p. 55]. To bring an unstructured text in a proper format, there are a few steps to keep in mind according to Aggarwal [25, p. 145]. First, stop-words need to be removed. Stop-words are words that appear on a high frequency like "a", "the", or "and". Furthermore, a word can appear in different variations in a text, for example, in singular and plural form. In this case, the system should not treat these words as two different ones. Finally, the content-based method should identify phrases like a "take-off". Here, the system should not treat the words "take" and "off" as two separate words since they have a different meaning than "take-off".

3.4.2 Creating a User Profile

In the second step, a so-called profile learner creates a user profile based on the item representations of the user's preferences [17, p. 75]. Different approaches exist to create such a user profile. Jannach et al. [20, p. 52f] describe a simple technique that just stores the attributes of items the user liked in the past in the user profile. Regarding the movie application example, the profile learner could create a user profile that contains the genres and actors of the movies the user liked in the past. Furthermore, if the movies have a description, the profile learner could store the extracted keywords using the TF-IDF technique in the user profile. If the description of a movie contains words like "crime" and "guns", other movies that also have these words in the description might be of interest. It is also important to highlight that the user can extend the user profile by explicitly providing preferences. For example, the profile learner could implicitly determine the preferred genres based on the movies the user already watched, but ask the user explicitly to provide preferred actors.

Another approach for creating a user profile is to use machine learning techniques. In this case, items the user liked or disliked in the past refer to the training data. To be more specific, the item representations from the preprocessing step are used together with the ratings of the user for these items. Machine learning techniques like classification or regression methods create a user-specific model utilizing this training data. The resulting model determines whether a user will like or dislike an item. Therefore, the model refers to the user profile [25, p. 142]. Pazzani and Billsus [34] also state that classification methods are used to create a model based on the user's history. The training data is classified based on the ratings of the user which, again, can be either explicit or implicit. If the example movie application uses explicit ratings, the training data could be categorized into "movies the user liked" and "movies the user did not like". Regarding implicit ratings, the movies could be categorized into "movies the user watched" and "movies the user did not watch". Many classification and regression methods exist to create a user profile. Prominent methods are for example decision trees, nearest neighbor methods, the Rocchio's algorithm, and Naive Bayes [34]. According to Aggarwal [25, p. 163], the Bayes approach is a successful method for content-based recommender

systems since it can deal with different types of content.

3.4.3 Filtering and Recommendation

Finally, the last step of the content-based method is the use of the user profile to make recommendations. As explained before, the system preprocesses items and creates user profiles in advance to the recommendation phase. At the time when the user demands recommendations, only the constructed user profile is used [25, p. 142]. Content-based methods usually assess how similar an item is to the user's preferred items in the past to decide if the item should be recommended or not. Jannach et al. [20, p. 54] present one way if the user profile simply contains attributes of items the user liked in the past. The recommender system could check if the user profile holds an attribute of a given item and recommend it if this is the case. For example, if "action" is the genre of a movie and the user profile contains "action" in the list of preferred genres, the movie might be of interest to the user. Another approach would be to use similarity measures to evaluate the similarity between an item and the user profile. Jannach et al. [20, p. 54] propose the use of the Dice coefficient to measure how much the attributes of an item and the user profile overlap. In this case, a movie that has almost identical lists of genres and actors as in the user profile has a higher potential to be recommended than a movie that just matches one genre.

However, a user profile can also refer to a model that was created using classification or regression methods. In this case, the model gets executed on items the user has not rated yet [25, p. 142]. For example, if the user profile is a classification model, the recommender system could classify movies the user has not rated yet into "movies the user likes" and "movies the user does not like".

3.4.4 Strengths and Weaknesses

An important benefit of content-based recommender systems is that they do not face the cold-start problem of items. As explained in Section 3.3.4, new items that have not been rated yet will hardly ever be recommended using collaborative-filtering methods. Since content-based methods calculate the similarity between items based on their content and not their ratings, even new and unrated items can be recommended [25, p. 161]. Similar to collaborative-filtering methods, the content-based approach delivers better results over time since the user will rate more items and the user profile will learn more about the user's preferences. Moreover, implicit ratings are sufficient to make recommendations [24]. For example, whether a user liked or disliked a movie could be determined by checking if the user watched the movie until the end.

Even though content-based recommender systems can deal with new items, the cold-start problem for new users still exists. It is necessary that a user rates a certain amount

of items to learn about the user's preferences and make recommendations [24]. Aggarwal [25, p. 162] even points out that the cold-start problem of users is even more critical in content-based than in collaborative-filtering methods. The reason is that the machine learning techniques are dependent on the items the user already rated. Therefore, a reasonable number of rated items is required to create a suitable model. Another weakness compared to collaborative-filtering methods is that content-based systems will only recommend items that are similar to those the user liked in the past. This limits the user to discover new items that are completely different to those already liked, but still interesting to the user [25, p. 162]. For example, if the user only watched action movies in the past, the content-based method will only recommend other action movies. However, the user could also be interested in other genres like drama or comedy.

3.5 Knowledge-based Recommender Systems

Both collaborative-filtering and content-based systems make recommendations based on the behavior or preferences of a user in the past. However, this information about the user needs to be acquired and extended over time to make suitable recommendations. In some situations, the user will not provide this information regularly. For example, a user will usually not buy a house, a camera, or a car on a regular basis. Moreover, if a person is about to buy a new car, it should probably be different from the ones the user bought before. In these cases but also generally in recommender systems, the time span is a significant factor. The opinion of a user will change over time which makes ratings for items that were made years ago less meaningful [20, p. 81]. Additionally, Aggarwal [25, p. 167] states that both collaborative-filtering and content-based methods are less useful for items that can be highly customized by the user. For example, when buying a house, the number of bedrooms might be important to one user, but a balcony more relevant to another user.

Knowledge-based recommender systems target these issues. First, they do not rely on any user history, and second, they let users explicitly provide their requirements for items. In contrast to collaborative-filtering and content-based methods, users can always change their requirements in knowledge-based recommender systems [25, p. 168f]. The system could provide an interface like a web-based form to let the users define their requirements [20, p. 87]. No strict rules for the design of such an interface exist. However, it should lead the user through an interactive recommendation process [25, p. 170].

According to Jannach et al. [20], Aggarwal [25] and Felfernig et al. [35], two main types of knowledge-based methods exist, namely case-based and constraint-based. The case-based approach applies similarity measures to determine how similar an item is to the requirements of a user. After that, the system will recommend the most similar items to the user. In case of a computer shop application, for example, the user could specify the desired display size, processor performance, and hard-drive storage and the case-based system will recommend computers that are most similar to these requirements. The

constraint-based method, on the other hand, uses predefined recommendation rules to relate the requirements of a user to item attributes. Thereby, the system filters out items that fulfill these rules. For example, a rule for the computer shop application could be that the display size must be greater or equal to 15 inches if the user specifies to use it for gaming [35]. The case-based and constraint-based methods will be explained in more detail in the following sections.

3.5.1 Constraint-based

As mentioned before, the constraint-based method uses a set of recommendation rules to filter relevant items. Thereby, the requirements or constraints specified by the user restrict particular item attributes. Regarding the computer shop application, for example, the user could specify that the computer should have more than 4GB RAM. However, the user might not always know about the meaning or impact of the item attributes on the product. Therefore, the system might let the user specify requirements that are different from the item attributes. Recommendation rules are used to link the requirements to the item attributes. For example, instead of explicitly asking the user how much RAM the desired computer should have, the system could let the user define for what purpose the computer will be used. If the computer will be used for gaming, more RAM will be needed than just for browsing the web. In this case, a recommendation rule could be that if the usage of a computer is for gaming, the computer must have at least 8GM of RAM [25, p. 172]. However, domain knowledge is required to define such recommendation rules. Therefore, a domain expert is needed for this task [36].

Both Jannach et al. [20] and Felfernig et al. [35] demonstrate a more detailed view of the constraint-based procedure in a similar way. First, two sets of variables and three sets of constraints need to be defined:

- Customer Variables V_C: These variables represent what a user can specify as requirements. The system will present these to the user for selection, typically as a web form.
- Product Variables V_{PROD}: This set of variables describe possible properties of items.
- Compatibility constraints C_R: These constraints restrict the selection of possible user requirements.
- Filter conditions C_F: These conditions build the link between user requirements and item attributes. If a condition is not met by an item, it will not be recommended.
- Product constraints C_{PROD}: This set represents the available items.

Regarding the computer shop application, following sets could be defined:

- V_C = { max-price(300 ... 3000), usage(gaming, high performance, standard) }
- V_{PROD} = { price(300 ... 3000), RAM(3GB ... 16GB), storage(128GB ... 2TB), displaysize(12",13", 15") }
- C_R = { usage = high performance $\rightarrow$ max-price $\geq$ 1000 }
- C_F = { usage = gaming $\rightarrow$ displaysize $\geq$ 15", usage = high performance $\rightarrow$ RAM $\geq$ 16GB }
- C_{PROD} = { (id=c1 $\wedge$ price=1250 $\wedge$ RAM=8GB $\wedge$ storage=256GB $\wedge$ displaysize=15") $\vee$ (id=c2 $\wedge$ price=2450 $\wedge$ RAM=16GB $\wedge$ storage=1TB $\wedge$ displaysize=12") }

As defined in the customer variables, the user can specify a maximum price between 300 and 3000 and the usage of the computer which can be either $standard$, $gaming$, or $highperformance$. The items, which are computers in this case, have a $price$, the amount of RAM, $storage$, and the $displaysize$. As mentioned before, C_R restrict the available user selections. In this case, the user must specify a max-price of greater or equal to 1000 if the usage of the computer is $highperformance$. C_F contains two filter conditions that map the user requirements to item attributes. On the one hand, the $displaysize$ must be greater or equal to 15" if the usage is for gaming. On the other hand, computers must have more or exactly 16GB of RAM if the usage will be for $highperformance$ tasks. Finally, C_{PROD} contains all available computers from the catalog. If the user defines requirements REQ, the system applies the filter conditions C_F on the products C_{PROD} using the requirements. This process returns a set of products as a result RES. For example, if the user defines requirements as shown below, following products will be recommended as a result RES:

- REQ = { usage=gaming, max-price=1500 }
- RES = { id=c1, price=1250, RAM=8GB, storage=256GB, displaysize=15" }

First, the user specified $gaming$ as the usage of the desired computer. Based on the filter conditions, the display size must be greater or equal to 15". Furthermore, the maximum price for the computer is 1500. If the constraint-based system applies these conditions to the set of products C_{PROD}, it turns out that computer $c1$ fulfills these criteria. Therefore, the system will recommend $c1$ to the user.

3.5.2 Case-based

In contrast to the constraint-based method, case-based systems use similarity measures to calculate the similarity between the requirements of the user and items based on

their attributes [20, p. 86]. This approach is well suited in domains where items have a clear set of attributes like the price, size or weight [37]. The case-based method will recommend items that are most similar to the requirements of a user. Other than the constraint-based approach, the system does not apply any constraints or restrictions on the requirements. For example, the user could specify that the desired computer should have 1024GB of RAM. Even though there will not be any computer with 1024GB of RAM in the catalog, the system can still determine items that match this requirement the best. As explained in the previous section about constraint-based recommender systems, the result of these systems can be empty if no items fulfill the filter conditions. This problem does not exist in the case-based approach since the system will always recommend the most similar items even though they are considerably dissimilar to the requirements [25, p. 181].

Many similarity measures exist to calculate the similarity between two elements. A proper similarity measure is decisive to recommend suitable items [25, p. 183]. McSherry [38] proposes a similarity measure $similarity(p, REQ)$ for case-based recommender systems which is shown in Formula (3.5).

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r} \tag{3.5}$$

In principle, to calculate the similarity between an item p and the set of requirements of a user REQ, the system needs to calculate the similarity between each requirement r and the corresponding item attribute using the similarity function $sim(p, r)$. The formula shows that each requirement is weighted to incorporate the importance of an individual requirement using w_r. The sum of the weighted similarities between each requirement r to the item p results in an overall similarity.

McSherry [38] presents three possibilities to compute the similarity between a requirement r and the corresponding item attribute. It is important to notice that item attributes might exist that should be as high as possible like the performance of a computer. On the other hand, some item attributes should be as low as possible like the price. McSherry [38] defines these attributes as $more - is - better$ and $less - is - better$ attributes. A formula for the $more - is - better$ approach is shown in Formula (3.6).

$$sim(p, r) = \frac{\pi_r(p) - min(r)}{max(r) - min(r)} \tag{3.6}$$

In this formula, $\pi_r(p)$ refers to the value of the item attribute of a given item p and $min(r)$ and $max(r)$ to the minimum and maximum values of the item attribute in the catalog. The $less - is - better$ approach is shown in Formula (3.7).

$$sim(p,r) = \frac{max(r) - \pi_r(p)}{max(r) - min(r)} \tag{3.7}$$

Finally, if the similarity between a requirement r and an item attribute should be as similar as possible, following formula can be used:

$$sim(p,r) = 1 - \frac{|\pi_r(p) - r|}{max(r) - min(r)} \tag{3.8}$$

The case-based system applies the similarity measure $similarity(p, REQ)$ to each item in the catalog. After that, the system will recommend the items with the best similarity value. In other words, the items that are most similar to the requirements of the user.

Regarding the computer shop application, a user could specify a desired price of 1000, $8GB$ RAM and a display size of 13". After examining the item attributes it is clear that the price is a $less-is-better$ attribute. Most users will want the price to be as low as possible. Both the RAM and storage are probably $more-is-better$ attributes because the user will want them to be as high as possible. The display size, on the other hand, should be as equal to the user requirement as possible. Since the user did not specify a desired storage of the computer, it will not be taken into account to calculate the similarity. To calculate the similarity between the requirements and computer $c1$, the price would be calculated as follows:

$$sim(p,r) = \frac{max(r) - \pi_r(p)}{max(r) - min(r)} = \frac{2450 - 1250}{2450 - 1250} = 1 \tag{3.9}$$

Since the price of computer $c1$ is the lowest in the example, the similarity is equal to 1. The RAM using the $more-is-better$ approach can be calculated as follows:

$$sim(p,r) = \frac{\pi_r(p) - min(r)}{max(r) - min(r)} = \frac{8 - 8}{16 - 8} = 0 \tag{3.10}$$

The result shows that the RAM of computer $c1$ is completely dissimilar to the requirement. The reason is that computer $c1$ has the lowest amount of RAM in the example, but the similarity is calculated using the $more-is-better$ approach. Finally, the case-based system will calculate the similarity of the display size as follows:

$$sim(p,r) = 1 - \frac{|\pi_r(p) - r|}{max(r) - min(r)} = 1 - \frac{|15 - 13|}{15 - 12} = 0.3333 \tag{3.11}$$

Now that the similarities between each requirement to the corresponding item attribute was calculated, the overall similarity to item p needs to be computed. Before that, the

item attributes can be weighted. In this case, for example, a domain expert weighted the price with 3, the RAM with 1 and the display size with 2. An example calculation of the user requirements and computer $c1$ in the catalog is shown in Table 3.4.

Attribute	**REQ**	**Computer c1**	$sim(p_i, r_i)$	w_r	Summe
Price	1000	1250	1	3	3
RAM	8GB	8GB	0	1	0
Display Size	13"	15"	0.3333	2	0.6666
			Sum:	6	3,6666

Table 3.4: Example similarity calculation

The overall similarity between the requirements of the user REQ and the computer $c1$ is then calculated as follows:

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r} = \frac{3.6666}{6} = 0.6111 \tag{3.12}$$

The result shows that computer $c1$ has a similarity of 61.11% to the user requirements. As mentioned before, the case-based recommender will calculate the similarity to each item in the catalog and recommend the items with the best similarity values.

3.5.3 Strengths and Weaknesses

In contrast to collaborative-filtering and content-based methods, knowledge-based recommender systems do not face any cold-start problem. Since the user defines preferences explicitly in the form of requirements, knowledge-based systems do not need any past behavior or opinion of the user to make recommendations. Therefore, no user cold-start problem exists. Furthermore, as with content-based recommender systems, knowledge-based systems also take new items that have not been rated before into account when making recommendations. Since knowledge-based methods only use the content of items to make recommendations, no ratings are required which means that no item cold-start problem exists [24].

Knowledge-based systems can also adapt to changes in a user's preferences as the user can always change the requirements. Additionally, the user can specify requirements that are not item attributes. Especially in the constraint-based method, the system uses filter conditions to map user requirements to item attributes [24].

However, knowledge-based recommender systems do not learn about user preferences

over time like collaborative-filtering and content-based systems do. The reason for this limitation is that knowledge-based methods do not use the rating database to compute predictions. Therefore, the output of a knowledge-based recommender will not change if the user provides more ratings to the system. As mentioned earlier, both collaborative-filtering and content-based methods do not require any domain knowledge to operate. In contrast, knowledge-based recommender systems depend on domain knowledge. For example, a domain expert is required to define recommendation rules for constraint-based systems. In this case, the recommendation rules are the domain knowledge needed for making recommendations [24].

3.6 Hybrid Recommender Systems

Three different categories of recommender systems have been presented in the previous sections. Each category has its strengths and weaknesses. Collaborative-filtering systems have the advantage that they can recommend items of a different type. Furthermore, just like content-based systems, they learn about the user's preferences over time as more ratings become available. Content-based systems do not face the item cold-start problem since they use the content of items to find similar items instead of their ratings by users. Knowledge-based systems go even further and do not have any cold-start problem. Additionally, they are much more sensitive to changes in a user's requirements. Therefore, each category performs best in a particular situation. However, most applications face multiple recommendation situations over time. For example, a new application will not have enough ratings in the database to perform collaborative-filtering, but as soon as the number of ratings grows, collaborative-filtering might deliver better results than, for instance, a knowledge-based system.

The goal of hybrid recommender systems is to exploit the benefits of two or more individual recommender systems. Thereby, a hybrid recommender combines multiple recommender systems into a single system to achieve better recommendations [22]. In the following, the recommender system refers to the final hybrid recommender system, and a recommender refers to an individual recommender system that gets combined with other recommender systems. Seven different techniques are introduced in [20, 25, 22] to combine multiple recommender. Jannach et al. [20, p. 128] classifies them into monolithic, parallelized and pipelined hybrids. Aggarwal [25, p. 200], on the other hand, categorizes the hybridization techniques into monolithic, ensembles and mixed techniques. Additionally, he categorizes ensembles techniques even further in parallel and sequential as shown in Figure 3.1.

The figure shows that both feature-combination and meta-level belong to monolithic hybridization techniques. Feature-augmentation, cascade, weighted and switching are ensembles techniques, and finally, the mixed technique is a category for its own. The next sections explain the different techniques in more detail.

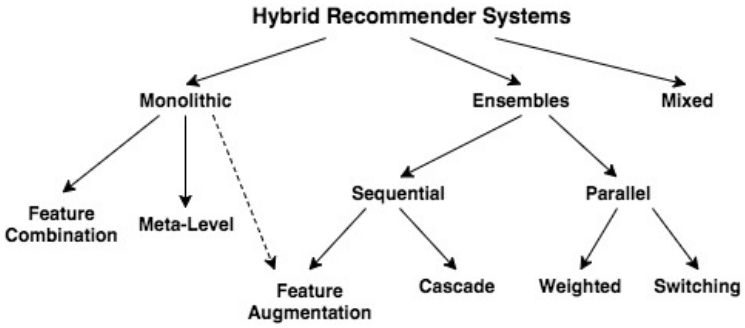


Figure 3.1: Hybridization Techniques [25, p. 201]

3.6.1 Mixed

When the hybrid recommender system applies this technique, the result of each recommender will be presented to the user [20, 25, 22]. Therefore, the purpose of this technique is not to merge the outcome of multiple recommenders [24]. The recommendation methods get executed in parallel and independently from each other [22]. The system will present the results of each recommender side by side to the user.

3.6.2 Ensembles

Ensembles techniques use off-the-shelf recommender algorithms and execute them either in parallel or sequential. Both the switching and weighted technique can be executed in parallel. Other than the mixed technique, the hybrid recommender system returns only one outcome. On the other hand, cascade and feature augmentation technique execute multiple recommender sequentially [25, p. 200].

Switching

The first step of switching is equal to the mixed technique. Multiple recommender algorithms get executed simultaneously. However, instead of returning the outcomes of all recommenders, switching conditions determine which results to present to the user. In other words, depending on the current situation, the recommender system decides which recommendations should be displayed to the user [20, p. 137f]. According to Aggarwal [25, p. 201], for example, a hybrid recommender system could use the results of a knowledge-based recommender in early stages of the application. Once enough ratings are available in the database, the system could switch to the results of a collaborative-filtering algorithm.

Weighted

Same as with the mixed and switching techniques, recommender algorithms get executed in parallel using the weighted approach. However, the weighted technique computes a recommendation score for each item based on the results of the individual recommenders [22]. The importance of each recommender can be defined by assigning them a weight factor [20, p. 135]. Jannach et al. [20, p. 137f] present a function $rec_{weighted}(u, i)$ to determine the combined score of n recommenders for an item i for user u which is shown in Formula (3.13).

$$rec_{weighted}(u, i) = \sum_{k=1}^{n} w_k * rec_k(u, i) \tag{3.13}$$

The function sums up the score for an item of each recommender and multiplies it by the relative weight w_k of each recommender. Same as with the switching technique, the hybrid recommender system will only return one recommendation list.

Cascade

As mentioned before, recommenders get executed sequentially when the cascade technique is in use. The task of each recommender is to refine the recommendations made by the previous recommender [20, p. 138f]. Therefore, the output of a recommender is the input for another one [25, p. 213]. Burke [24] states that the task of the first recommender is to produce a roughly ranked list of potential candidate items. The second recommender refines the candidate items and their ranking to create a final recommendation list. It is important to highlight that the recommended items highly depend on the previous recommenders. [20, p. 138f]. If the first recommender does not view a particular item as a potential candidate, it will not pass the item to the next recommender. Therefore, the next recommender will not consider this item in the refinement of the candidates.

Feature Augmentation

Same as with the cascade technique, recommenders get executed sequentially in the feature augmentation technique. However, the difference is that the goal of the first recommender is to predict a classification or rating of an item. The next recommender uses this information about the item to make recommendations for a user [24]. Jannach et al. [20, p. 132ff] explain that the purpose of the first recommender is to create features that augment the knowledge source of the second recommender. For example, a content-based method could be used to create pseudo-ratings for items. These pseudo-ratings could be used to augment the rating matrix for a collaborative-filtering recommender.

Thereby, the pseudo-ratings help to find similar users and to deal with the data sparsity problem.

3.6.3 Monolithic

The goal of monolithic techniques is to create a single recommendation algorithm that incorporates aspects of different recommenders. The individual parts of these hybrid recommender systems cannot be unambiguously assigned to a recommender method, meaning that it is not clear that a particular part of the system is a collaborative-filtering approach, for example. Therefore, in contrast to mixed and ensembles techniques, monolithic approaches do not use recommender algorithms as black-boxes but rather modify them [25, p. 200]. As shown in Figure 3.1, both feature combination and meta-level are monolithic hybridization techniques.

Feature Combination

The feature combination technique uses various types of input data to make recommendations. The input data refers to the knowledge sources used by recommender systems [20, p. 130]. For example, a hybrid approach using feature combination could use the rating matrix in conjunction with the content of items to compute recommendations. In this case, the rating matrix would be more of a collaborative-filtering aspect and the content of items a content-based aspect. Aggarwal [25, p. 217] presents an example in which the original rating matrix gets extended by additional keywords that got extracted using content-based methods. Therefore, the system not only takes the ratings of users for items into account to find similar users but also keywords of items the users liked in the past.

Meta-Level

The purpose of this technique is that one recommender method creates a model that is then used by another recommender. In contrast to feature combination where a combination of various input data gets passed to the next recommender, a whole model gets passed using the meta-level technique [20, p. 139f]. Aggarwal [25, p. 216f] presents an example that combines a collaborative-filtering with a content-based method. In the first step, the system uses content-based techniques to create a user-keyword matrix. The keywords get extracted from the items users liked in the past, and the cell refers to the weight of the word using TF-IDF, for instance. For the movie application example, the output of this step could look as shown in Table 3.5.

The example shows keywords as columns that were extracted from the movie descriptions

User/Keyword	crime	gun	detective	love	wedding
User 1	2.1	2.4	1.8	1.2	0
User 2	2.7	1.3	2.2	2.4	0.8
User 3	0.4	0	1.1	2.3	1.7

Table 3.5: User-Keyword Matrix

using TF-IDF. Based on the example presented by Aggarwal [25, p. 216f], the hybrid recommender system uses this matrix to find similar users using collaborative-filtering techniques. However, for the final recommendation, the system uses the original rating matrix.

3.6.4 Recommender Combinations

The previous sections presented different techniques to combine multiple recommenders to make precise recommendations in various situations. However, for some techniques, it is not possible to combine particular recommender methods. Burke [24] carried out a survey in 2002 to examine what recommender combinations are possible and implemented in the real-world based on the different techniques. The results of the survey for combinations of collaborative-filtering, content-based, and knowledge-based methods is shown in Table 3.6.

	Weighted	Mixed	Switching	FC	Cascade	FA	ML
CF/CN							
CF/KB							
CN/CF							
CN/KB							
KB/CF							
KB/CN							

Table 3.6: Possible Recommender Combinations [24]

The color of a cell determines if the combination of the two recommender methods is not possible (black cells), if there are any existing implementations (light gray cells), or if a combination is redundant (gray cells). A combination is redundant in cases where it does not matter if one recommender gets executed before the other. For example, it does not matter if a collaborative-filtering method gets executed before a content-based

method or the other way around using the mixed hybridization technique. White cells determine that a combination of two recommenders using a particular technique is rather unexplored since no known implementations existed so far.

The results show that the feature combination technique is not possible in combination with a knowledge-based recommender. Burke [24] argues that the reason is that knowledge-based methods allow any kind of data which makes them unsuitable for feature combination.

3.6.5 Strengths and Weaknesses

Each category of recommender systems has its strengths and weaknesses. For example, collaborative-filtering methods are not suitable for cold-start but might produce better results as soon as more ratings become available. Hybrid recommender systems aim to exploit the strengths of different types of recommenders. For this purpose, the system uses techniques to either enrich the input data or to increase the performance to produce more robust recommendations [25, p. 221f]. Jannach et al. [20, p. 141] states that the use of hybridization techniques improves the results of basic recommender methods. However, not enough empirical studies about the different techniques exist to conclude main advantages and disadvantages for each of them.

3.7 Summary

The goal of a recommender system is discussed in the first part of this chapter and can be summarized as the determination of items that might be interesting to a user. The recommender system can either use information about users like their preferences or behavior to make personalized recommendations or provide impersonalized recommendations that match the interest of most users. Since this thesis focuses on recommendations for a particular software architecture, this chapter only investigates personalized recommender systems. Many ways exist in the literature to categorize recommender methods. However, Jannach et al. [20] state that all recommender methods can be classified into one of four main types of recommender systems. These four types are collaborative-filtering, content-based, knowledge-based and hybrid recommender systems. Therefore, this chapter presents basic methods of these four categories.

Collaborative-filtering approaches were the first recommender systems and are still the most applied methods nowadays. The only information needed to make recommendations is the rating database. In other words, how users rated items in the past. As explained in Section 3.3, the three main collaborative-filtering methods which are user-based, item-based, and model-based exploit the rating matrix to make recommendations. The user-based method searches for users who rated items similar to the active user to recommend

items they liked. In contrast, item-based aims to find items that were rated similarly to a given item to calculate a prediction based on the ratings of the active user for these items. Finally, in the model-based approach, the system applies classification or regression methods to create a model based on the rating matrix. Only the model is used to predict potential items. However, collaborative-filtering methods face the cold-start problem of both users and items. Therefore, these methods are only suitable for applications with a large user and rating database.

Content-based methods, on the other hand, need information about the content of items as well as the ratings of the active user to make recommendations. First, the system prepares the content of items, for example, by using text-mining techniques and creates a user profile based on the items the user liked in the past. Finally, content-based methods use the user profile to determine if a user will like an unrated item or not. Content-based recommender systems overcome the cold-start problem for items since the content of items is used to determine their similarity. The cold-start problem for users still exists in these systems.

The third category is knowledge-based recommender systems. These methods explicitly obtain the preferences of a user in form of requirements and do not depend on any user ratings. There are two knowledge-based methods which are case-based and constraint-based. The case-based method calculates the similarity between the requirements and the content of items to recommend the most similar items. The constraint-based method, on the other hand, use predefined recommendation rules to filter potential items. Since knowledge-based systems do not need any information about the past behavior or opinions of users in the past, they do not face cold-start problems.

Each recommender system category has its strengths and weaknesses. Hybrid recommender systems combine multiple recommenders to exploit their advantages. This chapter presents seven different techniques to merge recommenders. The mixed technique presents the outcomes of multiple recommenders side by side. Ensemble techniques execute different recommenders either in parallel or sequential. Monolithic techniques merge aspects of several recommender methods to create one robust system. Hybrid recommender systems usually deliver better results than individual recommender systems.

Chapter 4

System Requirements and Concepts

The previous chapter introduced recommender systems and presented different methods to create recommender systems in general. This chapter focuses on an approach to develop a recommender system to support software architects in the decision-making process. For this purpose, a recommender system was developed as part of the AKB tool presented in Chapter 2. This chapter shows the procedure for creating the concept of the recommender system. First, this chapter introduces aspects of the AKB tool that are relevant for the recommender system. These aspects include the top-level software architecture, applied technologies, as well as the data model. The second part deals with requirements that were identified. The requirements include the kind of items that should be recommended and the kind of data that should be used to make recommendations. The last part deals with the procedure of selecting the most suitable recommender method for each requirement and visualizes the overall concept of the recommender system.

4.1 Context

Before going into more detail about the concept of the recommender system, this section provides some context about the AKB tool described in Chapter 2. For this purpose, this section provides information about the top-level software architecture of the AKB application and the technologies used. Furthermore, the data model is presented to show the information that is available for the recommender system. This information includes the objects that are used in the AKB tool and the relationships between them.

4.1.1 Software Architecture and Technologies

The AKB tool was built as a web application and is based on a client-server architecture. Therefore, the application consists of a client-component as well as a server component. The client-component is responsible for the user interface and parts of the business logic. It is accessible via a browser and was built on the AngularJS framework. AngularJS is a JavaScript web-framework which extends the HTML vocabulary and provides dynamic views in web-applications [39]. The server component is responsible for the business logic and data-access as well as data-manipulation in the database. The server-component was built using the Spring Framework which is a Java platform to develop Java applications [40]. The Spring Framework got extended by additional modules like Spring Boot, Spring Web MVC, and Spring Data Neo4j to ensure all functionalities for a web application and access to a Neo4j database. The AKB tool uses a Neo4j database to persist data. Neo4j is a graph database that stores data as nodes and edges [41].

4.1.2 Data Model

This section provides an overview of the data model in the AKB application. Since not all information is relevant for the recommender system, this section shows only the relevant part of the whole data model. As mentioned before, the AKB tool uses a graph database which means that data is stored as nodes and edges. Figure 4.1 shows the data model of the AKB tool.

Architecture profiles are essential objects in the AKB application. An architecture profile is a software architecture documentation using the architecture haiku approach, which was discussed in Section 2.2. An architecture profile is called $ArchProfile$ in the data model and is displayed at the top-left in the figure. As mentioned in Section 2.2, architecture profiles may contain multiple elements like attributes, constraints, and design decisions. Therefore, an architecture profile can have multiple $HAS\_ATTRIBUTE$ relationships to different architecture profile attributes which are called $APAttributes$ in the figure. An $APAttribute$ belongs to a specific category like attributes, constraints, or tactics indicating what kind of element the $APAttribute$ is. In the user interface, the elements of an architecture profile are then grouped by the category as shown in Figure 2.1.

One of the central concepts of the AKB tool is to build a common terminology among users. For this purpose, the idea of $CoreData$ was introduced. For example, the term performance might be used in multiple software architecture documentations. In this case, performance may become a $CoreData$ in the application with a uniform definition. An $APAttribute$ can $INHERIT$ from this $CoreData$ to include the attribute performance into the architecture profile with its uniform definition. Since an element in an architecture profile might affect other elements, $AttributeRelations$ were introduced. For example, if performance has a negative effect on maintainability, the application cre-

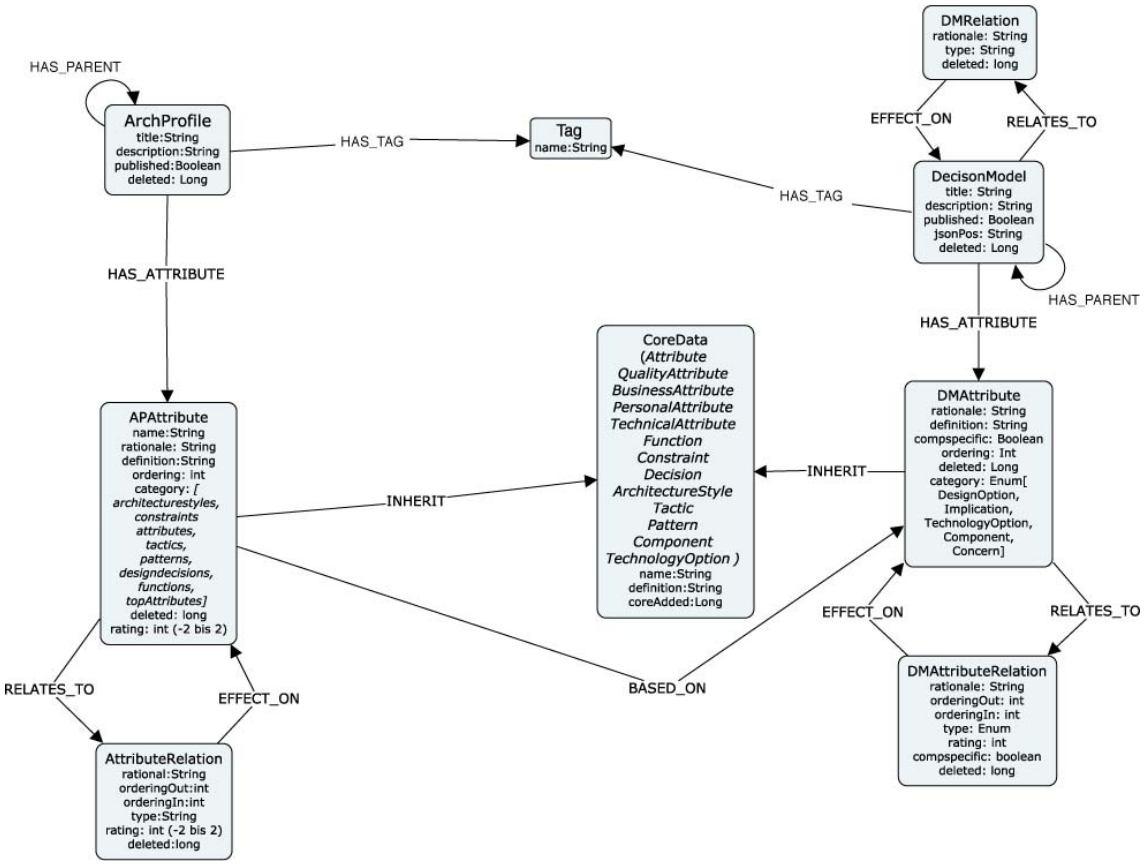


Figure 4.1: Data-Model

ates an $AttributeRelation$ for these elements. The $AttributeRelation$ includes a $rating$ that shows what kind of affect the elements have on each other. For example, performance can have a negative, positive, or neutral affect on maintainability. Additionally, a $rationale$ can be provided to this relationship.

Keeling [8] states that it is possible to create multiple architecture haikus for a software system with each describing a subcomponent of the system. To deal with this aspect, the HAS_PARENT relationship was introduced. A hierarchy of architecture profiles can be created with these relationships by declaring that one architecture profile is the parent of another architecture profile. Finally, as explained in Section 2.2, architecture profiles can have multiple properties in form of tags. For example, $Tags$ show the application context of an architecture profile. For example, if the architecture profile refers to a microservice architecture, the tag "Microservice" can be added to the architecture profile.

Besides architecture profiles, the AKB tool provides decision models which represent project-generic architecture knowledge. Decision models can be used for both decision guidance and decision documentation. Decision models are discussed in more detail in Section 2.3. The data model of decision models is similar to those of architecture profiles. A $DecisionModel$ can have multiple $HAS\_ATTRIBUTE$ relationships to

different $DMAttributes$. A $DMAttribute$ can $INHERIT$ from a $CoreData$ to support a common terminology in decision models and architecture profiles. Also, an element in a decision model might affect other elements. Therefore, $DMAttributeRelations$ show the relationship between elements in a decision model. A hierarchy of decision models can be created with the HAS_PARENT relationship between decision models. Decision models can have multiple tags which refer to properties of a decision model. These properties can be used to show the application context of a decision model, for example. As explained in Section 2.3, the AKB tool combines multiple decision models into a design process to show the order in which decision models should be used for decision making in a specific area. In the data model, a $DMRelation$ is shown in the top-right of the figure and determines the relationship between decision models regarding the design process.

As explained in Chapter 2, decision models can be used for decision documentation. In this regards, selected design options when using a decision model can be saved to an architecture profile. In this case, a new $APAttribute$ is created for the architecture profile which refers to the same $CoreData$ as the selected $DMAttribute$. Additionally, a $BASED\_ON$ relationship between the $APAttribute$ and the $DMAttribute$ of the decision model is created. This way, it becomes traceable which decision model was used for this design decision.

4.2 Requirements

Since the recommender system was developed for the AKB tool described in Chapter 2, the previous section addressed various aspects of the tool to provide context. This section, on the other hand, deals with the requirements that were identified regarding the recommender system. The first part examines general requirements of the recommender system. This part shows that the recommender system has two major application scenarios. On the one hand, the system should recommend decision models to use for decision making and/or documentation within an architecture profile. On the other hand, the system should recommend design options when using a particular decision model for decision making. The following parts of this section deal with the requirements for the recommendation of both decision models and design options.

4.2.1 General Requirements

The first question that needs to be answered when developing a recommender system is what items the system should recommend to the user. In the case of software architectures, two different items could help software architects in the decision-making process. On the one hand, the system could recommend decision models potentially supporting decision making and documentation for a particular software system. For example, if a

decision was made for the generation of monitoring data, the system could recommend that a decision for storing of monitoring data would also be required. On the other hand, the system could recommend suitable design options when using a decision model. For example, the storing of monitoring data of a microservice architecture can be designed either centralized or decentralized. In this case, the system should recommend which of these two design options is most suitable for a specific software system.

As mentioned before, the goal of this thesis is to develop a recommender system to support software architects and other stakeholders when making decisions for a software architecture. This goal is accomplished by recommending decision models to use in the decision-making and documentation process and design options when using a decision model.

The literature about recommender systems always assumes that items should be recommended to a particular user. In the case of decision making in software architecture, however, a person might be involved in multiple different software projects. Furthermore, many stakeholders could collaborate on the same software system. Therefore, the main factor influencing the recommendation is not the user and its behavior but the architecture of the system under development. This architecture is described using an architecture profile as described in Section 2.2. The items that are being recommended for a particular software architecture are decision models for particular architectural aspects of a system and the design options to be selected within a particular decision model.

As mentioned earlier, the system should make recommendations for architectural design decisions even if there is little known about the architecture of the software system. Therefore, the recommender system needs to deal with the cold start problem. The cold start problem is only addressed by a knowledge-based recommender system, which means that it has to be included to address this problem. On the other hand, the recommender system requires a learning-based technique like a collaborative-filtering or content-based method to become more precise when more architecture knowledge is available in the architecture knowledge base. These requirements show that multiple recommender methods need to be applied. Therefore, hybridization techniques as presented in Section 3.6 need to be applied to build a hybrid recommender system which is necessary to satisfy all requirements.

4.2.2 Recommendation of Decision Models

After explaining the general requirements for the developed recommender system, this section focuses on specific requirements for the recommendation of decision models. In this regards, this section presents five requirements that were identified.

Similar Architecture Profiles

The first requirement is to recommend decision models that are used for decision making and documentation in similar architecture profiles. A similar architecture profile is one that has a high number of used decision models in common with the active architecture profile. The system should recommend decision models that are used in similar architecture profiles but not yet in the active architecture profile. Therefore, the system needs to determine similar architecture profiles first and then recommend the decision models they also used for decision making and documentation. For example, if most of the similar architecture profiles use the decision model "storing of monitoring data", the system should recommend this decision model in the active architecture profile. This procedure can be expressed by "recommend decision models that were also used for decision making and documentation in similar architecture profiles".

Similar Context

Sections 2.2 and 2.3 showed, that properties can be provided to both architecture profiles and decision models. Properties are used to define the application context of architecture profiles and decision models, for example. Therefore, the system should recommend decision models that have a similar context as the active architecture profile. In other words, decision models that have similar properties to those of the active architecture profile should be recommended. For example, if the architecture profile refers to a microservice architecture and thus, has the tag Microservice, the system should recommend decision models for microservice architectures. Also, if the active architecture profile has the tags Microservice and Monitoring, decision models for monitoring in the context of microservices should be recommended. This procedure can be expressed by "recommend decision models that have a similar context as the architecture profile".

Similar Decision Models

This requirement relates to decision models already used for decision making and documentation in the active architecture profile. As explained in Section 2.2, properties of decision models are used to classify them and can refer to the application context of decision models, for example. Therefore, the properties of already used decision models should be utilized to recommend other decision models of the same classes. For example, if used decision models have the property "Monitoring", the system should recommend other decision models which also have this property. Therefore, the system should determine the similarity between properties of used decision models and the properties of unused decision models. Finally, the system should recommend the decision models with the highest similarity to the properties of used decision models. This procedure can be expressed by "recommend decision models that have similar properties as decision models

that were already used for decision making and documentation".

Related Decision Models

As explained in Section 2.3, dependencies between decision models exist. The main idea of this requirement is to recommend decision models that relate to decision models that were already used in the active architecture profile for decision making and documentation. In this case, the dependencies refer to excludes- and requires-relationships between design options of two decision models. This procedure can be expressed by "recommend decision models that have dependencies to decision models that were already used for decision making".

Design Process

The final requirement for the recommendation of decision models is to include the design processes into the recommender system. On the one hand, the system should recommend decision models that should be used next in the design process. On the other hand, decision models that were already used for experimentation or where the decision-making process is still unfinished should be recommended. This procedure can be expressed by "recommend decision models that should be used next in the design process or where the decision-making process is still unfinished".

As a result, there are five requirements for the recommendation of decision models. First, decision models that were used by similar architecture profiles should be recommended to make use of already documented architectural knowledge in other architecture profiles. Second, the properties of the active architecture profile should be taken into account when making recommendations. Therefore, the properties explicitly provided in the architecture profile and the properties of already used decision models should be considered in the recommendation process. Also, the system should recommend decision models that have dependencies to already used decision models. Finally, the design process in a specific design space should be considered when making recommendations.

4.2.3 Recommendation of Design Options

As mentioned before, the second major application scenario of the recommender system is the recommendation of design options when using a decision model for decision making. Four requirements could be identified in this regards.

Similar Architecture Profiles

The first requirement is to utilize architecture knowledge that is already documented in other architecture profiles in the AKB tool. When using a decision model, the recommender system should recommend design options that were selected by similar architecture profiles. In this case, a similar architecture profile is one that has a high number of design decisions in common with the active architecture profile. Therefore, the system should make recommendations based on the design decisions made in other architecture profiles. Additionally, the recommender system should return the rationales for decisions provided in similar architecture profiles. This procedure can be expressed by "recommend design options that were selected by similar architecture profiles when using a decision model".

Attributes in an Architecture Profile

As explained in Section 2.2, important attributes of a software system can be added to the architecture profile. For example, if maintainability is an important quality attribute of a software system, it may be added to the architecture profile. These attributes can be used to make recommendations for design options when using a decision model. In Section 2.3 it was discussed that design options can have positive, negative, or neutral influences on aspects. In this example, a design option that has a positive influence on the aspect maintainability should be more likely recommended than a design option that has a neutral or even a negative influence on maintainability. If both maintainability and performance are important attributes in an architecture profile, this procedure can be expressed by "recommend design options that influence the attributes maintainability and performance the most positive".

Design Decisions in an Architecture Profile

Another aspect would be to make recommendations based on design decisions that are already made and documented in an architecture profile. For example, if most of the made design decisions have a positive influence on scalability, design options that also have a positive influence on scalability should be more likely recommended when using a decision model. Therefore, the attributes that are influenced by selected design options should be taken into account when making recommendations. This procedure can be expressed by "recommend design options that have a similar influence on attributes than already made design decisions".

Excluded and Required Design Options

As explained in Section 2.3, dependencies between decision models exist. These dependencies express if a design option of a decision model excludes or requires a design option of another decision model. Therefore, if decision models were already used for decision making and documentation in an architecture profile, the system should analyze whether dependencies exist when using a decision model for decision making. For this purpose, the system should recommend a design option if it requires an already selected design option in the active architecture profile. On the other hand, the system should not recommend a design option if it is excluded by an already selected design option.

In total, these four requirements could be identified for the recommendation of design options. On the one hand, recommendations should be made based on the software architecture documentations of other projects. Therefore, similar architecture profiles need to be determined and the design options they selected in a specific decision model should be recommended. On the other hand, documented architecture knowledge in the architecture profile should be used to make recommendations. For this purpose, attributes that are declared as important and already made design decisions in the architecture profile are used. Finally, excludes- and requires-relationships should be used to decide which design options to recommend and which design options to not recommend. These relationships can be seen as domain knowledge in the corresponding design spaces.

4.3 Recommendation Strategies

The requirements for the recommender system were presented in the previous section. This section focuses on recommendation strategies to fulfill these requirements. First, this section deals with recommendation strategies for the recommendation of decision models and afterward with those for the recommendation of design options. The procedure to develop these recommendation strategies is divided into four steps. First, the knowledge sources to make recommendations need to be identified. Therefore, for each requirement, it needs to be determined what kind of data is used to make recommendations. The kind of data could be a rating, the content of items or some domain knowledge, for example. As explained in Section 3.2 the knowledge source is the main criterion to assign a recommender system to a category. Therefore, once the kind of data is identified, the recommender system category can be determined which is the second step in the process. In the third step, the recommender methods in the respective recommender system category get evaluated to select the best recommender method for each requirement. After the first three steps are done for each requirement, the final step deals with hybridization techniques to combine the results of all recommender methods.

4.3.1 Recommendation of Decision Models

Section 4.2.2 presented five requirements for the recommendation of decision models. Strategies for recommending decision models potentially supporting the decision-making process are examined in this section using the process described before. The following recommendation strategies aim to fulfill the requirements for the recommendation of decision models. It needs to be highlighted that the items that should be recommended are decision models.

Similar Architecture Profiles

The first requirement was the recommendation of decision models that were also used for decision making and documentation in similar architecture profiles. In this case, a similar architecture profile is one that has a high number of used decision models in common with the active architecture profile. Since recommendations are made based on other architecture profiles, it can be anticipated that the recommender system category is collaborative-filtering as it is the only category that utilizes items used by others. Content-based methods would only take into account the decision models already used in the same architecture profile to make recommendations. Knowledge-based recommender systems use explicitly provided requirements or preferences to make recommendations, which is not the case for this requirement. Therefore, the requirement can be fulfilled by either an user-based, item-based, or model-based collaborative-filtering recommender system.

First, the knowledge source needs to be determined. As explained in Section 3.3, a pure collaborative-filtering recommender system uses only ratings to make recommendations. There is no information used about items. Therefore, it needs to be defined what a rating is in this case. Whether an architecture profile is similar to the active architecture profile is determined by the used decision models they have in common. An implicit rating can be derived from this where decision models that were already used for decision making have a rating of 1 and unused decision models have a rating of 0. This way, a rating-matrix can be created as shown in Table 4.1.

The table shows architecture profiles on the y-axis and all decision models in the AKB tool on the x-axis. The rating is shown in the cells which is either 1 or 0 depending on whether the decision model was already used for decision making or not. The first and second decision model were already used for decision making in the active architecture profile. Decision Model 3 and 4 were not used for decision making yet which means that for these decision models it needs to be determined whether they should be recommended or not. Therefore, the decision models that were already used for decision making are utilized for finding similar architecture profiles. For all unused decision models, the recommender system should determine whether they should be recommended or not based on what

Profile/Decision Model	DM 1	DM 2	DM 3	DM4
Active Profile	1	1	**?**	**?**
Profile 2	1	1	0	0
Profile 3	1	0	1	0
Profile 4	1	1	0	1

Table 4.1: Decision Model Rating Matrix

similar architecture profiles used.

As explained in Section 3.3, the item-based collaborative-filtering method was developed to deal with large databases. In this regards, a large database is one that contains millions of records. The model-based method is usually less accurate than user-based and item-based methods but has a better performance when it comes to large databases. Since the AKB tool will not have a database of this size soon, the user-based method will be most appropriate for this requirement. Therefore, the techniques described in Section 3.3.1 should be used to make recommendations. The Person coefficient is used to determine the similarity between architecture profiles. As discussed in Section 3.3.1, either a defined threshold for the similarity value or a fixed number of neighbors can be used to select the most similar architecture profiles. Since a threshold might result in an empty list of similar architecture profiles and due to the difficulty of defining an appropriate threshold, a fixed number of similar architecture profiles should be used. The presented formula in Section 3.3.1 to calculate a prediction value for unrated items is used to compute a value for each decision model. The result of this recommender method is a list of all unused decision models with a value between 0 and 1 indicating how likely it should be used for decision making based on similar architecture profiles.

Similar Context

As explained in Section 2.2, the context of an architecture profile can be defined by providing properties to the architecture profile in form of tags. In this regards, the system should recommend decision models that have a similar context as the active architecture profile. For example, if an architecture profile has the tags Microservice and Monitoring, decisions models for monitoring in microservice architectures should be recommended. The system does not make recommendations based on other architecture profiles which makes collaborative-filtering recommender methods not suitable for this requirement. Also, recommendations are not based on already used decision models which excludes content-based methods as well. Therefore, this requirement can be fulfilled by using a knowledge-based recommender method.

The case-based and constraint-based method are the two standard methods for knowledge-based recommender systems. The case-based method uses similarity measures to compare preferences or requirements with the content of items. The constraint-based method makes use of rules to filter the item space based on the provided preferences or requirements and recommends the remaining items to the user. The case-based method will be more beneficial in this case since the result of the constraint-based method could be an empty list of recommendations and developing and maintaining domain-specific rules in the AKB tool will be a challenging task. In addition, it might be more interesting to see how well the context of a decision model fits to the context of an architecture profile. For this purpose, the recommender system should calculate the similarity between the properties of an architecture profile and the properties of decision models. Consequently, the preferences or requirements in a case-based approach refer to the list of properties in an architecture profile and the content of items refers to the properties of decision models.

In order to apply the similarity measure presented in Section 3.5.2, it needs to be determined if a property in the architecture profile is also present in a decision model. If this is the case, the decision model gets a value of 1 for this property. If the decision model misses a property of the architecture profile, it gets a value of 0 for this property. Table 4.2 demonstrates an example for an architecture profile that has the tags Microservice and Monitoring.

	Requirements	**DM 1**	**DM 2**	**DM 3**
Microservice	1	1	1	0
Monitoring	1	1	0	0

Table 4.2: Case-based Recommender Data for Decision Models

The table shows that decision model 1 has both the Microservice and Monitoring tag. Decision model 2 also has the Microservice tag. However, it misses the Monitoring tag. Decision model 3 does not have any of these tags. The result of this recommender method is a list of all unused decision models with a similarity value indicating how similar its properties are to the properties of the architecture profile.

Similar Decision Models

Regarding this requirement, the recommender system should take into account the properties of decision models already used for decision making. Since properties are used for classification, other decision models belonging to the same classes as used decision models should be recommended. For example, if used decision models have the

property "Monitoring", other decision models with the property "Monitoring" should be recommended. A decision model can have one or more properties which can be seen as the content of decision models. Consequently, the content of items is used to make recommendations which refers to the knowledge source of this recommender method.

Since collaborative-filtering recommender methods only use ratings and not the content of items to make recommendations, they are not applicable for this requirement. Furthermore, the system makes recommendations based on decision models used in the active architecture profile and not in other architecture profiles. Moreover, since items used in the past are used to make recommendations and not any requirements or preferences, a content-based method is most suitable for this requirement.

A content-based recommender system can be implemented by either using rated items as training-data for machine-learning techniques or by creating a user profile that contains preferences which are compared to the content of unrated items. In this case, a user profile that composes properties of all decision models already used for decision making will be more beneficial than machine-learning techniques. The reason for this is that the properties in the user profile can be extended by the properties explicitly provided in the architecture profile. Therefore, both the properties of used decision models and the explicitly provided properties in an architecture profile are used to find similar decision models.

The Dice coefficient as proposed by Jannach et al. [20] is used for computing the similarity between the user profile and unused decision models. Therefore, the system calculates how much the user profile overlaps with the properties of unused decision models. Table 4.3 shows an example of the data used for this recommender method.

	Tags
User Profile	Microservice, Monitoring, Service Discovery
DM1	Microservice, Monitoring
DM2	Microserivce, Fault Tolerance
DM3	Service Discovery

Table 4.3: Content-based Recommender Data for Decision Models

The table demonstrates that the user profile consists of three properties which are Microservice, Monitoring, and Service Discovery. Decision model 1 has two of these properties, decision model 2 only one and an additional property that does not exist in the user profile, and decision model 3 also has one of these properties. Therefore, decision model 1 will be more likely recommended than the other decision models. The result of this recommender method is a list of all unused decision models with a value indicating

how similar the properties of a decision model are to properties of used decision models and properties of the architecture profile.

Related Decision Models

The recommender system should also recommend decision models that have dependencies to decision models already used for decision making in an architecture profile. Since recommendations are not based on other architecture profiles, collaborative-filtering methods are not suitable for this requirement. Also, recommendations are not made based on the content of already rated items which excludes content-based methods as well. Also, neither the case-based nor the constraint-based method of knowledge-based recommender systems is suitable in this case because the content of items is not used to make recommendations. Therefore, a domain-specific recommender method needs to be implemented. Since domain knowledge is used to make recommendations, this recommender method can be categorized into knowledge-based recommender methods. The result of this recommender method is a list of all unused decision models with a value of 1 if a dependency to an already used decision model exists and a value of 0 if no dependency exists.

Design Process

The final requirement for the recommendation of decision models is to take the design process into account. Thereby, decision models that should be used next in the design process and decision models already used for experimentation should be recommended. Again, the recommendations are not based on other architecture profiles which makes collaborative-filtering methods inappropriate for this requirement. Since the content of decision models is also not considered, content-based methods, as well as the case-based and constraint-based methods, are not applicable in this case. The system utilizes domain-specific knowledge to make recommendations, similar to the previous recommender method. Therefore, this recommender method can be categorized as a knowledge-based recommender method. For this requirement, the system should return a list of all unused decision models with a value of 1 if it should be used next in a design process or if it was already used for experimentation and a value of 0 if none of the above is true.

Hybridization Techniques

Multiple different recommender methods need to be applied to fulfill the above mentioned requirements. The results of these recommender methods are combined using hybridization techniques as presented in Section 3.6. Table 4.4 provides an overview of

the requirements and the corresponding recommender methods.

Requirement	Recommender Method
Similar Architecture Profiles	User-based Collaborative-Filtering
Similar Context	Case-based
Similar Decision Models	Content-based
Related Decision Models	Knowledge-based
Design Process	Knowledge-based

Table 4.4: Decision Model Recommender Methods

In order to deal with the cold-start problem, the case-based method can be used as soon as the architecture profile is created. Thereby, if at least one property is provided to show the context of the architecture profile, this property can be used to recommend decision models in the same context. Additionally, as soon as the first decision model was used for decision making and documentation, recommendations can be made based on both related decision models and the design process. In other words, the knowledge-based methods can be applied for making recommendations even if there is little known about the software architecture. The user-based collaborative-filtering method and the content-based method are learning-based techniques and will be applied at a later stage of the decision-making process. Since the case-based and the content-based recommender method deliver decision models based on similar properties, the switching technique can be used for these recommender methods. The case-based method is used as long as a certain amount of decision models was used for decision making and documentation. Afterward, the content-based method will be used which should deliver better results at this point of the decision-making process. The other recommender methods produce results with a different meaning. All of these results might be of interest to software architects. Therefore, these results should be presented side by side using the mixed hybridization technique. The user-based collaborative-filtering method will only be used if a certain amount of decision models was already used for decision making and documentation and if a certain amount of architecture profiles exists in the AKB application. These numbers are configurations to the recommender system and should be able to be changed at any time. Figure 4.2 shows an overview of the hybrid recommender system. The figure includes the requirements, the data that is used, the recommender methods that are applied, and the hybridization techniques used to combined the different results.

However, using only the hybridization techniques mentioned above, software architects will receive up to four different recommendation results for each decision model. It might bring much effort to evaluate these different recommendation results. For software archi-

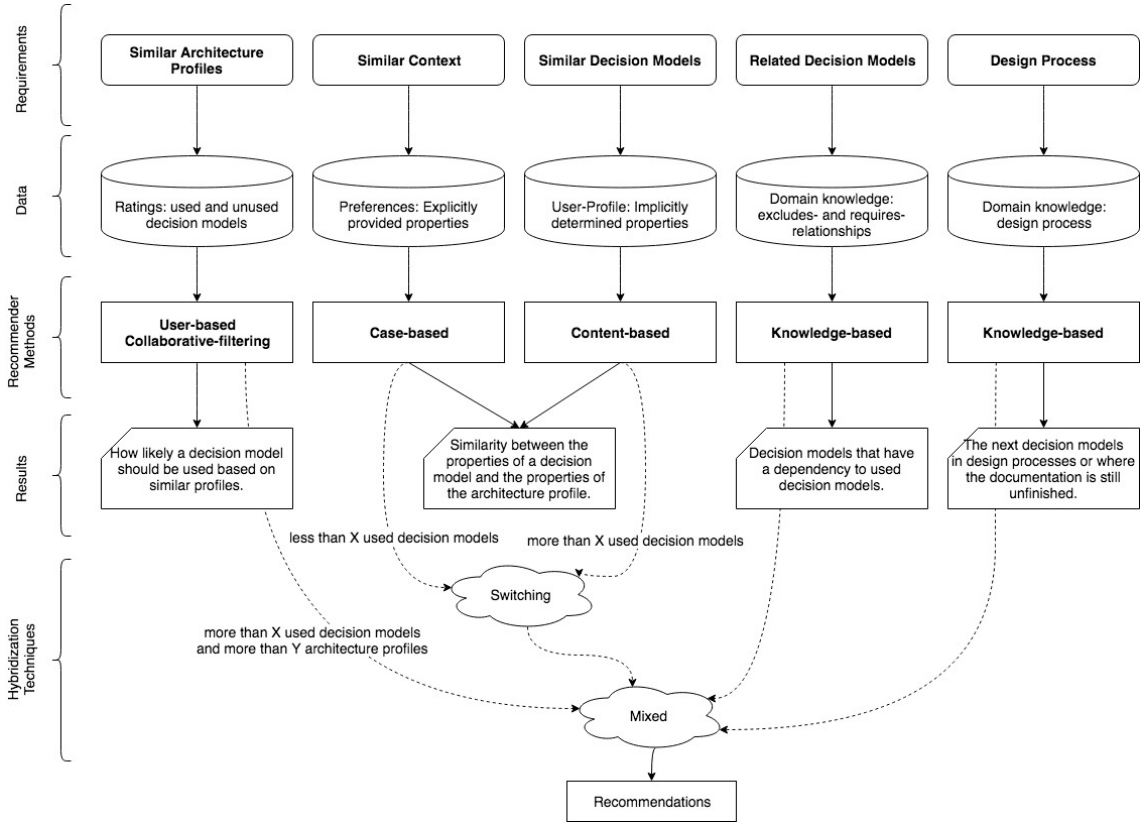


Figure 4.2: Hybrid Recommender System for Decision Models

tects, it might be interesting to see an overall recommendation-score for each decision model to receive an immediate insight into the results. To address this issue, a weighted overall score can be calculated for each decision model by applying the weighted hybridization technique. Thereby, a weight can be defined for each recommender method to add its importance compared to other recommender methods into the calculation. However, the results of the recommender methods have a different meaning. The user-based collaborative-filtering method returns a value between 0 and 1 indicating how likely a decision model should be used based on similar architecture profiles. Both the case-based and content-based method return a value between 0 and 1 indicating how similar the properties of a decision model is to the preferred properties. A min-max normalization of the results can address this issue. Thereby, regardless of the actual result, the decision model with the best result in a recommender method becomes a value of 1 for the calculation of the overall score. Formula 4.1 shows the equation to calculate the min-max normalized value.

$$minmax = \frac{result - min}{max - min} \tag{4.1}$$

The $result$ is the actual result of the recommender method. max is the value of the

decision model with the best result in the recommender method. In this case, min is always 0 and not the actual minimum value of the recommender results. The reason for this is if two decision models have a result of 0.9 and 0.85 in a recommender method, the decision model with a result of 0.9 will have a min-max normalized value of 1 and the decision model with the result of 0.85 would have a min-max normalized value of 0 which does not represent the actual result. Table 4.5 shows an example were the results of the user-based collaborative-filtering and the content-based method get min-max normalized.

Recommender Method	**DM1**	**DM2**	**DM1 Min-Max**	**DM2 Min-Max**
User-based CF	0.8	0.6	1	0.75
Content-based	0.25	0.3	0.83	1

Table 4.5: Min-Max Normalization of Recommender Results

The table shows that the user-based collaborative-filtering method concludes a value of 0.8 for decision model 1 and a value of 0.6 for decision model 2. Using the min-max normalization, the results are transformed to a value of 1 for decision model 1 and a value of 0.75 for decision model 2. The weighted overall score is then calculated using the weighted hybridization formula presented in Section 3.6. The weight of each recommender method should be configurable to be able to change them at any time. For example, if the user-based collaborative-filtering method is more important than the content-based method, it may get a weight of $\frac{2}{3}$ and the content-based method a weight of $\frac{1}{3}$. The calculation for the weighted overall score for decision model 1 is shown in Formula 4.2.

$$rec_{weighted}(u, DM1) = \sum_{k=1}^{n} w_k * rec_k(u, DM1) = \frac{2}{3} * 1 + \frac{1}{3} * 0.83 = 0.943 \qquad (4.2)$$

Formula 4.3 shows the calculation for the weighted overall score for decision model 2.

$$rec_{weighted}(u, DM1) = \sum_{k=1}^{n} w_k * rec_k(u, DM1) = \frac{2}{3} * 0.75 + \frac{1}{3} * 1 = 0.833 \qquad (4.3)$$

The results show that decision model 1 has a higher overall score because it performed better in the user-based collaborative-filtering method which has a higher weight compared to the content-based method. To sum up the hybrid recommender system for decision models, the system returns a weighted overall score for each decision model

based on the results of the different recommender methods. Additionally, the results of each recommender method get returned to get a more detailed insight. Since both the case-based and content-based method return the same result, either the case-based or the content-based method is used based on how many decision models were already used for decision making in the active architecture profile.

4.3.2 Recommendation of Design Options

Section 4.2.3 presented four requirements for the recommendation of design options. The process described in the introduction of this section will be examined for these requirements. However, the general requirements for the recommender systems need to be borne in mind which were presented in Section 4.2.1. In general, it must be highlighted that the items that need to be recommended are design options in this case.

Similar Architecture Profiles

The first requirement was the recommendation of design options that were selected by similar architecture profiles when using a decision model. A similar architecture profile refers to one that has a high number of design decisions in common with the active architecture profile. Since recommendations are made based on other architecture profiles, the recommender system category is collaborative-filtering. Both content-based and knowledge-based methods do not use any information about the behaviour of others to make recommendations. Therefore, they are not applicable for this requirement. Both the item-based and model-based collaborative-filtering method were developed for large databases that contain millions of records. However, the AKB tool will not have a database of this size in the near future. Therefore, the user-bases collaborative-filtering method is used to fulfill this requirement.

As mentioned before, collaborative-filtering methods only make recommendations based on ratings. Similar to the user-based collaborative-filtering method for decision models, it needs to be determined what a rating is in this case. Whether an architecture profile is similar to the active architecture profile should be determined by the design decisions they have in common. Another way to see this is to look at the design options they have selected and the design options they did not select in the past. For example, in a microservice architecture, monitoring data can be stored centralized or decentralized. If centralized storing of monitoring data was selected and decentralized was not selected in two architecture profiles, they have a design decision in common. An implicit rating can be concluded from this where selected design options have a rating of 1 and not selected design options have a rating of 0. Based on this information, a rating-matrix can be created as Shown in Table 4.6.

Profile/Design Option	DO 1	DO 2	DO 3	DO4	DO 5	DO 6
Active Profile	1	0	1	1	**?**	**?**
Profile 2	1	1	0	0	0	1
Profile 3	1	0	1	0	0	0
Profile 4	1	1	0	1	0	0

Table 4.6: Design Option Rating Matrix

The table shows architecture profiles on the y-axis and design options on the x-axis. The rating is shown in the cells which is either 1 or 0 depending on whether the design option was selected or not. To consider all design spaces defined in the AKB tool, design options of all decision models should be used to create the rating matrix. Additionally, design decisions that are explicitly provided in an architecture profile and that are not covered in a decision model should be considered. For example, if a design decision needs to be documented which is not available in a decision model, it can be added explicitly in the architecture profile. Other architecture profiles which also have this design decision documented may also have a rating of 1 for this design decision. Therefore, the similarity between architecture profiles is calculated based on design decisions in all available design spaces. For example, in Table 4.6, the first three design options may belong to the same decision model. Hence, the first and third design option in this decision model was selected in the active architecture profile. Design Option 4 may be an explicitly defined design decision which is documented in the active architecture profile and in the fourth architecture profile. Design Option 5 and 6 belong to another decision model which was not used for documentation in the active architecture profile, yet. For these design options, the system should make recommendations based on similar architecture profiles.

Similar to the user-based collaborative-filtering method for the recommendation of decision models, the Pearson coefficient is used to determine the similarity between the active architecture profile and other architecture profiles. Also, a fixed number of similar architecture profiles will be used instead of a defined threshold for neighborhood selection. Otherwise, the result might be an empty list of similar architecture profiles and the definition of a proper threshold can be difficult in this context. For the calculation of the prediction value, the same formula as presented in Section 3.3.1 is used. The result of this recommender method is a list of all design options in the respective decision model with a value between 0 and 1 for each design option indicating how likely it should be selected based on the decisions in similar architecture profiles. Additionally, the rationale for the decision of a design option in similar architecture profiles might be of interest. Therefore, a list of all rationales for the decision of a design option will be returned by the recommender system.

Attributes of an Architecture Profile

Attributes that are declared as important in an architecture profile should be used to make recommendations when using a decision model. These attributes can be seen as preferred attributes the respective software system should have. Therefore, the knowledge sources are preferences or requirements of the software system. Since recommendations should not be made based on the decisions in other architecture profiles, collaborative-filtering methods will not be suitable for this requirement. Furthermore, recommendations should be made based on preferences or requirements and not on implicit or explicit ratings. Regarding content-based methods, design decisions made in the past would be used to make recommendations which is not the case for this requirement. Therefore, a knowledge-based approach should be applied since these recommender methods use explicitly provided preferences or requirements to make recommendations.

As mentioned earlier, a knowledge-based recommender system is typically implemented using either the case-based or constrained-based recommender method. The case-based method will be more beneficial in this case as it returns a value indicating how well a design option fits the preferred or required attributes in the architecture profile. The constraint-based method would only return the design options that fulfill the defined rules which can result in an empty list of recommendations as discussed in Section 3.5.1. Furthermore, defining reasonable rules and maintaining them might be a difficult task in the AKB tool.

The case-based method calculates the similarity between the preferences or requirements and the content of items. The content of a design option can refer to the attributes that are influenced by the design option. For example, if a design option has a positive influence on performance, a negative influence on maintainability, and no influence on scalability, the content of this design option would be: performance: 1, maintainability: -1, and scalability: 0. On the other hand, all attributes declared in the architecture profile can be seen as preferences or requirements and, consequently, have a value of 1. Therefore, the more positive influences a design option has on important attributes in an architecture profile, the more likely it will be recommended. As mentioned before, architecture profiles can be organized in a hierarchy with each describing a subcomponent of the software system. Since important attributes might be already defined in a parent-architecture profile, it might not be repeated in a sub-architecture profile. Consequently, attributes of parent-profiles also need to be considered in this recommender method. Table 4.7 shows an example where performance, maintainability, and scalability were defined as important attributes in an architecture profile or parent-architecture profiles.

The table shows that design option 1 has a positive influence on performance, a negative influence on maintainability, and a neutral or no influence on scalability. The other two design options have different influences on these attributes. Using the case-based method, the similarity between the requirements and each design option should be calculated. For this purpose, similarity measures as presented in Section 3.5.2 should be

	Requirements	DO 1	DO 2	DO 3
Performance	1	1	1	0
Maintainability	1	-1	1	0
Scalability	1	0	-1	1

Table 4.7: Case-based Recommender Data

used. The result of this recommender method is a list of all design options in the decision model and their similarity value to the preferences or requirements defined in the architecture profile.

Design Decisions of an Architecture Profile

The recommender system should also take into account the design decisions that are already documented in the architecture profile. For example, if most of the already selected design options have a positive influence on performance, design options that also have a positive influence on performance might be of interest when using a decision model. Therefore, similar to the previous recommender method, recommendations should be made based on the influences a design option has on attributes.

Since recommendations should be made based on the design decisions made in the same architecture profile and not in other architecture profiles, collaborative-filtering methods are not suitable for this requirement. A knowledge-based recommender will not be suitable as well since the recommendations should be made based on design decisions made in the past. Therefore, a content-based recommender should be applied which utilizes both the content of items.

As mentioned before, either a user profile containing preferences or machine-learning techniques can be applied regarding content-based recommender systems. In this case, a user profile containing preferred attributes will be most suitable for this requirement as explicitly provided attributes in the architecture profile can also be considered in this method. Therefore, implicitly determined attributes of already made design decisions and explicitly provided attributes in the architecture profile are used to create a user profile. As explained in Section 3.4, similarity measures are used to calculate the similarity between the user profile and the content of items. Jannach et al. [20] propose to use the Dice coefficient for this purpose. However, the Dice coefficient only expresses how much the user profile and the content of an item overlap. In this case, the content of design options has an additional dimension as it also considers the influence a design option has on an attribute which can be either positive, negative, or neutral. The Dice coefficient would only consider if a design option has an influence on an attribute. Therefore, a different similarity measure needs to be applied. Due to the similarity between the data used in

the case-based method for the previous requirement and the data for this recommender method, the same similarity measure as for the case-based method will be used. This similarity measure guarantees the consideration of the influence a design option has on the attribute. An example of the data for the content-based method is shown in Table 4.8.

	User Profile	**DO 1**	**DO 2**	**DO 3**
Performance	1	1	1	0
Maintainability	1	-1	1	0
Scalability	1	0	-1	1
Usability	1	1	0	-1

Table 4.8: Content-based Recommender Data

Compared to the recommender method of the previous requirement, usability is an attribute that got determined implicitly by already made design decisions and, therefore, is added to the list of preferences. However, for this purpose, it needs to be determined which attributes were important during the decision-making process. Section 2.3.2 showed that in the documentation process aspects can be rated to show their importance for the decision. This rating is used to determine if an aspect should be added to the user profile or not. Thereby, aspects that got rated with four or five stars during the documentation are added to the preferences. For example, if usability got rated with five stars during the documentation of a decision, it will be added to the list of preferences. If an aspect got rated with one to three stars, it is not seen as an important attribute for this decision. The difference to the case-based recommender method is that attributes that had a high impact on already made design decisions are also considered in this recommender method. The result of this recommender method is a list of all design options in a decision model and their similarity to the list of preferences.

Excluded and Required Design Options

As mentioned earlier, dependencies between decision models exist. These dependencies refer to excludes- or requires-relationships between design options of different decision models as explained in Section 2.3. Similar to the recommendation of related decision models presented in the previous section, this recommender method does not belong to a standard recommender method. Neither are recommendations based on other architecture profiles nor is the content of items used to make recommendations. Therefore, collaborative-filtering, content-based, and the standard knowledge-based recommender methods are not applicable. However, domain-specific knowledge is used to make recom-

mendations whereby this recommender method can be categorized into knowledge-based recommender methods.

In this recommender method, for each design option it needs to be examined if a relationship exists to an already selected design option in the active architecture profile when using a decision model. If a requires-relationship exists, the design option is recommended. If an excludes-relationship exists, the design option is not recommended. Therefore, the result of this recommender method is a list of design options in a decision model that require or exclude selected design options in the active architecture profile.

Hybridization Techniques

Multiple different recommender methods have been identified in the previous parts of this section. Each recommender method produces a different result of some sort. These different results need to be combined to create overall recommendations for design options when using a decision model. The combination of the results can be made by using hybridization techniques presented in Section 3.6. Table 4.9 provides an overview of the requirements and the corresponding recommender methods.

Requirement	**Recommender Method**
Similar Architecture Profiles	User-based Collaborative-Filtering
Attributes of an Architecture Profile	Case-based
Design Decision of an Architecture Profile	Content-based
Excluded and Required Design Options	Knowledge-based

Table 4.9: Design Option Recommender Methods

One of the general requirements is that the recommender system must deal with the cold-start problem. Recommendations should be made even if there is little known about the software architecture. To address this aspect, the case-based method will be used for making recommendations as soon as the architecture profile is created. Consequently, as soon as the first attribute is added to the architecture profile, the case-based method is applied which will recommend design options that have a positive influence on this attribute. Also, the other two knowledge-based methods to deal with required and excluded design options can be applied as soon as the first design decision is added to the architecture profile. The user-based collaborative-filtering and the content-based method are learning-based techniques, which means that they become more precise the more architecture knowledge is documented. Therefore, these methods will be applied once enough architecture knowledge is documented. As for the user-based collaborative-

filtering method, a certain amount of design decisions need to be documented in order to get a meaningful similarity to other architecture profiles. Also, a certain amount of architecture profiles is necessary to guarantee the identification of similar profiles. A certain amount of design decisions is also required for the content-based method to determine a trend of important aspects implicitly. In the AKB application, these numbers should be configurable.

Since both the case-based and content-based method use important attributes to make recommendations with one using only explicitly provided attributes and the other one using implicitly determined attributes as well, the switching technique can be applied. Thereby, the case-based method is used as long as the specified number of required design decisions is not reached. Afterward, the content-based method is used. All other recommender methods deliver different results. All of these results might be of interest to software architects. Therefore, the results of the user-based collaborative-filtering method, either the case-based or content-based method, and the knowledge-based recommender method will be presented side by side using the mixed technique. However, it needs to be highlighted that the user-based collaborative-filtering method will only be applied if a certain number of design decisions is already documented in the active architecture profile and if a certain amount of architecture profiles exist in the AKB tool. Figure 4.3 shows an overview of the resulting hybrid recommender system including the requirements, the data that is used, the recommender methods that are applied, and how the results are combined using hybridization techniques.

Similar to the recommendation of decision models, a weighted overall score should be calculated for each design option to provide immediate insight into the results of the recommender system. For this purpose, the system should use the weighted hybridization technique. At his point, it is important to highlight that the excludes- and requires-relationships have a significant impact on the recommendation. Hence, if a design option excludes an already selected design option, it gets an overall score of 0, and if a design option requires an already selected design option, it gets an overall score of 1. If neither an excludes- nor a requires-relationship exists, the weighted overall score is calculated as usual. Since the results of the recommender methods have a different meaning, the results need to be min-max normalized as discussed in the previous section about the hybridization techniques of the decision model recommender system. Afterward, the weighted overall score is calculated using the min-max normalized results of each recommender method. The weight of each recommender method to show its importance compared to other recommender methods should be configurable. The hybrid recommender system for the recommendation of design options returns a weighted overall score for each design option based on the results of the different recommender methods. Additionally, the results of each recommender method get returned to get a more detailed insight. Since both the case-based and content-based method return the same result, either the case-based or the content-based method is used based on how many design decisions are already documented in the active architecture profile.

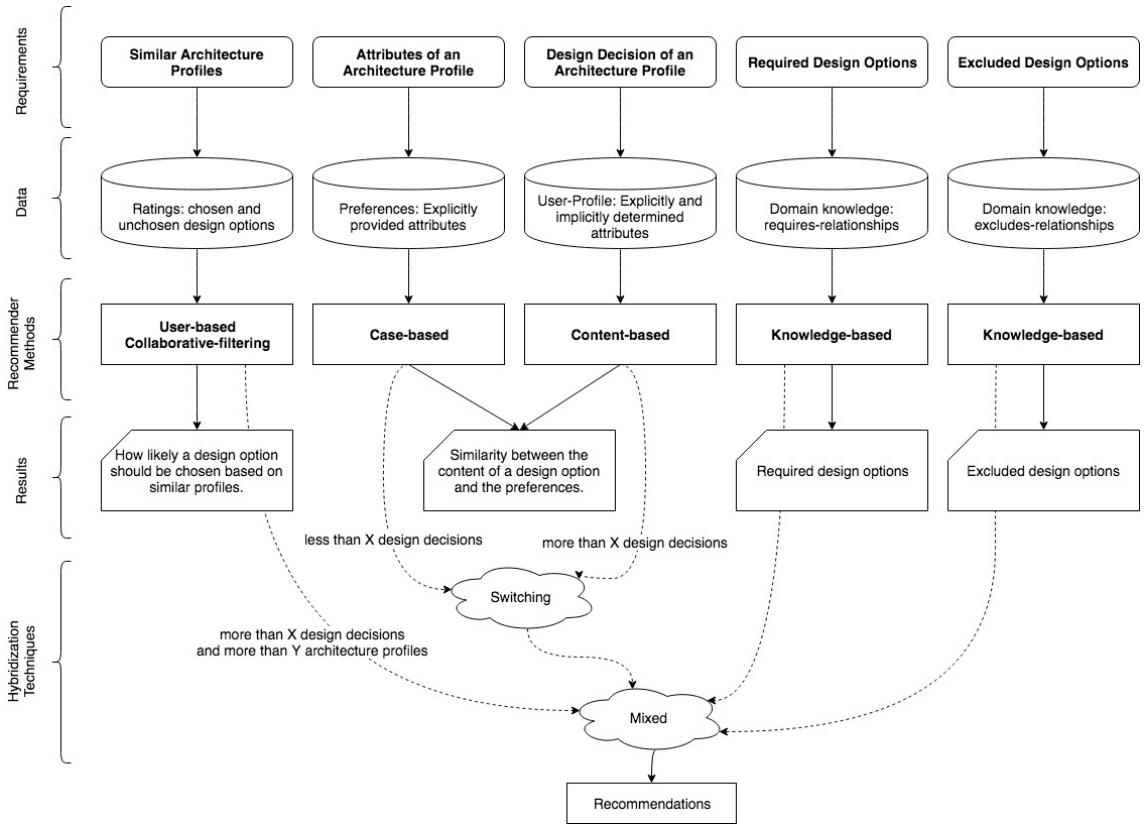


Figure 4.3: Hybrid Recommender System for Design Options

Chapter 5

User Interface and Imlementation

The previous chapter presented the requirements and concepts of the recommender system that was developed for the AKB tool. This chapter, on the other hand, provides aspects regarding the realization of the developed recommender system. First, this chapter demonstrates how the recommendations for both decision models and design options are presented in the user interface. Second, this chapter deals with software architecture and implementation aspects of the developed system.

5.1 User Interface

This section focuses on the integration of the results of the recommender system into the user interface of the AKB tool. Thereby, the chapter demonstrates what information is available and how the information is presented to the user. On the one hand, the calculated recommendations for decision models need to be displayed in an architecture profile. On the other hand, the calculated recommendations for design options need to be integrated into the user interface of decision models when an architecture profile is selected.

5.1.1 Recommendation of Decision Models

The hybrid recommender system for the recommendation of decision models in an architecture profile delivers multiple different outputs. On the one hand, the system calculates the weighted overall score for each decision model. On the other hand, the results of multiple recommender methods are returned using the mixed hybridization technique. The results of the recommendation of decision models are integrated into the active architecture profile. For this purpose, the system displays a list of all currently not used

decision models on the right side of an architecture profile. This list is displayed when the owner of the architecture profile or a user with permissions to edit the architecture profile enables the edit-mode of the profile. The weighted overall score provides a consolidated outcome of all recommender methods and should be the first result that is presented to the user. For this reason, the weighted overall score is displayed next to each decision model in the list. The decision models are ranked by the weighted overall score as shown in Figure 5.1.

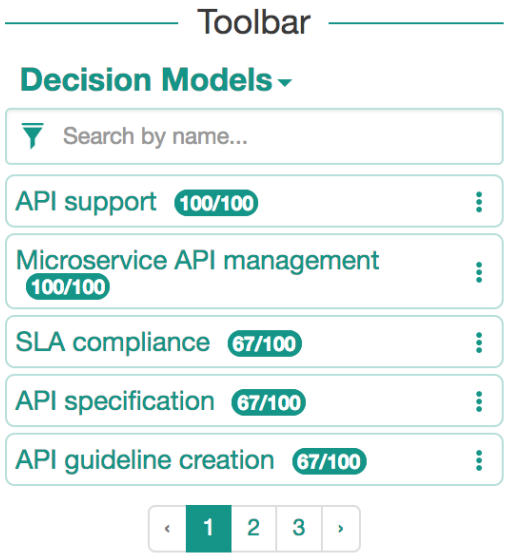


Figure 5.1: Decision Model Recommendations

The weighted overall score for each recommended decision model has a value between 0 and 100. The decision models are ranked by that score and the user can search for a specific decision model to see how it performed in the recommender system. The results of the individual recommender methods are not displayed at first. However, they are displayed in a popover as soon as the user clicks on the weighted overall score of a decision model. This way, the user gets a more detailed view of the results of the recommender system. An example of a popover is presented in Figure 5.2.

In the example shown in the figure, the decision model API support has a weighted overall score of 100 since it is recommended based on similar architecture profiles and the design process. As shown in the figure, the decision model is used by similar architecture profiles and it had already been opened and viewed. For each recommender method, the popover shows a different description of the results. The descriptions of the recommender method results read as follows:

Similar Architecture Profiles As shown in Figure 5.2, if a decision model is used

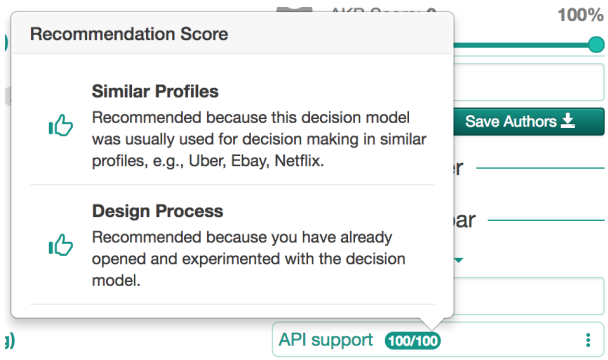


Figure 5.2: Decision Model Recommendation Popover

by similar architecture profiles, following information is shown in the popover: "Recommended because this decision model was usually used for decision making in similar architecture profiles, e.g., ..." with a list of the most similar architecture profiles.

Similar Context The case-based recommender method compares the properties of an architecture profile with the properties of decision models. If there are any matches between properties, the popover will display them as follows: "This decision model shares following properties with this profile: ..." with a list of the matching properties.

Similar Decision Models The content-based recommender method uses the properties of an architecture profile and the properties of used decision models for making recommendations. If there are any matches between the properties of the already used decision models and the properties of any other decision model, they are displayed as follows in the popover: "This decision model shares following properties with this profile or with already used decision models in this profile: ..." with a list of the matching properties.

Related Decision Models This knowledge-based recommender method analyses if there are any dependencies between already used decision models and the current decision model. If there exist excludes- or requires-relationships between decision models, the popover includes following information: "This decision model relates to the *decisionmodelname* decision model already used for decision making and documentation in this profile.". *decisionmodelname* will be replaced with the actual name of the decision model.

Design Process Finally, the popover should display information if the decision model is recommended based on the design process. For this purpose, the popover shows following information if the decision model should be used based on the design process: "Recommended because you have already documented decisions with decision models in the same design process.". Also, if the decision model was already opened and used for experimentation, the popover will show following

information: "Recommended because you have already opened and experimented with the decision model.".

The results of the recommender system of decision models are integrated into the user interface of architecture profiles. The recommendations are only shown if the owner or a user with permissions to edit the architecture profile enables the edit-mode of the profile. At first, a list of all unused decision models with their weighted overall scores is presented to the user to provide an immediate insight into the results. If the software architect requires more information, the popover shows more details of the results of the recommender methods.

5.1.2 Recommendation of Design Options

Similar to the recommendation of decision models, the hybrid recommender system for the recommendation of design options delivers multiple different outputs. For each design option, the system returns the weighted overall score and the results of each recommender method. As for the recommendations based on similar architecture profiles, the rationales for the decisions in other architecture profiles get returned as well. These results are integrated into the user interface of decision models when an architecture profile is selected for which a decision should be made. The weighted overall score is the first result the user should see and is displayed at the bottom right corner of a design option. An example is shown in Figure 5.3.

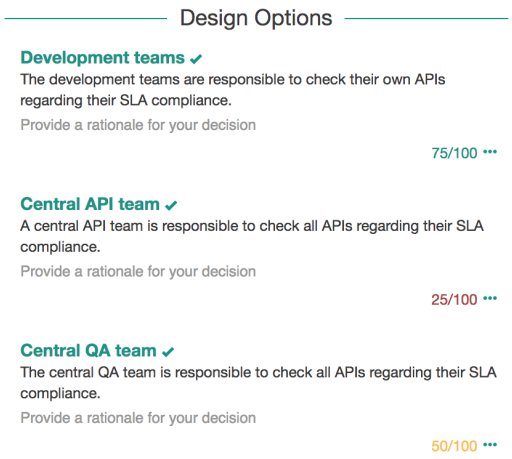


Figure 5.3: Design Option Recommendations

Similar to the weighted overall score of decision models, the score of each design option has a value between 0 and 100. The score has a green color if it is 70 or higher, red

if it is 30 or less, and yellow otherwise. Therefore, the color is an additional indication of how well a design option performed in the recommender system. The results of the individual recommender methods are not displayed at first. However, they are displayed in a popover as soon as the user clicks on the weighted overall score of a design option. This way, the user gets more detailed information about the recommendation. An example of a popover is presented in Figure 5.4.

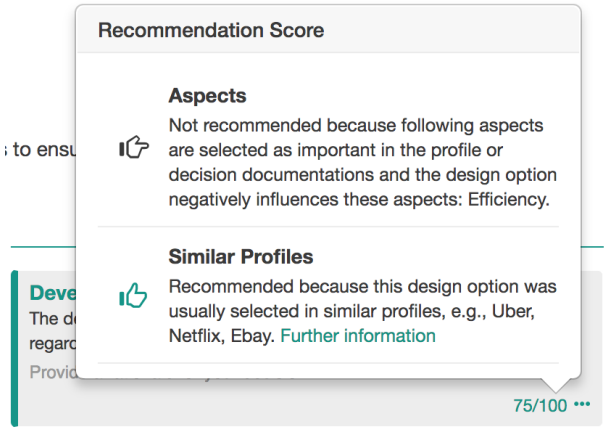


Figure 5.4: Design Option Recommendation Popover

The figure shows the outcomes of both the content-based method and the user-based collaborative-filtering method. In this example, the design option is not recommended based on the content-based method as it has a negative influence on efficiency and efficiency is an important attribute in the architecture profile. On the other hand, the design option is recommended based on the user-based collaborative-filtering method. Hence, it was usually chosen by similar profiles. Therefore, for each recommender method, the popover displays an explanation of whether a design option is recommended or not. The explanations for each recommender method are as follows:

Similar Architecture Profiles A design option can either be recommended or not recommended based on similar architecture profiles. In other words, a design option was selected or not selected in similar architecture profiles. Therefore, the result of this recommender method will either be "Recommended because this design option was selected in similar profiles, e.g.," with a list of the most similar profiles or "Not recommended because this design option was not selected in similar profiles, e.g.," also with a list of similar profiles. Additionally, a button "Further information" is displayed that will show a list of rationales for the decision in similar architecture profiles once it is clicked. This way, the software architect can see why a design option was selected or not selected in other architecture profiles.

Attributes of an Architecture Profile The case-based method will display important attributes in the architecture profile that are positively, negatively, or neutrally

influenced by the design option. Positively influenced attributes will have following description: "Recommended because following aspects are selected as important in this architecture profile and the design option positively influences these aspects: ...". Negatively influenced attributes will have following description: "Not recommended because following aspects are selected as important in this architecture profile and the design option negatively influences these aspects: ...". Finally, neutrally influenced attributes have following description: "This design option has a neutral influence on following aspects, i.e. it has a positive as well as a negative impact: ...".

Design Decisions of an Architecture Profile Similar to the case-based method, the content-based method shows the attributes of the user profile and how they are influenced by the design option. Positively influenced attributes have following description: "Recommended because following aspects are selected as important in the profile or decision documentations and the design option positively influences these aspects: ...". Negatively influenced attributes have following explanation: "Not recommended because following aspects are selected as important in the profile or decision documentations and the design option negatively influences these aspects: ...". Also, neutrally influenced attributes are considered as follows: "This design option has a neutral influence on following aspects, i.e. it has a positive as well as a negative impact: ...".

Required Design Options The knowledge-based recommender method for requires-relationships provides following result for each design option in other decision models that have a relationship to the current design option: "Recommended because $thedesignoption$ relates to the already selected design option $designoption$ in $decisionmodel$. Therefore, the recommendation score is 100.". Instead of $thedesignoption$ the user interface displays the name of the design option, $designoption$ is replaced by the name of the design option in the other decision model, and $decisionmodel$ is replaced by the name of the other decision model. As discussed in Section 4, requires-relationships have a significant impact on the overall score. Therefore, the overall score is 100 if the design option requires an already selected design option.

Excluded Design Options Similar to the requires-relationship, selected design options that exclude the current design option should be displayed in the popover. Therefore, the popover includes following information in this regards: "Not recommended because $thedesignoption$ is excluded by the selected design option $designoption$ in $decisionmodel$. Therefore, the recommendation score is 0.". $thedesignoption$ refers to the current design option in the decision model. $designoption$ is replaced by the name of the design option in the other decision model, and $decisionmodel$ is replaced by the name of the other decision model. Also, excludes-relationships have a significant impact on the overall score. If an excludes-relationship exists, the overall score of a design option is 0.

If there exists both requires- and excludes-relationships to a design option, a conflict

message will be displayed. The conflict message contains a list of all design options of other decision models and their relationship to the design option. For example, a conflict message can read as follows: "Conflict since $thedesignoption$ is: required by Service-side Discovery, excluded by Client-side Discovery.". Again, $thedesignoption$ is replaced by the name of the current design option. If a conflict exists, the design option has an overall score of 0.

To sum up the integration of the recommender system results for design options into the user interface, it can be said that the overall score is the first information the user sees when using a decision model as soon as an architecture profile gets selected. The overall score provides an immediate consolidated result of the recommender system. When the user clicks on the weighted overall score, a popover provides more detailed information about the results of each recommender method. Software architects can use this information to make the best decision for a given software architecture in a specific design space.

5.2 Architecture and Implementation

The previous section showed how the results of the recommender system are integrated into the user interface of the AKB tool. This section covers implementation aspects of the recommender system. Thereby, this section presents the general architecture of the recommender system by describing the components that were implemented and how they interact with each other. Additionally, this chapter describes the recommendation objects that are returned by the API of the recommender system.

5.2.1 Top-Level Software Architecture

As explained in Section 4.1, the server component of the AKB tool is a Java Application using the Spring Framework with the Spring Web MVC module. Therefore, a RestController, service, and a business component were created for the recommender system. These components and their interactions are shown in Figure 5.5.

The figure shows that the RestController offers two REST-interfaces. One interface to get recommendations for design options when using a specific decision model and another to get recommendations for decision models. Both interfaces require the ID of the corresponding architecture profile $apId$ in the URL. Since recommendations for design options should be made when using a particular decision model, the ID of the decision model $dmId$ is also required in this REST-interface of design options. Both the recommender system service and business component have two methods that take the ID of the architecture profile and decision model as parameter. One to get recommendations

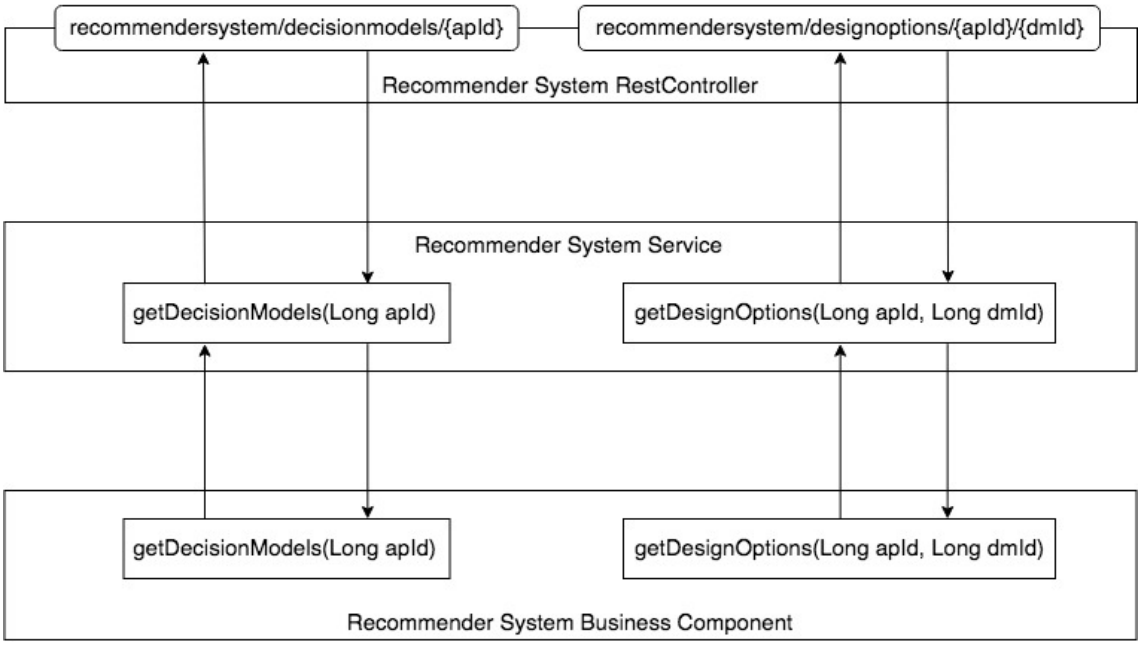


Figure 5.5: Spring Web MVC Recommender System Architecture

for decision models and another one to get recommendations for design options. The service, however, only calls the method of the business component. Finally, the business component orchestrates the recommender system. Therefore, the business component is responsible to decide which recommender methods to apply and how the results should be combined as described in Chapter 4. As for the hybridization techniques, if-statements are used in the business component to decide which recommender methods to apply. In the business component, configurations like the minimum number of made design decisions in an architecture profile can be made. These configurations are used in the if-statements of the switching- and mixed hybridization techniques. Finally, the business component returns the results of the recommender system to the service and the service forwards the results to the RestController. Afterward, the RestController sends a response to the client component with the results of the recommender system.

5.2.2 High-Level Component Interactions

In general, the recommender system business component uses three types of components to create recommendations. First, data preparation classes are responsible for querying required data from the Neo4j database and preparing them for further processing. Therefore, data preparation classes can be seen as the preprocessing steps for recommender algorithms. For collaborative-filtering methods, for example, the data preparation class would be responsible for creating the rating matrix. In case of content-based methods, the data preparation class creates the user profile. The second type of components are

the actual recommender algorithm implementations. They receive data from the data preparation classes and are responsible for creating the actual recommendations. For example, user-based collaborative-filtering is a recommender algorithm that takes the rating matrix and calculates recommendations as described in Section 3.3.1. The recommender algorithms were implemented as black-box algorithms. Therefore, the same recommender algorithm can be used for both design options and decision models. Only the data preparation classes are different for design options and decision models. Since most recommender algorithms use similarity measures in one way or the other, similarity measures refer to the third type of components. For example, the user-based collaborative-filtering algorithm uses the Pearson coefficient to determine the similarity between the active architecture profile and other architecture profiles. The Dice coefficient can be used to calculate the overlap of the user profile and the content of unrated items in a content-based approach. An overview of the three components and their interaction is shown in Figure 5.6.

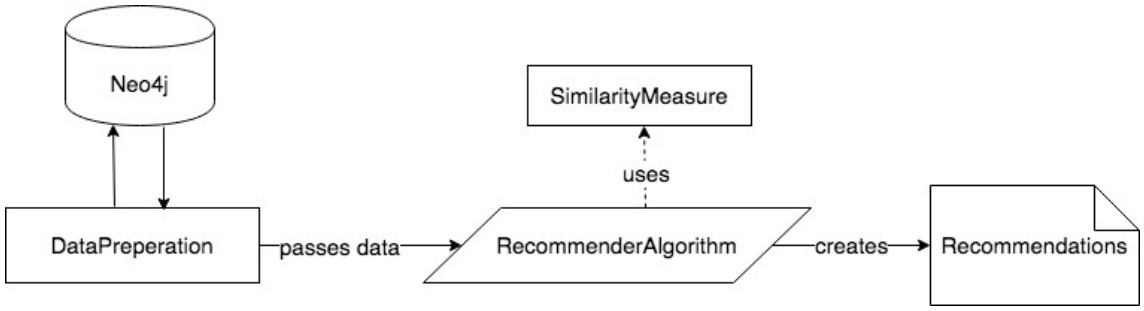


Figure 5.6: Recommender System Application Flow

The figure demonstrates the interaction between the data preparation class and the Neo4j database on the left side. The first step of each data preparation class is the querying of required data from the Neo4j database. This data gets manipulated to bring it in a form that can be used by the recommender algorithm. The necessary data preparation manipulations vary between the different recommender algorithms. However, the result of the data preparation class gets passed to the corresponding recommender algorithm. The recommender algorithm optionally uses a similarity measure in its algorithm. However, the calculation of similarities is not hard coded in a recommender algorithm. The idea is to make similarity measures exchangeable in a recommender algorithm to make it easy to compare the output using different similarity measures. Finally, the recommender algorithm creates the actual recommendations.

5.2.3 Recommender Algorithms

As mentioned earlier, the recommender algorithms were implemented as black-box algorithms to make them reusable for any kind of items. For this purpose, a RecommenderAlgorithm interface was created that has a $recommend()$ method. Therefore, each recommender algorithm implements this interface and has a $recommend()$ method that creates recommendations. Each recommender algorithm has further individual methods to set prepared data and configurations. The different recommender algorithms are presented in Figure 5.7.

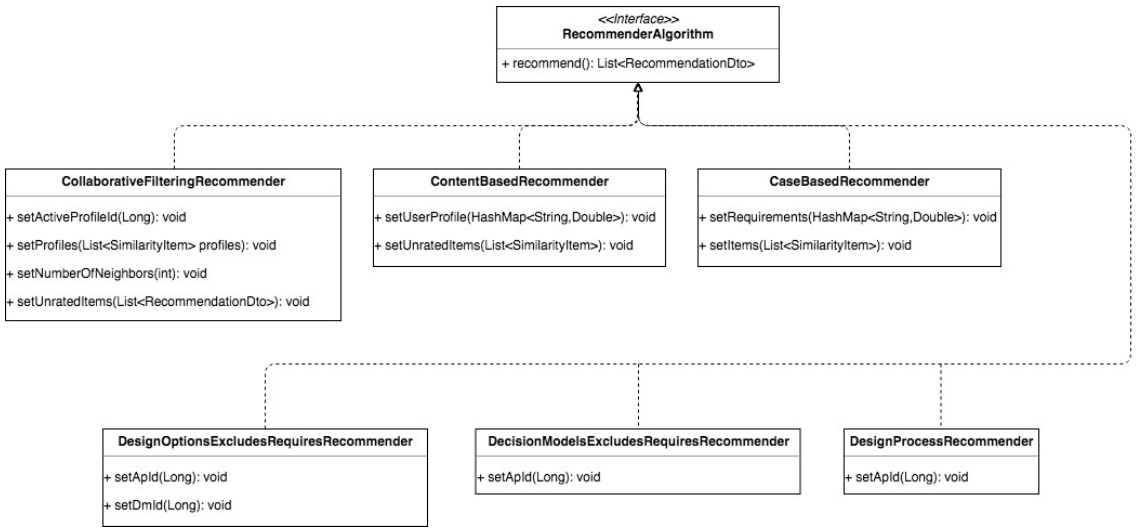


Figure 5.7: RecommenderAlgorithm Interface

CollaborativeFilteringRecommender The collaborative-filtering recommender method refers to the user-based collaborative-filtering algorithm. It can be configured by setting the active architecture profile id and the number of neighbors for the neighborhood selection. Furthermore, the rating matrix can be set that was created in the corresponding data preparation class by calling the $setProfiles()$ method. Also, the unrated items can be set for which recommendation predictions should be calculated.

ContentBasedRecommender The content-based recommender can be configured by setting the user-profile that was created in the data preparation class and the items that have not been rated yet.

CaseBasedRecommender Similar to the content-based recommender algorithm, the case-based recommender algorithm can be configured by setting the preferences or requirements and the items.

DesignOptionsExcludesRequiresRecommender This recommender algorithm only takes the architecture profile ID and the decision model ID as input. There is no data preparation needed since the final recommendations are created by the Neo4j query itself.

DecisionModelsExcludesRequiresRecommender Similar to the previous recommender algorithm, no data preparation is needed for this algorithm. Only the architecture profile ID needs to be set.

DesignProcessRecommender Also, this recommender algorithm needs no data preparation step. However, the architecture profile ID needs to be set for this recommender algorithm as well. The recommendations are computed by the Neo4j query itself.

5.2.4 Similarity Measures

Similar to the recommender algorithm interface, a $SimilarityMeasure$ interface was created. The goal of this interface is to bring similarity measures in a uniform form to make them exchangeable. Therefore, each $SimilarityMeasure$ implementation needs to implement the $calcSimilarity()$ method that takes two objects as parameters. The method calculates the similarity between these objects, and returns the similarity value of data type Double. For the recommender system, three similarity measures have been implemented. First, the Pearson coefficient for collaborative-filtering methods. Second, the case-based similarity measure as presented by Jannach et al. [20]. Finally, the Dice coefficient for the content-based method. The similarity measures are shown in Figure 5.8.

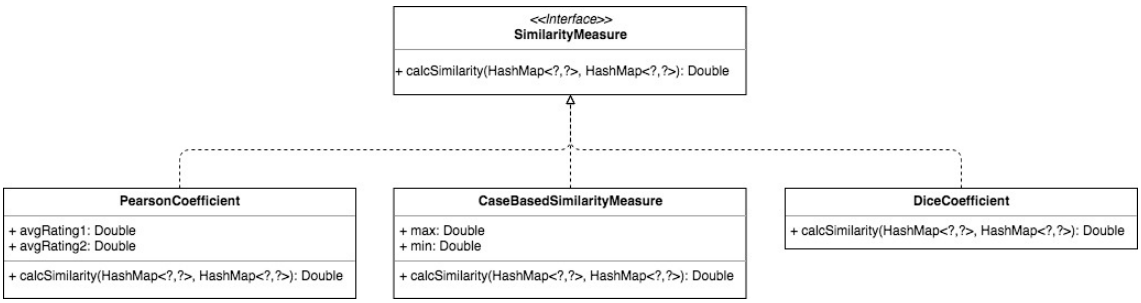


Figure 5.8: SimilarityMeasure Interface

The figure shows that the two parameters that get passed to the $calcSimilarity()$ method are key-value pairs. The key-value pairs are implemented as a HashMap in the Java Application. For example, to calculate the similarity between two architecture profiles based on used decision models, the keys would refer to the IDs of the decision

models and the values would indicate whether the decision model was used (1) or not (0). For the case-based recommender method, the keys would refer to the IDs of attributes and the value would refer to the influence a design option has on this attribute which is either positive (1), negative (-1), or neutral (0). Therefore, each similarity measure calculates the similarity between a map of key-value pairs. The Pearson coefficient requires the average rating of each object as input. The $CaseBasedSimilarityMeasure$ requires the max and min values as presented in the formula by Jannach et al. [20].

5.2.5 Recommender System API

Section 5.2.1 described that the recommender system is accessible by two REST-interfaces. One interface for the recommendation of design options and one interface for the recommendation of decision models. This section demonstrates the recommendation objects that get returned by these interfaces. As explained before, the recommender system returns the results of multiple recommender methods at the same time using the mixed hybridization technique. Furthermore, the weighted overall score is calculated and returned by the recommender system. Therefore, a list of all applied recommender methods including the weighted overall score needs to be returned. Each of these recommender methods must include a list of all items for which recommendations should be made. A recommender method refers to a $RecommendationListDto$ object in the recommender system as shown in Figure 5.9.

Figure 5.9: RecommendationListDto Class

The figure shows that a recommender method has a $recommendationType$ indicating which recommender method was used to make these recommendations. Furthermore, this object contains a list of $recommendations$ which refers to all items for which recommendations were made. These $recommendations$ refer to unrated items and include information about the item itself and information about the recommendation result of this item. For example, for the recommendation of design options in a decision model, a $RecommendationListDto$ object with the $recommendationType$ "casebased" would be created for the case-based recommender method. This object includes a $RecommendationDto$ object for each design option in the decision model in the $recommendations$ list. The $RecommendationDto$ object is shown in Figure 5.10.

The $RecommendationDto$ object contains the ID, title, and definition of the corre-

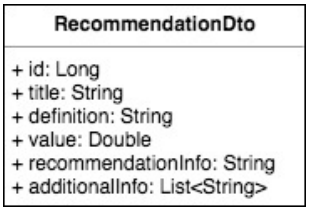


Figure 5.10: RecommendationDto Class

sponding item and additional information about the recommendation. In case of decision model recommendations, the $RecommendationDto$ object would contain the ID, title and description of the decision model. On the other hand, if design options should be recommended, the $RecommendationDto$ object would contain the ID, title and definition of the design option.

Each recommender algorithm produces some kind of recommendation results. For example, the collaborative-filtering method calculates a prediction value for each unrated item. The case-based method calculates the similarity between the requirements and the content of items. In the recommender system, these results got generalized into a $value$ property. Since every recommender algorithm calculates some kind of value for items, it made sense to store this value in the recommendation. This also allows to order the recommendations by this value. For example, the prediction value calculated by the collaborative-filtering recommender gets stored in this property. Furthermore, a $RecommendationDto$ object has a $recommendationInfo$ property. This property can be used to define why an item was recommended or not recommended. For example, the content-based method could provide the properties of a decision model that lead to the recommendation. The recommendation information of the different recommender methods were presented in the previous section. Finally, more detailed information can be stored in the $additionalInfo$ property. For example, in the AKB tool, this property is used to store the rationales for decisions of design options of similar profiles in a collaborative-filtering algorithm. To sum up the $RecommendationDto$ class, it can be said that it contains information about the recommended item, on the one hand, and information about why it is recommended or not recommended, on the other hand.

Also, a $RecommendationListDto$ object is created for the weighted overall score of items. In this case, the $recommendationType$ is "overallscore" and the $recommendations$ list includes all unrated items with the computed weighted overall score in the $value$ property. Consequently, the interfaces of the recommender system will return a list of $RecommendationListDto$ objects.

This section demonstrated that the hybrid recommender systems for design options and decision models are accessible through REST-interfaces. Only the ID of the architecture profile and, in case of design options, the ID of the decision model is required. Data preparation classes are responsible for the preprocessing of data. Recommender algorithms are implemented as black-boxes to make them usable for any kind of items.

Similarity measures were implemented in a way to make them exchangeable. This way, different similarity measures can be used in recommender methods to determine the similarity measure that delivers the best results. Finally, the REST-interface returns a list of $RecommendationListDto$ objects with each referring to an applied recommender method or to the results of the weighted overall score. The $RecommendationListDto$ objects contain a list of $recommendations$ which include information about the items and more detailed information about the recommendation results.

Chapter 6

Conclusion

The goal of this thesis was to develop a recommender system for decision making in software architecture design. To reach this goal, existing concepts and approaches for recommender systems were first analyzed based on literature and then an existing tool for software architecture knowledge management was extended with recommender systems for design decision models and design options.

In the course of this thesis, two hybrid recommender systems were developed. One for the recommendation of decision models and one for the recommendation of design options. In both cases it was necessary to include knowledge-based recommender methods into the system to deal with the cold-start problem. Thereby, it is guaranteed that the system can make recommendations even if there is little known about a software system. Regarding decision models, a content-based recommender method was implemented to find decision models similar to those already used for decision making and documentation in an architecture profile. Also, in the hybrid recommender system for design options, a content-based recommender method was used to recommend design options that have a similar influence on aspects to those already selected in an architecture profile. Finally, user-based collaborative-filtering recommender methods were implemented to recommend decision models used in similar architecture profiles and design options selected in similar architecture profiles. Using the mixed hybridization technique the results of all recommender methods are presented to the user side by side. Also, the weighted hybridization technique was applied to compute an overall score for each recommended decision model or design option. In addition, the switching hybridization technique was used to switch between the case-based and content-based recommender method as both deliver a similar result.

The developed hybrid recommender systems have been prototypically tested. For this purpose, decision models in the context of microservices have been created and added to the AKB tool. These decision models cover various design spaces like monitoring, fault tolerance, and service discovery. Afterward, multiple architecture profiles have been

created with the support of the recommender system. The goal of the prototypical tests was to ensure that the recommender systems deliver the expected results. However, an empirical evaluation of the system is missing and would be necessary in order to make stronger statements about the suitability of the recommender system for the decision-making process in a real architectural design scenario.

An important aspect of the developed recommender systems is that the user-based collaborative-filtering method only makes sense if a reasonable number of architecture profiles already exist in the AKB tool. The reason for this is that there should be at least twenty similar profiles available for a specific architecture profile in order to make recommendations when using collaborative-filtering. This can only be guaranteed for each architecture profile if many software systems are documented in the AKB tool. Otherwise the selected architecture profiles could be too different from the active architecture profile and might result in insufficient recommendations. In comparison to the other recommender methods, the knowledge-based methods are particularly suitable at the beginning of the decision-making process to receive recommendations. The content-based methods will not make proper recommendations at the beginning and should therefore be used at a later point in time. However, content-based methods should provide better results than knowledge-based methods when more architectural knowledge is documented.

With the developed recommender system, software architects receive recommendations for decision models and design options in the decision-making process. Based on this recommendations, software architects should be able to make better decisions for a software system's architecture. The recommender system is designed in a way that the quality of the recommendations increases with the amount of architectural knowledge that is captured using learning-based recommendation techniques. However, the system provides recommendations even at the beginning of the decision-making process. As a result, this should lead to proper design decisions from the beginning of the architectural design process and will avoid costly and time-intensive changes to the architecture later on.

It is obvious that the recommender system could be designed in a different way because of the amount of available hybridization techniques to combine multiple recommender methods. This thesis presented one possible implementation of a recommender system. One aspect that is currently not addressed by the developed recommender system is whether an already made design decision was a satisfying decision or not. This aspect could be addressed in future extensions of the system. For example, the collaborative-filtering method recommends design options selected in similar architecture profiles. However, if it turns out that this design option was not suitable for this type of software architecture, it still gets recommended in new software projects. Therefore, the recommender system should not only consider if a design option was selected but also if it was a good decision to select this design option. For this purpose, the AKB tool could be extended by a feature to let software architects rate the made design decisions and incorporate this knowledge into the recommender system. Additionally, when it comes to the classification of decision models, a list of properties is used. All properties in this list are

considered equally important. However, some properties may have more significance for the classification than others. For example, the property Microserivce could be more important for the classification of a decision model than the property Monitoring. Therefore, it might be appropriate to use weighted properties to make recommendations. For example, the weight of a property could be defined either by the software architect or by the number of occurrences of the property in the AKB tool.

List of Figures

List of Tables

Bibliography

[1] R. Weinreich and I. Groher, "Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review," *Information and Software Technology*, vol. 80, pp. 265–286, Dec. 2016.

[2] R. N. Taylor, N. Medvidović, and E. M. Dashofy, *Software architecture: foundations, theory, and practice*. Hoboken, NJ: John Wiley, 2010. OCLC: ocn231670965.

[3] M. Ali Babar, ed., *Software architecture knowledge management: theory and practice*. Dordrecht ; New York: Springer, 2009. OCLC: ocn428028065.

[4] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for software architecture and tools to support them," *IEEE transactions on software engineering*, vol. 21, no. 4, pp. 314–335, 1995.

[5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 2nd edn. SEI Series in software engineering*. Addison-Wesley Pearson Education, Boston, 2003.

[6] J. Bosch, "Software architecture: The next step," in *European Workshop on Software Architecture*, pp. 194–199, Springer, 2004.

[7] P. Liang and P. Avgeriou, "Tools and technologies for architecture knowledge management," in *Software Architecture knowledge management*, pp. 91–111, Springer, 2009.

[8] M. Keeling, "Architecture Haiku: A Case Study in Lean Documentation [The Pragmatic Architect]," *IEEE Software*, vol. 32, pp. 35–39, May 2015.

[9] G. Fairbanks, "Architecture Haiku," 2010. URL: https://www.georgefairbanks.com/assets/pdf/Haiku-tutorial-2011-06-24-final.pdf, Accessed: 26.9.2018.

[10] S. Haselböck, R. Weinreich, and G. Buchgeher, "Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements," in *Software Architecture* (A. Lopes and R. de Lemos, eds.), vol. 10475, pp. 155–170, Cham: Springer International Publishing, 2017.

[11] A. MacLean, R. M. Young, V. M. Bellotti, and T. P. Moran, "Questions, options, and criteria: Elements of design space analysis," *Human–computer interaction*, vol. 6, no. 3-4, pp. 201–250, 1991.

[12] A. Maclean and D. McKerlie, "Design space analysis and use-representations," *Scenario-based design: envisioning work and technology in system development*, pp. 183–207, 1995.

[13] O. Zimmermann and C. Miksovic, "Decisions required vs. decisions made: connecting enterprise architects and solution architects via guidance models," in *Aligning Enterprise, System, and Software Architectures*, pp. 176–208, IGI Global, 2013.

[14] G. A. Lewis, P. Lago, and P. Avgeriou, "A decision model for cyber-foraging systems," in *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*, pp. 51–60, IEEE, 2016.

[15] S. Haselböck, R. Weinreich, and G. Buchgeher, "Decision guidance models for microservices: service discovery and fault tolerance," in *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems - ECBS '17*, (Larnaca, Cyprus), pp. 1–10, ACM Press, 2017.

[16] S. Haselbock and R. Weinreich, "Decision Guidance Models for Microservice Monitoring," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, (Gothenburg, Sweden), pp. 54–61, IEEE, Apr. 2017.

[17] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds., *Recommender Systems Handbook*. New York: Springer, 2011. OCLC: ocn373479846.

[18] J. A. Konstan and J. Riedl, "Recommender systems: from algorithms to user experience," *User Modeling and User-Adapted Interaction*, vol. 22, pp. 101–123, Apr. 2012.

[19] X. Sun, F. Kong, and S. Ye, "A comparison of several algorithms for collaborative filtering in startup stage," in *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.*, pp. 25–28, Mar. 2005.

[20] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems An Introduction*. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2011.

[21] J. Zhou and T. Luo, "Towards an Introduction to Collaborative Filtering," in *2009 International Conference on Computational Science and Engineering*, vol. 4, pp. 576–581, Aug. 2009.

[22] R. Burke, "Hybrid Web Recommender Systems," in *The Adaptive Web: Methods and Strategies of Web Personalization* (P. Brusilovsky, A. Kobsa, and W. Nejdl, eds.), pp. 377–408, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[23] J. K. Tarus, Z. Niu, and G. Mustafa, "Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning," *Artificial Intelligence Review*, Jan. 2017.

[24] R. Burke, "Hybrid Recommender Systems: Survey and Experiments," *User Modeling and User-Adapted Interaction*, vol. 12, pp. 331–370, Nov. 2002.

[25] C. C. Aggarwal, *Recommender Systems*. Switzerland: Springer International Publishing, 2016.

[26] F. Ge, "A User-Based Collaborative Filtering Recommendation Algorithm Based on Folksonomy Smoothing," in *Advances in Computer Science and Education Applications: International Conference, CSE 2011, Qingdao, China, July 9-10, 2011. Proceedings, Part II* (M. Zhou and H. Tan, eds.), pp. 514–518, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[27] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," in *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 158–167, ACM, 2000.

[28] Z. Huang, D. Zeng, and H. Chen, "A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce," *IEEE Intelligent Systems*, vol. 22, pp. 68–78, Sept. 2007.

[29] A. Sureka and P. P. Mirajkar, "An Empirical Study on the Effect of Different Similarity Measures on User-Based Collaborative Filtering Algorithms," in *PRICAI 2008: Trends in Artificial Intelligence: 10th Pacific Rim International Conference on Artificial Intelligence, Hanoi, Vietnam, December 15-19, 2008. Proceedings* (T.-B. Ho and Z.-H. Zhou, eds.), pp. 1065–1070, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[30] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 230–237, ACM, 1999.

[31] L. Ren, J. Gu, and W. Xia, "An Item-Based Collaborative Filtering Algorithm Utilizing the Average Rating for Items," in *Signal Processing and Multimedia: International Conferences, SIP and MulGraB 2010, Held as Part of the Future Generation Information Technology Conference, FGIT 2010, Jeju Island, Korea, December 13-15, 2010. Proceedings* (T.-h. Kim, S. K. Pal, W. I. Grosky, N. Pissinou, T. K. Shih, and D. Slezak, eds.), pp. 175–183, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[32] C. Birtolo, D. Ronca, R. Armenise, and M. Ascione, "Personalized suggestions by means of Collaborative Filtering: A comparison of two different model-based techniques," in *2011 Third World Congress on Nature and Biologically Inspired Computing*, pp. 444–450, Oct. 2011.

[33] Z. Yun-tao, G. Ling, and W. Yong-cheng, "An improved TF-IDF approach for text classification," *Journal of Zhejiang University-SCIENCE A*, vol. 6, pp. 49–55, Aug. 2005.

[34] M. J. Pazzani and D. Billsus, "Content-Based Recommendation Systems," in *The Adaptive Web: Methods and Strategies of Web Personalization* (P. Brusilovsky, A. Kobsa, and W. Nejdl, eds.), pp. 325–341, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[35] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "Developing Constraint-based Recommenders," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds.), pp. 187–215, Boston, MA: Springer US, 2011.

[36] M. Zanker, M. Aschinger, and M. Jessenitschnig, "Development of a Collaborative and Constraint-Based Web Configuration System for Personalized Bundling of Products and Services," in *Web Information Systems Engineering – WISE 2007: 8th International Conference on Web Information Systems Engineering Nancy, France, December 3-7, 2007 Proceedings* (B. Benatallah, F. Casati, D. Georgakopoulos, C. Bartolini, W. Sadiq, and C. Godart, eds.), pp. 273–284, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[37] B. Smyth, "Case-Based Recommendation," in *The Adaptive Web: Methods and Strategies of Web Personalization* (P. Brusilovsky, A. Kobsa, and W. Nejdl, eds.), pp. 342–376, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[38] D. McSherry, "Similarity and compromise," in *International Conference on Case-Based Reasoning*, pp. 291–305, Springer, 2003.

[39] "AngularJS MVW Framework," 2018. URL: https://angularjs.org/, Accessed: 13.08.2018.

[40] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack, and others, "The spring framework–reference documentation," *Interface*, vol. 21, p. 27, 2004.

[41] "Neo4j," 2018. URL: https://neo4j.com/, Accessed: 13.08.2018.