The present work was submitted to
the Research Group
Software Construction

of the Faculty of Mathematics,
Computer Science, and
Natural Sciences

Bachelor Thesis

Conception of a Security Solution Pattern Catalog for Constraint-based Recommender Systems

presented by

Arben Abazi

Aachen, September 27, 2024

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Alex Sabau, M.Sc.

Acknowledgment

First and foremost, I would also like to express my appreciation to Prof. Dr. rer. nat. Horst Lichter for allowing me to write this thesis at the Research Group Software Construction and for reviewing my work together with Prof. Dr. rer. nat. Bernhard Rumpe.

I would also like to extend my deep and sincere gratitude to my supervisor, Alex Sabau, for providing me with the opportunity to engage in what truly felt like my first experience in academic research. His continuous feedback, guidance, and support were invaluable throughout the entire process of working on this thesis. It always felt like a collaborative effort.

Special thanks go to my family, who have consistently supported me throughout my academic journey, despite its ups and downs. Their unwavering belief in me and their constant encouragement will never be forgotten. I am also deeply grateful to Metehan and especially Yaren, who have been by my side since the very beginning of my computer science studies. This journey has truly been a team effort, marked by the challenges we faced together, whether it was working on submissions or preparing for exams. In the end, we made it!

Thank you.

Arben Abazi

Abstract

Software patterns offer a promising approach to enhancing the security of software products, particularly during the early stages of development. The novel combination of security patterns with recommender systems has the potential to improve overall software security significantly. A crucial element of such a system is the knowledge base, which serves as the reasoning component for selecting the most appropriate pattern. In this thesis, we review 27 peer-reviewed articles to identify concepts for creating a catalog that aids in pattern selection. As a result, we introduced security solution patterns, a specialized subset of security patterns designed to be both practical and specifically tailored for use in recommender systems. We adopted and synthesized key concepts for organizing a catalog of security solution patterns, along with classification factors that define their essential features and provide relevant contextual information about their applicability. By encoding these concepts, we facilitate the implementation of the catalog within knowledge bases, enabling their use in recommender systems. These findings establish a solid foundation for the practical implementation of a knowledge base, providing a functional framework for integrating patterns into recommender systems and advancing the development of secure software architecture design.

Contents

List of Tables

List of Figures

1 Introduction

In today's rapidly growing digital environment, security has become a crucial issue for organizations worldwide. As reliance on digital technologies increases, the potential for cyber attacks also grows, making it critical for software systems to include strong security measures. This need is especially urgent with the rise of new platforms and devices such as the Internet of Things (IoT), smartphones, and cloud computing services. As a result, security threats have intensified, and the frequency of security breaches is on the rise [Asl+23]. A significant number of these breaches occur due to vulnerabilities in software systems [Asl+23]. Moreover, the lack of skilled software security professionals further worsens the problem [RK20; Sab24].

The software development process comprises multiple stages, and addressing security vulnerabilities early – particularly during the design and architectural phases – is crucial for building secure software. Thus, the Research Group Software Construction at RWTH Aachen University seeks to develop support in the early secure design of software systems in the "Security-Centered Architecture Modeling (SCAM)" research project. One approach is a recommender system designed to reduce security risks during the architectural design phase. The system leverages the concept of security patterns, which encapsulate best practices for addressing specific security concerns in software design. Building on this, we introduce security solution patterns, specialized subsets of security patterns specifically tailored for use in a recommender system, which are defined in detail in Chapter 4. The purpose of this recommender system is to recommend and integrate relevant security solution patterns into the software architecture, thereby establishing a secure foundation for further development.

At the core of this recommender system is a knowledge base built around a catalog of security solution patterns. This catalog forms the foundation for the reasoning process, enabling the system to select the most appropriate patterns. Effective organization and classification of the patterns within the catalog are essential to ensure that the system delivers accurate and relevant recommendations. This structured approach allows the reasoning engine to identify the most suitable patterns for specific security challenges. The reasoning is then employed by the constraint-based recommender system to provide precise and tailored security recommendations.

This chapter introduces the research question addressed in this thesis and outlines the methodology used for developing a conception of a catalog of security solution patterns.

1.1 Research Question

The primary objective of this thesis is to develop a framework for the catalog of a knowledge base to be utilized in the SCAM project's recommender system. To achieve this, the thesis investigates methods for effectively organizing and classifying security solution patterns to enable efficient decision-making for recommender systems. This leads to the research question addressed in this thesis:

RQ: How can security solution patterns be effectively organized and classified to be realized by knowledge bases to effectively reason about them?

Effective organization and classification of security solution patterns within a catalog are crucial for enabling precise and accurate reasoning. This involves identifying key attributes that are both expressive and relevant to the reasoning process, as well as grouping related patterns to facilitate organization and support the selection of appropriate patterns.

Since the reasoning will be implemented within a knowledge base used by a constraint-based recommender system, this thesis proposes an approach to encode the catalog in a machine-readable format. The organization and classification framework must be logically structured and machine-interpretable to ensure that the system can make accurate and suitable recommendations based on the catalog.

1.2 Methodology

To discover suitable concepts for the catalog base of security solution patterns, a light systematic literature review (SLR) was conducted. Given the time constraints of this thesis, a full SLR was not feasible; however, the approach used aimed to capture an overview of the current state of research on security patterns.

We began by searching the Scopus database using the following search string:

Search Query: *("security patterns" OR "security design patterns")*
AND
("systematic literature review" OR "slr" OR "survey")

The search was restricted to papers published from 2019 onwards to focus on the most recent developments in the field. Additionally, we limited the results to the subject area of computer science to ensure relevance. This search yielded 14 papers, the majority of which were concentrated on specific fields, such as IoT. However, for the purpose of this thesis, we focused primarily on survey papers that addressed broader topics.

To expand the scope of our review and identify more relevant literature, we employed the snowballing technique. This involved examining the references cited in the identified systematic literature reviews to discover additional papers of interest. Through this method, we were able to locate key papers that contributed to the classification and organization of security patterns in existing catalogs. Since the goal of our catalog of security solution

patterns, we paid particular attention to how other authors structured and classified patterns in their respective catalogs.

Based on the findings from the literature, we evaluated the various concepts introduced by other researchers. These evaluations informed the adoption of the most suitable ones for our catalog, allowing it to effectively organize and classify security solution patterns for reasoning about the content.

This methodology ensured that our framework was informed by current research trends while remaining tailored to the specific requirements of a constraint-based recommender system.

2 Related Work

The landscape of security patterns has been the subject of several researches. In this chapter we will present the related work consisting of survey papers and existing catalogs containing security patterns.

2.1 Survey Work

In this section two comprehensive survey papers will be presented.

Washizaki et al.'s systematic literature review of 240 papers offers a structured and comprehensive overview of existing security patterns and their practical applications [Was+21]. Their work presents a detailed taxonomy categorizing security pattern research, addressing key challenges in effectively applying these patterns. This taxonomy aims to improve communication between practitioners and researchers, standardize the often inconsistent terminology, and enhance the usefulness of security patterns beyond the traditional Confidentiality, Integrity, and Availability (CIA) triad.

Washizaki et al.'s review offers critical insights into the broader security pattern landscape, which directly informs the conception of our knowledge base. Their analysis revealed multiple trends within security research, particularly the prominence of security objectives in most of the patterns they reviewed. This finding is crucial as it suggests that security objectives can serve as a fundamental criterion for classifying patterns in our proposed knowledge base. Moreover, approximately 25% of the patterns are platform-specific, addressing domains, such as cloud computing, web applications, and distributed systems. This underscores the necessity for our knowledge base to accommodate platform specificity when classifying security patterns.

However, despite these valuable contributions, Washizaki et al. only briefly mention platform specificity without providing a comprehensive taxonomy for classifying security patterns based on these criteria. This gap highlights the need for a more detailed and structured approach to pattern classification, particularly in terms of platform considerations, which our concept aims to address.

Further supporting this need for a standardized catalog of security patterns, Jafari and Rasoolzadegan conducted a systematic mapping study that reviewed security pattern research from 1997 to 2017 [JR20]. Their study employed a comprehensive search strategy, including manual searches, backward snowballing, and database searches across four scientific libraries. 403 papers were retrieved, but only 274 were analyzed based on quality criteria. The research identified several key classification criteria for security patterns, including security objectives,

development lifecycle phases, and application environments.

A major finding of the study was that the lack of consensus on classification methods among researchers is a persistent issue, as it results in disparate catalogs of security solutions that complicate the practical application of these patterns. This challenge is echoed in Washizaki et al.'s findings, further demonstrates the necessity of a consistent and well-structured taxonomy.

Thus, this thesis aims to contribute to the field by developing a concept for a knowledge base of security patterns that standardizes the classification process for recommender systems. By incorporating as many relevant factors as possible – such as security objectives, platform specificity, and other emerging trends – this concept aims to facilitate a more robust reasoning process when selecting appropriate security patterns. Ultimately, this concept will make it easier for recommender systems to choose suitable security solutions and improve the overall software design process.

2.2 Existing Catalogs

There are several existing catalogs of security patterns; however, none of the current catalogs are suitable for use in recommender systems. In this section, we will review the related work on existing security pattern catalogs, examine their methodologies for catalog development, and explain why they are not well-suited for integration with recommender systems.

2.2.1 Catalog by Cordeiro, Vasconcelos, and Correia

One notable example is the catalog developed by Cordeiro, Vasconcelos, and Correia, which contains 106 security patterns that address a broad range of security concerns in software systems [CVC22]. This work introduces a uniform and up-to-date catalog designed to improve navigability and usability for both researchers and organizations. The catalog creation process followed four key phases: extraction, filtration, conversion, and classification. These phases aimed to ensure quality and consistency by selecting relevant patterns, applying qualitative metrics, standardizing descriptions, and categorizing the patterns effectively.

The authors evaluated the catalog utilizing the MITRE ATT&CK library, which is a guide that explains how hackers typically attack systems. They demonstrated that it could either mitigate or detect 175 out of 191 attacks, showcasing its diversity and comprehensiveness. Nonetheless, approximately 36% of these patterns were found to lack complete descriptions in certain areas, such as implementation and dynamics.

Although the catalog was evaluated positively in addressing a variety of security threats, it is not directly applicable to recommender systems. One significant issue with the classification arises when multiple patterns address the same security concern. For example, when searching for an authentication pattern, there is no clear metric to easily differentiate be-

tween the patterns. This makes it difficult to quickly identify the most appropriate solution. Furthermore, the relationships between patterns are not consistently documented. While some patterns are noted as alternatives or specializations of others, these relationships are not uniformly maintained. In some cases, a pattern is described as a specialization of another, but the generalized pattern does not reflect this connection. This inconsistency leads to vague abstraction layers, with some patterns being too abstract to integrate effectively into a software architecture.

To address these limitations, our work introduces additional classification factors to better distinguish between similar patterns. Moreover, we will implement a pattern hierarchy that clearly defines the abstraction level of each pattern, providing a more structured and practical framework for selecting and applying security patterns specifically within recommender systems.

2.2.2 Catalog by Rath et al.

Another significant contribution to the field of security patterns for cloud applications is the work by Rath et al., which focuses on developing security best practices and documentation specifically for Software-as-a-Service (SaaS) developers building Cloud applications [Rat+19]. The primary goal of this research is to provide a comprehensive set of guidelines for securing Cloud SaaS applications from the ground up. This is achieved through a methodical approach that involves five key steps: identifying security requirements, conducting risk assessments, identifying relevant security features, defining security patterns, and finally, classifying those patterns into distinct categories, such as system security, data security, communication security, and privacy.

The key outcome of the study is a detailed catalog of Cloud SaaS security patterns, covering crucial security aspects. The patterns are mapped to solutions provided by popular cloud platforms, such as Amazon Web Services (AWS) and Microsoft Azure, making them highly relevant for real-world applications. These patterns offer practical and actionable solutions for many security challenges that SaaS developers face.

In relation to our work, the research by Rath et al. focuses solely on security patterns for Cloud applications and is limited to two specific service providers: AWS and Azure. Additionally, the suggested solutions are mostly best practices explained in text form instead of specific design recommendations for the software architecture. This distinction helps clarify the type of security patterns we aim to include in our knowledge base, which will focus more on design-level recommendations that are easier to integrate into software architectures. Furthermore, the catalog developed in their work was designed for human users through a graphical interface, whereas our knowledge base will be designed to be machine-interpretable, specifically by recommender systems.

2.2.3 Catalog by Den Berghe, Yskout, and Joosen

Den Berghe, Yskout, and Joosen aimed to develop a comprehensive and well-organized catalog of software security patterns designed to enhance the security of software systems [DYJ22]. Their primary objective was to provide an accessible, up-to-date, and structured repository of best practices for software security design.

To achieve this, the authors adopted a systematic approach to identify, refine, and categorize existing software security patterns. Building on previous research and practical implementations, they reviewed security patterns across various domains. Their methodology involved analyzing current patterns and restructuring them to form a cohesive catalog that addresses modern security challenges and solutions more effectively. The resulting catalog was found to better meet contemporary software security needs compared to previous collections, as the authors identified gaps and addressed them by introducing more organized classifications and detailed implementation descriptions for the software architecture in each pattern.

This catalog contains only 23 security patterns and is primarily designed for pattern selection via a graphical interface. While the interface groups related patterns, it does not allow for easy differentiation between them without examining and comparing each one individually. This limitation mirrors an issue identified by Cordeiro, Vasconcelos, and Correia, highlighting the need for improvement in this area. In our case, since the pattern selection will be conducted by a machine, it is crucial for the system to differentiate between similar patterns by providing detailed information about their distinguishing features.

In this chapter, we reviewed the related work for this thesis and highlighted gaps in the classification of security patterns. Furthermore, a clearer overview of the platform specifications is needed. Moving forward, it is essential to make the classification machine-interpretable, with a particular focus on its application in recommender systems.

3 Background

This chapter provides the foundational knowledge necessary to understand the key concepts of this thesis. It starts with an overview of security patterns and then provides a brief introduction to recommender systems, emphasizing the aspects that are most relevant to this thesis.

3.1 Security Pattern

This section explores the concept of patterns and outlines the key specifications relevant to this thesis, specifically focusing on software architecture patterns and security patterns. These patterns will be defined and explained based on the study by Schumacher et al. [Sch03; Sch+06].

In 1977, architect and design theorist Christopher Alexander introduced the concept of design patterns in his book "A Pattern Language" [Ale18]. The key idea behind patterns for architects is to provide design solutions to common problems in architecture and urban planning. According to Schumacher et al., patterns are defined as follows [Sch+06]:

Definition. *" [A pattern is] a solution to a problem that arises within a specific context".*

The core components of patterns include the context, problem, and solution:

- **Context**. This refers to the environment and conditions that exist prior to the application of a pattern. It defines the conditions necessary for the problem and solution, thereby illustrating when and where the pattern is applicable.
- **Problem**. This represents a nontrivial recurring issue within the specified context. The term "nontrivial" indicates that the problem requires an expert for resolution, and "recurring" suggests that it frequently appears within the context.
- **Solution**. This component defines the essential principle for addressing the problem and describes thereby a proven way to solve the problem. This means that the suggested approach must have been successfully applied at least once.

In addition to the three components mentioned, patterns can also include elements like "consequences" and "see also." These elements provide an overview of the results of using the pattern and suggest related patterns to explore.

For software engineering, the adoption of patterns has also been recognized as advantageous. This became mainstream, particularly after the publication by the Gang of Four, who

introduced numerous design patterns for object-oriented programming [Gam+95]. Today, patterns are a fundamental concept in the field of software engineering and architecture.

According to Schumacher et al., a software architecture pattern is defined as follows [Sch+06]:

Definition. *"A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts and presents a well-proven generic solution for it."*

The *Microservice Pattern* is an example of a software architecture pattern [Ric18]. It emerged as a solution to the challenges caused by monolithic architecture in large and complex applications. As software systems grow, traditional monoliths become harder to manage, update, and scale. Especially since a single failure can affect the whole system, and any change can pose unwanted side effects. The proposed solution by Richardson addresses the challenges by breaking down the application into smaller, self-contained services, whereas each service focuses on a specific functionality and operates independently from each other. This approach offers significant benefits in terms of scalability and maintainability.

Software architecture patterns represent a specific category of design patterns focused on the overall structure and organization of software systems. In addition to these, there are security patterns, which are specialized patterns aimed at addressing security concerns. Schumacher et al. defines those specific patterns as follows [Sch+06]:

Definition. *"A security pattern (SP) for software architecture describes a particular recurring security problem that arises in specific design contexts and presents a well-proven generic solution for it."*

The *Authenticator Pattern* serves as an example of a security pattern designed to protect valuable resources within computer systems, such as business plans or medical records. These sensitive assets should only be accessible to legitimate users with valid reasons. This pattern addresses the challenge of restricting access to authorized users by implementing an authentication mechanism to verify user identities. While this example provides a brief overview of applying a security pattern, it does not delve into all its complexities.

3.2 Recommender System

This section presents recommender systems and their essential components, and it relies on [FB08], if not indicated differently. It starts with an overview of recommender systems in general, then delves into constraint-based recommender systems – a particular kind that the SCAM project aims to develop. Lastly, it introduces the concept of knowledge, a fundamental element of recommender systems, which will be explored in greater detail in a subsection.

In today's world, recommender systems have become important to daily life. Whether searching for information on Google, shopping on Amazon, or scrolling through TikTok, each platform utilizes a system that filters data and suggests the most relevant items [RJM20].

Regardless of the specific application, every recommender system involves two core components: a *user* and *items*. In this context, the user represents an individual to whom items are recommended. For instance, when searching for a product on Amazon, the user is the customer, while the items refer to the available products. A recommender system in this scenario suggests products by considering various factors, such as the user's search query, past purchases, and browsing history.

This thesis adopts the definition of a recommender system as proposed by Felfernig and Burke, which describes it as follows [FB08]:

Definition. *"[A recommender system is a] system that guides a user in a personalized way to interesting or useful objects in a large space of possible options, or that produces such objects as output."*

Constraint-based Recommender Systems are a specific type of recommender systems that suggest items by matching user preferences and requirements with predefined rules and constraints [FB08]. These constraints may arise from user specifications, such as particular preferences, or from the domain of the items themselves.

The SCAM project, as introduced in Chapter 1, aims to develop a constraint-based recommender system tailored to software architects. This system provides recommendations for security solution patterns, which are a special subset of security patterns. These special types will be further explained and defined in Section 4.1. The developed constraint-based recommender system aims to enable architects to enhance the security of their software architecture through the integration of appropriate security solution patterns with the overall architecture description [Sab24].

3.2.1 Knowledge

In this subsection, the sources of knowledge relevant to recommender systems will be presented based on the study by Felfernig and Burke [FB08]. First, the general concept of knowledge sources will be introduced, followed by a categorization of these sources. The focus will then shift to content knowledge, the category most relevant to this thesis, which will be explored in greater detail.

The knowledge utilized by recommender systems plays an important role in making suggestions, as it is based on this knowledge that the systems identify and recommend suitable items to users. As described by Felfernig and Burke, there are three primary sources of knowledge in recommender systems: social, individual, and content knowledge. Social knowledge is derived from the collective behavior of other users within the system, while individual knowledge originates from individual users. Content knowledge, on the other hand, includes information related to the items themselves and the domain in which the recommendation occurs.

This thesis concentrates specifically on content knowledge as the primary source of knowledge for the recommender system under study. The focus on content is justified by its relevance to the security pattern recommendations being explored in this work. Figure 3.1 illustrates the taxonomy of the relevant content and knowledge sources adopted in this research.

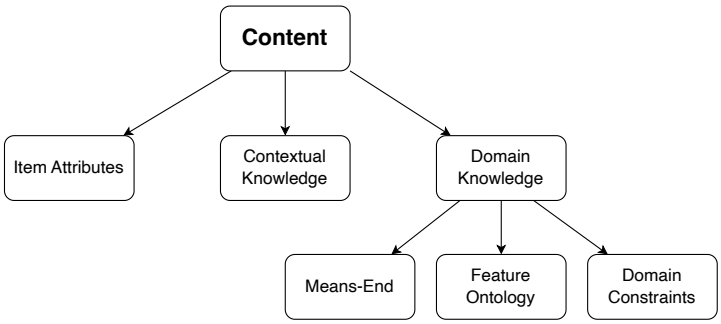

Figure 3.1: Taxonomy of the content Knowledge [FB08]

The content knowledge for this study is divided into three main categories:

- **Item Attributes** are characteristics and properties of an item that are used to describe and identify them.
- **Contextual Knowledge** refers to the consequences of external circumstances associated with the user's situation.
- **Domain Knowledge** gives more information about the recommended content and the purpose it fulfills. This category can also be further categorized into three different types:
  1. **Means-End** knowledge depicts which specific measure (the means) will be effective in achieving the desired goal (the ends) within that specific context.
  2. **Feature Ontology** is the understanding of how different features, components, or attributes within a system, are related and interact with each other.
  3. **Domain Constraints** are rules or limitations that must be followed within a particular area. These constraints define what is possible or acceptable within a particular area or field.

After identifying the factors that influence the recommender system's suggestions, these factors will be categorized based on the outlined types of content knowledge. This categorization will clarify the foundation on which each specific recommendation is made.

In this chapter, patterns, software architecture patterns, and security patterns were defined based on the study by Schumacher et al.[Sch+06]. Patterns are recognized as solutions to recurring problems within a specific context. As a subset of these, software architecture patterns offer tested solutions to common structural issues in software design while security patterns address frequently occurring security challenges. Following this, constraint-based recommender systems, which the SCAM project aims to develop, were introduced. These systems match user preferences and requirements with predefined rules and constraints. The chapter then focused on the knowledge needed for recommender systems to generate

suggestions, concluding that content knowledge is the most relevant for this thesis. This source of knowledge was categorized into three main types: item attributes, contextual knowledge, and domain knowledge. The next chapter will outline the factors that influence the reasoning process of recommender systems in suggesting appropriate patterns.

4 Concepts

This chapter explores the concepts of a catalog utilized in a constraint-based recommender system. First, it explains the hierarchical structure of patterns within the catalog, emphasizing how this hierarchy facilitates the system's ability to generate optimal suggestions in an organized manner. Furthermore, due to the significance of relationships among the patterns, a detailed examination of these inter-pattern connections is provided. This analysis is crucial for recommending appropriate solutions. Next, the chapter presents the security objectives that form the basis for classifying security patterns. Recognizing that these objectives alone are not sufficient for comprehensive classification – especially when multiple patterns address the same security objective – additional categorization criteria, the quality characteristics, are introduced. Lastly, technical and environmental factors, which play a crucial role in recommending security solution patterns, will be discussed.

4.1 Pattern Hierarchy

This section presents the proposed hierarchy of patterns in the catalog, distinguishing between conceptual and concrete types. It also explains how this hierarchy is to be applied within the knowledge base of the developed recommender system.

In the context of software, security patterns are essential tools for addressing common security challenges. However, for effective use in recommender systems, it is crucial to distinguish between different levels of abstraction in these patterns. Building on the concept of *abstract security patterns* introduced by Fernandez, Washizaki, and Yoshioka, we introduce a hierarchical approach to security patterns, differentiating between *abstract security solution patterns* (ASSPs) and security solution patterns (SSPs) [FWY08]. First, we define a security solution as follows [Shi07; Sta22]:

Definition. *A* security solution *is a constituent part of an architecture that integrates measures and techniques to achieve a security objective in order to increase the security of a system. It includes the necessary components and behavior to implement these security measures effectively within the system's architecture.*

ASSPs provide a conceptual framework for fundamental security solutions, similar to abstract classes in object-oriented programming, offering high-level guidance without specific implementation details.

We deduce the following definitions for the two pattern types:

Definition. *An* abstract security solution pattern *is a high-level pattern that represents a*

general security solution. It describes the core principles of what must be addressed to meet security objectives but is too abstract to be integrated into a software architecture.

ASSPs provide a good structure for a security solution and can be further concretized by security solution patterns, which represent implementable solutions derived from their abstract counterpart.

Definition. *A* security solution pattern *is a concrete, applicable pattern that is detailed enough to incorporate a specific security solution into a software architecture. Derived from the principles in an ASSP, an SSP offers the detailed building blocks to realize a security solution.*

It is important to note that a general security solution of an ASSP can be further detailed by other ASSPs, meaning they can have child ASSPs in the hierarchy. However, every path from any ASSP ultimately leads to an SSP, representing the final, concrete realization of the security solution. Figure 4.1 shows the meta-model in the UML 2 class definition. It illustrates the different types of security patterns and their relationships. The term "security pattern" is represented with dashed lines, indicating its role solely at the meta-level, encompassing both ASSP and SSP. The instantiated objects in the catalog are abstract security solution patterns and security solution patterns.

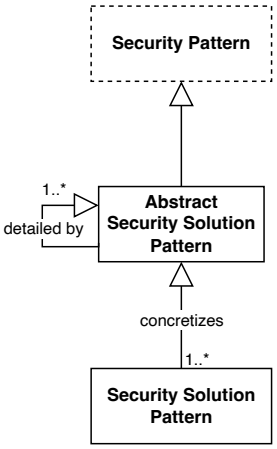

Figure 4.1: Meta-model of security patterns.

An example of the proposed hierarchy for authentication SPs is illustrated in Figure 4.2. At the root of the hierarchy tree is the ASSP *Authenticator*, representing the general concept of

an authentication mechanism. From there, the ASSPs *Distributed Authenticator* and *Centralized Authenticator* provide more specific security solutions, differing at a conceptual level. The *Distributed Authenticator* is further detailed by the ASSP *Credential*, as a specialized case of distributed authentication. Finally, the *Credential* pattern is concertized by the SSPs *X.509*, *SAML*, and *Token*, which appear as leaf nodes. For the *Centralized Authenticator*, the leaf SSPs are *Password*, *Biometric*, and *Card-based*.

The advantage of the reasoning of a catalog lies in its hierarchical structure, as it allows the recommender system to narrow down the appropriate patterns for recommendation by focusing on specific subtrees within the hierarchy. Without this structure, all patterns would exist on the same abstraction level, preventing the recommender system from making decisions based on the hierarchy and resulting in less efficient recommendations. When the recommender system selects a security pattern to suggest, only SSPs should be chosen, as they are the ones that can be directly implemented into the software architecture according to our definition of SSPs.

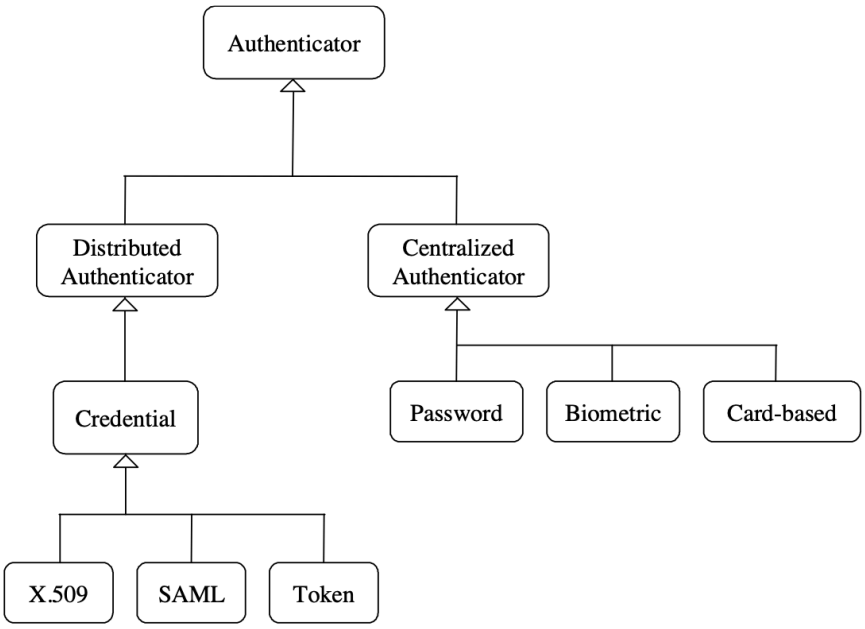

Figure 4.2: Security Pattern Hierarchy Example [FWY08].

Figure 4.3 comprehensively illustrates the hierarchical structure of patterns encompassed within a catalog realized by a knowledge base. The principal function of this catalog is to facilitate reasoning processes concerning the identification and selection of SSPs. At the granular level of this hierarchical structure, we find the leaf nodes of the SP knowledge base (SP_1), which embody the individual SSPs.

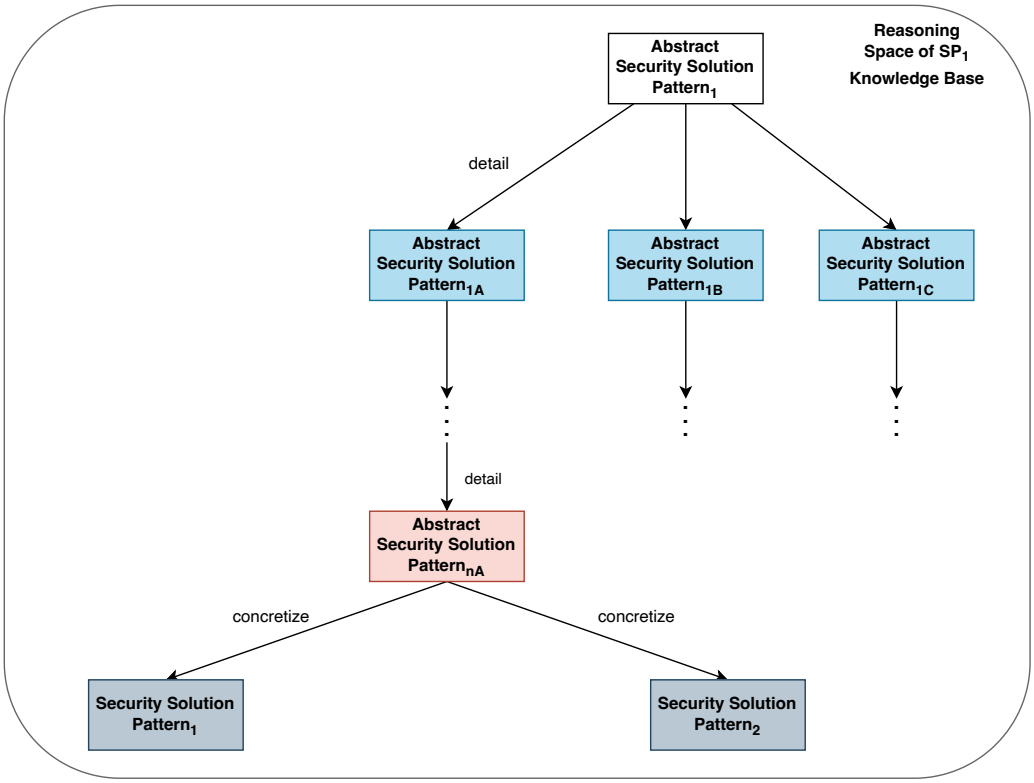

Figure 4.3: Reasoning Space of the Knowledge Base

4.2 Inter-pattern relationships

As seen in the previously introduced hierarchy, security patterns frequently interact and influence each other. Understanding these inter-pattern relationships is critical when recommending appropriate solutions since patterns are often used in combination [Was+21]. According to Yskout et al., these relationships are defined as follows [Ysk+06]:

Definition. *The inter-pattern relationships describe how the implementation of one pattern affects another, ranging from positive to negative interactions.*

Yskout et al. identified different types of inter-pattern relationships and categorized them into five distinct types, as shown in Table 4.1.

Relation	Explanation
Depends	Pattern A requires Pattern B to function correctly.
Benefits	Pattern A is improved or enhanced by Pattern B.
Alternative	Pattern A and Pattern B are interchangeable without affecting the system.
Impairs	Pattern A may be hindered by the implementation of Pattern B.
Conflicts	Pattern A and Pattern B cannot coexist without causing inconsistencies.

Table 4.1: Inter-pattern relationships categorized by [Ysk+06].

To enhance clarity, an example is provided for each type. These examples show how different security patterns interact and highlight the importance of understanding these relationships when selecting and recommending patterns.

1. Multi-factor Authentication (MFA) **depends** on the combination of two or more authentication patterns, as it integrates different methods (e.g., passwords, biometrics) to strengthen security [Das+17].

2. Single Sign-On (SSO) **benefits** from OpenID Connect (OIDC) by allowing users to authenticate through a third-party service, improving usability and security [Hos+18].

3. The Distributed Authenticator is an **alternative** to the Centralized Authenticator, as both patterns provide authentication mechanisms but follow different conceptual approaches [FWY08]. This demonstrates that inter-pattern relationships can exist between abstract security solution patterns and their more concrete implementations.

4. SSO **impairs** Fine-grained Access Control, because SSO provides uniform authentication, making it challenging to enforce different levels of access or require additional authentication without disrupting the user experience [Fug23]

5. Token-based Authentication **conflicts** with Session-based Authentication, due to the fact that tokens are designed to be stateless, thus the server does not need to store any information. Whereas Session-based Authentication relies on the server maintaining the session state [Bal17].

4.3 Security Objectives

As the name suggests, the goal of a security pattern is to enhance the security of a system. Consequently, we classify the security patterns in the catalog based on the specific security objectives they try to achieve. This section presents those security objectives.

We adopt the classification of security objectives from Yskout et al. [Ysk+06], which has been recognized as the most comprehensive by various studies, including those from [AZ12] and [CVC22]. The following security objectives are used to guide the selection of security solution patterns:

- **Confidentiality**. Ensures that sensitive data can only be accessed by the intended people. It involves securing data transmission and controlling access to stored data.
- **Integrity**. Ensures that data or resources have not been changed or tampered with. This includes keeping applications in a consistent state and using secure transmission and controlled access to prevent unauthorized changes.
- **Availability**. Ensures that the system is always available and responsive to its users.
- **Accountability**. Tracks actions performed on a system and links them to specific users.
- **Secure Data Transmission**. Protects data while it is being sent over a network, ensuring it stays private, is not altered, and verifies the sender/receiver.
- **Controlled Access**. Restricts access to resources so that only authorized users can perform certain actions. It relies on verifying user identity.
- **Identification**. The process of claiming and verifying a user's identity, usually done through authentication methods.
- **Non-repudiation**. Ensures that a user cannot deny performing an action, such as sending or receiving a message.
- **Anonymity**. Ensures that an individual cannot be identified within a group of users.
- **Privacy**. Gives individuals control over how their personal information is shared and ensures they are informed about how it is used.

As the descriptions of the security objectives demonstrate, certain objectives are inter-dependent, meaning they can rely on one another. For instance, ensuring *Confidentiality* requires implementing security patterns that also guarantee *Secure Data Transmission* and *Controlled Access*. Additionally, it is possible for security patterns to address multiple security objectives simultaneously.

4.4 Characteristics

Security objectives alone do not suffice for the recommendation of security solution patterns, especially when multiple patterns ensure the same security objective. To refine the recommendation and better align it with the specific needs of the software architect, we include quality characteristics in the classification of security patterns, similar to the approach taken by Yskout et al. [Ysk+06]. This section introduces these characteristics, which aid in the classification process.

We adopt the product quality model from the International Organization for Standardization [23], depicted in Figure 4.4. The model splits up the broad term product quality into nine quality attributes, which are defined as follows:

Definition. *A* quality attribute *is a property of a software system that determines its overall quality and can be measured or evaluated against predefined criteria [23].*

A quality attribute consists of quality factors that describe the properties of a system in greater detail and can be defined as follows:

Definition. *A* quality factor *is a subcharacteristic of a quality attribute that contributes to the overall quality of a software system and can be measured or evaluated against predefined criteria [23].*

Based on Figure 4.4 a quality attribute would be "compatibility", which consists of the quality factors: "co-existence" and "interoperability". The term quality characteristic encompasses both quality attributes and quality factors.

Quality characteristics of a software system may be positively or negatively influenced by security patterns. For instance, consider the quality factor *operability*, which belongs to the quality attribute *interaction capability*. In the context of authentication SSPs, the number of steps and waiting time required during user authentication decreases operability. Hence, password-based authentication, with fewer steps for authenticating a user, typically has a lesser negative impact on operability compared to multi-factor authentication, which consists of multiple authentication methods. Therefore, if operability is prioritized, password-based authentication should be preferred.

Figure 4.4 depicts a catalog of quality characteristics from the standardization institute ISO\IEC [23].

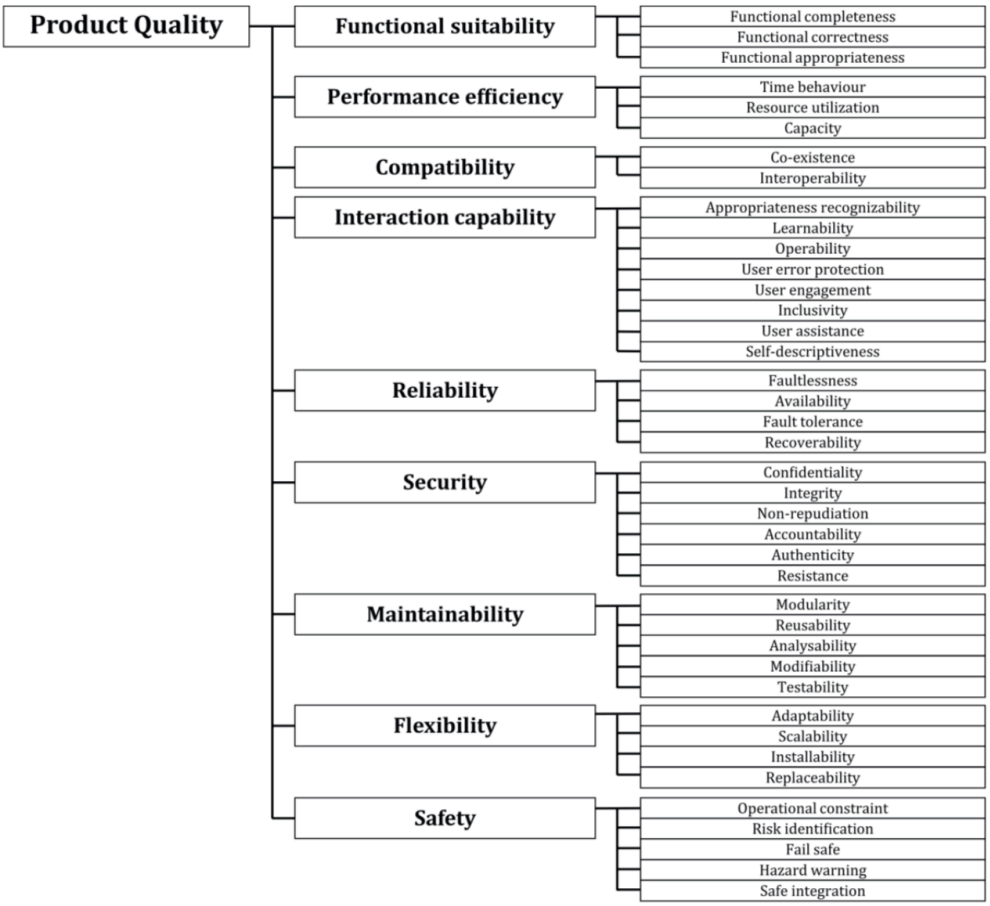

Figure 4.4: Quality model by ISO\IEC [23]

4.5 Technical and Environmental Factors

Another essential aspect in recommending a suitable security solution pattern is the consideration of technical and environmental factors, which provide crucial information helping the recommendation process. This section presents these factors.

In this thesis, environmental factors are defined as follows:

Definition. Environmental Factors *are external conditions and constraints that influence the application and deployment of a security pattern in a software product.*

The following environmental factors have been extracted for our knowledge base:

- **Industry Sector**: A broad category of businesses and organizations engaged in similar types of economic activities.

- **Legal and Regulatory Requirements**: Laws, regulations, and other rule sets applicable to a specific jurisdiction.

For example, in the FinTech industry, software products need to adhere to particular regulations concerning security measures. The Payment Services Directive 2 (PSD2), for instance, requires Fintech companies in the European Economic Area to incorporate multi-factor authentication for online transactions and accessing personal data [Wan21]. This illustrates that some factors not only favor specific SSPs but also mandate some while excluding others during the recommendation process. Using password-based authentication by itself would not be a suitable recommendation, as it does not meet regulation standards. However, combining password-based authentication with a one-time password would satisfy the multi-factor authentication criterion. Although legal and regulatory requirements are often tied to specific industry sectors, they also vary significantly by location, necessitating their treatment as a distinct factor separate from the industry sector itself.

In addition to environmental considerations, technical factors, which provide a more detailed description of the software product, also play a significant role. Technical factors are defined as follows:

Definition. Technical Factors *are platform-specific and technical considerations that influence the selection and integration of a security pattern in a software product.*

The following technical factors are included in our catalog:

- **Application Domain**: The specific field within an industry sector that focuses on a particular software solution.
- **Architecture Pattern**: The high-level structure of a software system, defining its components, interactions, and overall organization.
- **Technology Type**: The underlying technology of the software product.

An example of an application domain would be IoT or e-commerce. For architecture patterns, examples include microservices and fat-client, while for technology types, examples are embedded systems, web applications, and mobile applications.

Incorporating these factors allows the recommender system to narrow down suitable SSPs for specific scenarios. For example, if an architect searches for an authentication solution to be applied in a software product based on a microservices architecture, knowledge of the architecture type suggests that token-based authentication methods are preferred due to their stateless nature.

4.6 Integration into a Recommender System

In this section, we categorize the identified factors from this chapter in alignment with the knowledge sources introduced in Chapter 3. Additionally, we provide a machine-readable

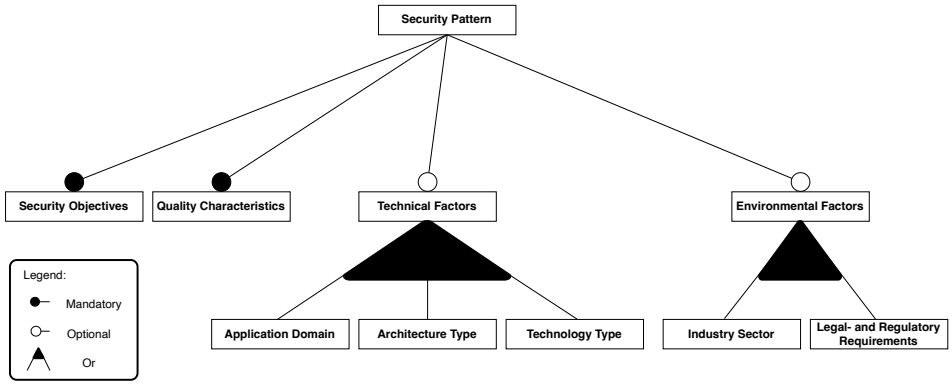

Figure 4.5: Classification factors for security patterns

encoding for these factors to ensure that the recommender system can effectively interpret and apply them. Figure 4.5 provides a feature diagram that summarizes the key factors used in the classification of security patterns. While the hierarchy of patterns is not explicitly depicted in the diagram, its main function is to offer a structured framework for organizing and navigating through security patterns, rather than being a direct attribute for filtering them. This hierarchy can be efficiently implemented by storing it as a tree structure within a database, enabling more intuitive and systematic access to the security patterns.

As one can see, security objectives and quality characteristics must be specified for every security pattern, as they represent the core objectives the pattern is designed to achieve. On the other hand, technical and environmental factors should be alternatively included if possible. Providing this additional context helps the recommender system make more informed decisions by incorporating broader insights.

Knowledge Sources. We propose the following alignment of the identified factors with the knowledge sources outlined in Chapter 3.

- *Security Objectives* and *Quality Characteristics* are classified as *Item Attributes*, as they are used to describe properties and classify security patterns within the knowledge base.
- *Industry Sector* falls under *Contextual Knowledge*, as it relates to external circumstances and the environment of the user.
- *Architecture Type* and *Technology Type* should be categorized under *Feature Ontology*, as they describe the structure and attributes of the system.
- *Legal and Regulatory Requirements* belong to *Domain Constraints* since they represent the rules and limitations governing what is possible within a specific domain.

Encoding. To ensure that security patterns are machine-interpretable, we must encode their effects on various factors in a structured format. Unlike traditional human-readable

patterns, this encoding will allow the recommender system to evaluate how patterns influence different attributes and make informed suggestions.

We propose two encoding methods: one for security objectives and quality characteristics, and another for technical and environmental factors. This differentiation is necessary due to the distinct influence these factors have on the recommendation process.

Security Objectives and Characteristics. Security objectives and quality characteristics can be affected both positively and negatively by security patterns. Therefore, we propose using a simple ordinal scale to classify the extent of this influence. A five-step scale is recommended to capture the impact of a specific security pattern on these factors, as follows:

- - - (Strongly Negative)
- - (Negative)
- **0** (Neutral)
- + (Positive)
- ++ (Strongly Positive)

This scale will provide the recommender system with a clear assessment of how a security pattern enhances or diminishes a particular security objective or quality characteristic. For example, a security pattern might strongly improve identification (++) but have a negative impact (-) on operability.

Technical and Environmental Factors.

When considering technical and environmental factors, we classify the relationship between security patterns and these factors into four distinct categories:

- **Suitable**: The pattern is suitable with the factor and enhances the likelihood of being recommended.
- **Not Suitable**: The pattern is not suitable with the factor but may still be recommended under certain circumstances.
- **Mandatory**: The pattern is required in the presence of the factor, meaning it must be included in the recommendation.
- **Prohibited**: The pattern is explicitly forbidden in the context of the factor, and it must be excluded from any recommendations.

"Suitable" and "not suitable" are considered soft classifications, as they influence the system's preferences without enforcing strict constraints. For instance, an SSP marked as "suitable" increases the likelihood that the pattern will be recommended, while "not suitable" decreases that likelihood, though it does not exclude the possibility entirely. In contrast, "mandatory" and "prohibited" are hard classifications that directly govern the recommender

system's decision-making process. If a pattern is labeled as "prohibited," the system will not recommend it under any circumstances.

For example, password-based authentication could be prohibited (p) in a highly regulated industry like FinTech, meaning it will never be suggested. On the other hand, patterns marked as "mandatory" are essential and must be recommended in the presence of certain factors. For instance, multi-factor authentication might be mandatory (m) for systems handling sensitive financial data, ensuring it is always included in the final recommendation.

4.7 Summary

In summary, this chapter has introduced the fundamental concept required for selecting suitable security solution patterns.

First, we outlined a hierarchical structure of patterns within a knowledge base designed for a recommender system, distinguishing between abstract security solution patterns that offer solutions at a conceptual level and concrete patterns that are directly implementable. The relationships among these patterns were defined according to the framework presented by Yskout et al. [Ysk+06], which categorizes them into five types: depends, benefits, alternative, impairs, and conflicts. Furthermore, we incorporated security objectives such as confidentiality and integrity as attributes for items within the knowledge base. These attributes represent the primary security objectives that each pattern aims to address, aiding in the fulfillment of the security requirements for software architecture. However, these security objectives alone were insufficient for comprehensive classification, necessitating the inclusion of quality characteristics such as operability and scalability to further refine the selection criteria. Additionally, we integrated contextual and domain-specific knowledge, including technical and environmental factors, to offer a thorough understanding of the application context for each pattern.

Following the presentation of the theoretical concept of the catalog, we will demonstrate the system's practical application through various scenarios in the next chapter.

5 Demonstration

In this chapter, we demonstrate the application of the concept introduced in Chapter 4 through a step-by-step scenario. The chapter begins with an explanation of how the recommending process functions, followed by an overview of the scenario's initial state. The scenario is then progressively developed, with each step introducing additional contextual information, thereby increasing the scenario's complexity. Each step illustrates how these additional factors contribute to recommending the most appropriate security solution patterns. For simplicity, the demonstration focuses on the following three SSPs that are directly applicable to the software architecture and achieve the security objective "Identification":

- *Password-based authentication* (PBA): A user proves their identity by providing a valid identifier, such as a username or email address, along with the corresponding password [DYJ22].
- *Token-based Authentication based on OpenID Connect* (OIDC): A user proves their identity by logging in through an external service, like Google or Facebook, where they already have an account. The process starts with the system redirecting the user to the external service to log in. Once the user enters their credentials (like their Google password) and is verified, the external service sends a code back to the system. The system then exchanges this code for a token, which contains the user's information. The system checks this token to confirm the user's identity, similar to how a password would be checked, but the external service manages the actual login process [Okt].
- *Multifactor-based Authentication* (MFA): A user verifies their identity by following the same procedure as in PBA. If the authentication is successful, they receive an SMS code on their mobile device, which they must then enter into the system to complete the process [Das+17].

As mentioned in Chapter 4, the catalog does not include design patterns, which represent the realization of an SSP within a software architecture. Its sole purpose is to facilitate the selection process for recommending the appropriate SSP.

The proposed recommendation process for SSPs starts when a user provides a security objective as an input. The system takes this along with the user's preferences for quality, technical, and environmental factors. These preferences help the recommender system make better decisions by focusing on the most relevant aspects.

For quality characteristics, the user prioritizes both the characteristics and the security objectives based on their importance. This is done by assigning a value from 1 (least important) to 5 (most important), which we refer to as the "weight." In the knowledge base, each

SSP is rated for its compatibility with these factors. These ratings follow the five-step scale introduced in Section 4.6, but are converted into numerical values for calculation purposes, as shown in Table 5.1.

five-step scale	–	-	0	+	++
corresponding numeric values	1	2	3	4	5

Table 5.1: Mapping of the encoding scale into numeric values.

To compute a recommendation score, the system multiplies the user's weight for each factor with the corresponding rating of the SSP in the knowledge base. These weighted scores are then summed up to give the final ranking score for each SSP, as shown in 5.1.

$$\text{ranking score}_{\text{SSP}} = \sum \text{weight}_{\text{quality characteristic}} \cdot \text{rating}_{\text{quality characteristic, SSP}} \tag{5.1}$$

For technical and environmental factors, the system relies on the predefined classification outlined in Section 4.6. SSPs marked as "prohibited" are excluded from the recommendation results, while those labeled as "mandatory" are given top priority in the recommendation list. Following these are SSPs classified as "suitable," with those deemed "not suitable" positioned at the bottom of the list.

The scenario to be demonstrated is as follows:

Step 1: Initial Setup and Simple Scenario. In the first step, the recommender system will suggest SSPs by considering only the *security objectives*.

Scenario. Joe is developing a website for selling honey. He needs an identification method for customers to log in and check their order status. The following SSP classifications are stored in the knowledge base of the recommender system and are available for Joe to use:

1. **PBA**
   Security Objective: Identification, rated with *"0"* (3)
   Reason: PBA offers a basic level of identification and is only as secure as the password strength.

2. **OIDC**
   Security Objective: Identification, rated with *"="* (4)
   Reason: Tokens are securely generated and validated, offering a reduced risk of impersonation compared to passwords due to features like expiration.

3. **MFA**
   Security Objective: Identification, rated with *"++"* (5)
   Reason: MFA provides a higher level of identification by combining multiple authentication factors.

Recommendation. Based on the classification of SSPs with respect to identification, the recommender system ranks the SSPs as follows: MFA, OIDC, and PBA. This ranking is based solely on how well each SSP improves the identification objective. However, this approach ignores Joe's specific needs. As his website handles low-sensitivity data, implementing MFA may frustrate customers due to the additional steps involved in the login process. Therefore, while MFA might offer the highest security, its usability is not suited for Joe's scenario.

The recommendation, considering only the security objective, would be:

1. MFA
2. OIDC
3. PBA

Step 2: Including Quality Characteristics. In this step, we introduce *quality characteristics* such as operability and scalability. Operability refers to the ease and speed of the user identification process, while scalability relates to the system's capacity to handle increasing user requests without performance degradation.

Scenario. Joe's website is thriving and has grown a lot in customer size. Now he is searching for a new identification method, as customers keep complaining about the tedious MFA process they have to complete to just check their orders. He now prioritizes operability and scalability over security, as the data sensitivity remains low. Joe ranks his preferences as follows: operability, scalability, and identification. The SSP classifications are expanded as follows:

1. **PBA**
   Quality Characteristic: Operability, rated with "+" (3)
   Reason: PBA is simple and familiar to most users.

   Quality Characteristic: Scalability, rated with "-" (2)
   Reason: While suitable for smaller systems, PBA faces challenges at scale, such as storage overhead.

2. **OIDC**
   Quality Characteristic: Operability, rated with "+" (3)
   Reason: OIDC allows users to log in using trusted third-party providers, reducing the need for new account creation.

   Quality Characteristic: Scalability, rated with "++" (5)
   Reason: Stateless tokens improve scalability, as they eliminate server-side storage dependencies.

3. **MFA**
   Quality Characteristic: Operability, rated with "- -" (1)
   Reason: MFA adds extra steps and waiting time for users.

   Quality Characteristic: Scalability, rated with "-" (2)
   Reason: Managing passwords and delivering SMS codes to a large user base introduces operational overhead.

Recommendation. The recommender system computes the total rating for each SSP based on Joe's preferences and the weighted sum of the factors:

$$\begin{aligned}
\text{rating}_{\text{PBA}} &= \text{weight}_{\text{operability}} \times \text{rating}_{\text{operability, PBA}} \\
&\quad + \text{weight}_{\text{scalability}} \times \text{rating}_{\text{scalability, PBA}} \\
&\quad + \text{weight}_{\text{identification}} \times \text{rating}_{\text{identification, PBA}} \\
&= 5 \times 3 + 4 \times 2 + 3 \times 3 \\
&= 32
\end{aligned}$$

$$\begin{aligned}
\text{rating}_{\text{OIDC}} &= \text{weight}_{\text{operability}} \times \text{rating}_{\text{operability, OIDC}} \\
&\quad + \text{weight}_{\text{scalability}} \times \text{rating}_{\text{scalability, OIDC}} \\
&\quad + \text{weight}_{\text{identification}} \times \text{rating}_{\text{identification, OIDC}} \\
&= 5 \times 3 + 4 \times 5 + 3 \times 4 \\
&= 47
\end{aligned}$$

$$\begin{aligned}
\text{rating}_{\text{MFA}} &= \text{weight}_{\text{operability}} \times \text{rating}_{\text{operability, MFA}} \\
&\quad + \text{weight}_{\text{scalability}} \times \text{rating}_{\text{scalability, MFA}} \\
&\quad + \text{weight}_{\text{identification}} \times \text{rating}_{\text{identification, MFA}} \\
&= 5 \times 1 + 4 \times 1 + 3 \times 2 \\
&= 15
\end{aligned}$$

OIDC has the highest score (47), followed by PBA (32) and MFA (15). Hence, the system recommends the SSPs in the following order:

1. OIDC
2. PBA
3. MFA

Step 3: Including Technical and Environmental Factors In this step, we incorporate technical and environmental factors, such as compliance with the Payment Services Directive 2 (PSD2) regulation. This regulation mandates strong customer authentication in the form of multifactor-based authentication for the user.

Scenario. After Joe's website grew and introduced various new features, he transformed the underlying software architecture into a microservice-based design to increase maintainability and scalability. He also introduced a cryptocurrency, HoneyCoin, primarily used for purchasing items on his website. HoneyCoin can also be traded on cryptocurrency markets. After being notified by his lawyer, Joe realizes that with the introduction of a cryptocurrency, he has to comply with the PSD2 regulation, which affects the identification methods on his website.

Joe adds this new regulatory requirement to the recommender system and indicates that his website now operates in a microservice architecture. The preferences remain the same as in Step 2, with operability, scalability, and identification being prioritized. The following classifications are added:

1. **PBA**
   Environmental Factor: Legal and Regulatory Requirements, classified as "prohibited" under PSD2
   Reason: PBA alone does not meet the PSD2 mandate for Strong Customer Authentication (SCA).

2. **OIDC**
   Tecnical Factor: Architecture Type, classified as "suitable" for microservices
   Reason: Stateless tokens are ideal for microservices, as they eliminate the need for repeated authentication across services.

3. **MFA**
   Environmental Factor: Legal and Regulatory Requirements, classified as "mandatory" under PSD2
   Reason: PSD2 mandates the use of MFA to meet SCA requirements.

Recommendation. The rating calculation for each SSP, based on the security objective identification and the quality characteristics of operability and scalability, remains unchanged. However, the recommender system now incorporates additional information about legal and regulatory requirements, as well as technical factors. Specifically, the system recognizes that the user must comply with the PSD2 regulation. Under this regulation, MFA is classified as mandatory, while PBA is prohibited. Although MFA has the lowest overall rating of 15 compared to PBA and OIDC, it is recommended due to the regulatory mandate. Despite the system's awareness that Joe's microservice architecture would be well-suited for OIDC, MFA remains the top recommendation, as compliance with PSD2 takes precedence over architectural considerations. Thus, the only resulting recommendation is:

1. MFA

In this scenario, the recommender system identifies MFA as the only security solution pattern in its knowledge base suitable for the given circumstances and, as a result recommends it.

In this chapter, we demonstrated how the constraint-based recommender system for security solution patterns functions in a practical scenario. Through a step-by-step approach, we illustrated how the system evolves from a basic security objective-based recommendation to a more complex decision-making process that incorporates quality, technical, and environmental factors.

By providing only a security objective as an input, the system initially targeted only the immediate security issue the user was facing and was unable to account for specific needs, as these are not included in the input. However, the additional steps demonstrated that incorporating further inputs such as quality characteristics, technical aspects, and environmental considerations significantly improves the recommendations. This improvement arises because the system can tailor its decisions to the specific needs of the user. The system's effective decision-making is achieved through the developed conceptual framework and its encoding, which ensures practical usability.

Moreover, the simple mathematical approach used in calculating recommendation scores for each security solution pattern, based on a weighted sum of factors, provided a straightforward yet efficient method to guide the system's decision-making. This process ensured that the recommendations aligned with the calculated scores, enhancing the overall effectiveness of the recommender system.

6 Discussion

This chapter provides a detailed discussion of the key findings of this thesis by addressing the research question mentioned in Chapter 1. Additionally, it critically evaluates the methods and outcomes, reflecting on both the strengths and critique points of the work.

RQ1: How can security solution patterns be effectively organized and classified to be realized by knowledge bases to effectively reason about them?

To organize security solution patterns effectively, we have developed a hierarchical structure that distinguishes between them and abstract security solution patterns. ASSPs describe the core principles of a security solution, while SSPs offer detailed enough guidelines on how to integrate these principles in software architecture. This hierarchical structure allows us to group patterns based on the similarities of the security solutions they provide, resulting in a systematic framework for managing security patterns. Additionally, by accounting for relationships between patterns, we gain insights into their structural connections and dependencies, further enriching the catalog.

In terms of classification, we employed multiple factors to tag patterns with detailed and contextual information. These factors include security objectives and quality characteristics, which define the key attributes of each pattern. In addition, technical and environmental factors provide important contextual information regarding the pattern's applicability and suitability for particular environments or technical situations. This multi-dimensional classification ensures that patterns are both organized in a structured manner and described comprehensively for practical implementation.

For a constrained-based recommender system to effectively use the catalog, the organization and classification of security patterns must be machine-interpretable. The hierarchical organization of patterns, which can be realized as a tree structure in a knowledge base, is particularly suitable for this purpose. In this tree, relationships between patterns – such as which ASSP concretizes into which SSP – are clearly defined, along with the inter-pattern relationships being dependencies and associations. This structured representation enables the recommender system to efficiently understand and manage the relationships between patterns.

Additionally, regarding classification, we adopt a numerical five-step scale to evaluate and quantify the attributes of each pattern. This approach allows the recommender system to reason systematically about patterns based on security objectives, quality characteristics, and other relevant factors. By representing these attributes numerically, the system can

better analyze and compare patterns to meet specific security needs. Additionally, we classify technical and environmental factors into four levels of applicability and suitability. This categorization helps the system identify the most relevant patterns for a given scenario by analyzing how well a pattern aligns with the technical context or industry regulations. This multi-layered approach ensures that recommendations are not only technically robust but also contextually appropriate.

The developed conception of a security solution pattern catalog for constraint-based recommender systems in this thesis offers several advantages but also contains some critique points that should be taken into account.

On the positive side, the developed conception of a catalog is effective in enabling a recommender system to reason about which SSP should be recommended. Especially, incorporating security objectives and quality characteristics as key attributes provides valuable indicators for both identifying a pattern's intended goals and evaluating its broader impact on the system. These attributes guide the recommender system in selecting patterns that align with specific security objectives while considering the possible broader effects on the system.

Despite these strengths, there are some critical points in the current approach.

One area for improvement lies in the use of the five-step scale for classifying pattern attributes. Although this scale offers a basic framework for classification, it may lack the granularity required for high precision in certain scenarios. For example, in cases where a security objective is the only input provided to the recommender system and more than five patterns are relevant to that objective, the scale may not be sufficient to differentiate effectively between them. This lack of precision can lead to less effective recommendations because the system might have trouble distinguishing between different patterns.

Similarly, the use of four levels for classifying technical and environmental factors may be overly restrictive. While a pattern might be considered suitable for a given scenario, another pattern could be a better fit based on the specific user requirements. By introducing a more granular scale – possibly expanding the range of levels – the recommender system could provide more refined and context-sensitive recommendations. This would allow the system to prioritize patterns based on a more precise understanding of technical and environmental nuances, resulting in more tailored and effective security solutions.

In conclusion, while the hierarchical organization and multi-dimensional classification of security patterns provide a strong foundation for reasoning in constraint-based recommender systems, there is still room for improvement in the precision and granularity of the classification system. Enhancing the scales for pattern attributes and environmental factors would enable the recommender system to deliver more nuanced, context-aware recommendations, potentially increasing its effectiveness in diverse scenarios. These adjustments would refine the system's ability to differentiate between similar patterns and ensure that recommendations are both precise and highly relevant to the user's specific security requirements.

7 Conclusion and Future Work

This chapter presents the conclusion of this thesis and future work.

7.1 Conclusion

In this thesis, the conception of a security solution pattern catalog for constraint-based recommender systems was developed.

The work was structured to achieve the overall goal of creating a conception of a catalog that can be realized by knowledge bases, tailored for secure software architecture, and optimized for use in a recommender system.

The work began with a light systematic literature review, which provided valuable insights into existing research on security patterns, their classification, and their application in secure software design. These findings laid the foundation for the development of the conception of the catalog. Moreover, we introduced a refined definition of a security solution pattern, specifically tailored for use in a recommender system for secure software architecture. In addition, abstract security solution patterns were incorporated into a hierarchical structure in order to group related security solutions, making them easier to navigate and manage within the catalog. Building on this structure, we adopted various classification factors to enrich the catalog. These factors not only define the attributes of security patterns but also offer contextual information about the applicability and suitability of each pattern. This multi-dimensional classification was important for ensuring the catalog could support practical and relevant security recommendations. To ensure that the catalog is machine-interpretable, we developed an encoding that allows the recommender system to process and interpret the classification factors. This encoding is essential for the integration of the catalog into a recommender system. Additionally, we demonstrated the practical application of the catalog through a simplified example within a recommender system, illustrating the functionality and potential of the developed conception. Finally, we addressed the research questions outlined at the beginning of this thesis and discussed the capabilities and performance of the catalog, confirming that the goals of the thesis were successfully achieved.

The adopted concepts present a usable catalog of security solution patterns, tailored for implementation within a recommender system, and contribute toward the advancement of secure software architecture design.

7.2 Future Work

The adopted concepts in this thesis lay the groundwork for a catalog of security solution patterns. However, several key steps remain for its full realization and optimization in practical

applications.

One of the primary areas for future work is the **realization of the catalog by a knowledge base**, using the framework and classifications outlined in this thesis. This would involve building the actual infrastructure to support the catalog of security solution patterns, ensuring that it is both scalable and adaptable for use in a recommender system.

Once the knowledge base is established, the next step is to **populate it with security patterns**. This process will require identification and organization of a wide range of security patterns from various domains to ensure comprehensive coverage of security solutions. The quality and relevance of the patterns chosen will be critical for the practical utility of the knowledge base.

Another crucial aspect to address is the **improvement of the encoding scale** for the patterns. The current encoding system should be refined to ensure it is neither too granular nor too broad. It needs to strike a balance, providing enough detail to differentiate between patterns without becoming unnecessarily complex for the system.

In addition to refining the encoding scale, there is a need to adopt a method for **evaluating the quality of the patterns** included in the knowledge base. Each pattern should meet a certain quality standard to ensure its effectiveness and reliability in practical applications. One possible approach is to adopt a **scoring method** for patterns, as introduced by Heyman et al., who proposed a scoring system based on key elements of security patterns [Hey+07]. Such a method would allow for the systematic evaluation of each pattern's quality and relevance, improving the overall robustness of the knowledge base.

In conclusion, although the foundational concepts have been laid, the actual implementation, improvement, and evaluation of the catalog remain significant areas for future development.

Bibliography

[23] *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Tech. rep. ISO/IEC 25010. 2023 (cit. on pp. 21, 22).

[Ale18] C. Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 2018 (cit. on p. 9).

[Asl+23] Ö. Aslan et al. "A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions." In: *Electronics* 12.6 (2023), p. 1333 (cit. on p. 1).

[AZ12] A. K. Alvi and M. Zulkernine. "A comparative study of software security pattern classifications." In: *2012 Seventh International Conference on Availability, Reliability and Security*. IEEE. 2012, pp. 582–589 (cit. on p. 20).

[Bal17] Y. Balaj. "Token-based vs session-based authentication: A survey." In: *no. September* (2017), pp. 1–6 (cit. on p. 19).

[CVC22] A. Cordeiro, A. Vasconcelos, and M. Correia. "A Catalog of Security Patterns." In: *Proceedings of 29th Conference on Pattern Languages of Programs, PLoP*. 2022 (cit. on pp. 6, 8, 20).

[Das+17] D. Dasgupta et al. "Multi-factor authentication: more secure approach towards authenticating individuals." In: *Advances in User Authentication* (2017), pp. 185–233 (cit. on pp. 19, 27).

[DYJ22] A. van Den Berghe, K. Yskout, and W. Joosen. "A reimagined catalogue of software security patterns." In: *Proceedings of the 3rd international workshop on engineering and cybersecurity of critical systems*. 2022, pp. 25–32 (cit. on pp. 8, 27).

[FB08] A. Felfernig and R. Burke. "Constraint-based recommender systems: technologies and research issues." In: *Proceedings of the 10th international conference on Electronic commerce*. 2008, pp. 1–10 (cit. on pp. 10–12).

[Fug23] S. Fugkeaw. "Achieving Decentralized and Dynamic SSO-Identity Access Management System for Multi-Application Outsourced in Cloud." In: *IEEE Access* 11 (2023), pp. 25480–25491. URL: `https://api.semanticscholar.org/CorpusID:257446811` (cit. on p. 19).

[FWY08] E. B. Fernandez, H. Washizaki, and N. Yoshioka. "Abstract Security Patterns." In: *Proceedings of the 15th Conference on Pattern Languages of Programs*. PLOP '08: Pattern Languages of Programs. Nashville Tennessee USA: ACM, Oct. 18, 2008, pp. 1–2. ISBN: 978-1-60558-151-4. DOI: 10.1145/1753196.1753201. URL: https://dl.acm.org/doi/10.1145/1753196.1753201 (visited on 08/06/2024) (cit. on pp. 15, 17, 19).

[Gam+95] E. Gamma et al. *Design Patterns*. Vol. 47. Addison Wesley Professional Computing Series February. 1995, pp. 1–429 (cit. on p. 10).

[Hey+07] T. Heyman et al. "An Analysis of the Security Patterns Landscape." In: *Third International Workshop on Software Engineering for Secure Systems (SESS'07: ICSE Workshops 2007)*. 2007, pp. 3–3. DOI: 10.1109/SESS.2007.4 (cit. on p. 36).

[Hos+18] N. Hossain et al. "OAuth-SSO: A Framework to Secure the OAuth-Based SSO Service for Packaged Web Applications." In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2018, pp. 1575–1578. DOI: 10.1109/TrustCom/BigDataSE.2018.00227 (cit. on p. 19).

[JR20] A. J. Jafari and A. Rasoolzadegan. "Security patterns: A systematic mapping study." In: *Journal of Computer Languages* 56 (2020), p. 100938 (cit. on p. 5).

[Okt] Okta. *What is OpenID Connect (OIDC)?* Accessed: 27.09.2024, 05:24. URL: https://auth0.com/intro-to-iam/what-is-openid-connect-oidc (cit. on p. 27).

[Rat+19] A. Rath et al. "Security pattern for cloud SaaS: From system and data security to privacy case study in AWS and Azure." In: *Computers* 8.2 (2019), p. 34 (cit. on p. 7).

[Ric18] C. Richardson. *Microservices patterns: with examples in Java*. Simon and Schuster, 2018. Chap. 2.1.3 (cit. on p. 10).

[RJM20] P. Rana, N. Jain, and U. Mittal. "An Introduction to Basic Concepts on Recommender Systems." In: *Recommender System with Machine Learning and Artificial Intelligence* (2020). URL: https://api.semanticscholar.org/CorpusID:225762283 (cit. on p. 10).

[RK20] S. T. Rotim and V. Komnenić. "Cybersecurity Talent Shortage." In: 2020. URL: https://api.semanticscholar.org/CorpusID:234655683 (cit. on p. 1).

[Sab24] A. R. Sabau. "A Guided Modeling Approach for Secure System Design." In: *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. 2024, pp. 105–110. DOI: 10.1109/ICSA-C63560.2024.00026 (cit. on pp. 1, 11).

[Sch+06] M. Schumacher et al. *Security Patterns: Integrating Security and Systems Engineering*. ProQuest Ebook Central. John Wiley & Sons, Incorporated, 2006. URL: https://ebookcentral.proquest.com/lib/rwthaachen-ebooks/detail.action?docID=257711 (cit. on pp. 9, 10, 12).

[Sch03] M. Schumacher. *Security engineering with patterns: origins, theoretical models, and new applications*. Vol. 2754. Springer Science & Business Media, 2003 (cit. on p. 9).

[Shi07] R. Shirey. *Internet Security Glossary, Version 2*. Tech. rep. RFC 4949. Internet Engineering Task Force (IETF), Aug. 2007. URL: https://www.rfc-editor.org/rfc/rfc4949 (cit. on p. 15).

[Sta22] W. Stallings. *Cryptography and Network Security: Principles and Practice*. 8th. Hoboken, NJ: Pearson, 2022. Chap. 1 (cit. on p. 15).

[Wan21] R. Wandhöfer. "TITLE IV 'RIGHTS AND OBLIGATIONS IN RELATION TO THE PROVISION AND USE OF PAYMENT SERVICES', CHAPTER 5 'OPERATIONAL AND SECURITY RISKS AND AUTHENTICATION' (ARTS 95-98)." In: *The Payment Services Directive II* (2021). URL: https://api.semanticscholar.org/CorpusID:245356997 (cit. on p. 23).

[Was+21] H. Washizaki et al. "Systematic literature review of security pattern research." In: *Information* 12.1 (2021), p. 36 (cit. on pp. 5, 6, 19).

[Ysk+06] K. Yskout et al. "A system of security patterns." In: *CW Reports* (2006) (cit. on pp. 19–21, 26).