Institute of Architecture of Application Systems

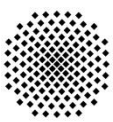
Pattern Views: Concept and Tooling for Interconnected Pattern Languages

Manuela Weigold, Johanna Barzen, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, Karoline Wild

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{weigold, barzen, breitenbuecher, falkenthal, leymann, wild}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Weigold2020_PatternViews,
  author    = {Weigold, Manuela and Barzen, Johanna and Breitenb{\"u}cher, Uwe
               and Falkenthal, Michael and Leymann, Frank and Wild, Karoline},
  title     = {{Pattern Views: Concept and Tooling of Interconnected Pattern
               Languages}},
  booktitle = {Proceedings of the 14th Symposium and Summer School on
               Service-Oriented Computing (SummerSOC 2020)},
  pages     = {86--103},
  publisher = {Springer International Publishing},
  month     = dec,
  year      = 2020,
  doi       = {10.1007/978-3-030-64846-6_6}
}
```

University of Stuttgart
Germany

Pattern Views: Concept and Tooling for Interconnected Pattern Languages

Manuela Weigold[0000−0002−4554−260X], Johanna Barzen[0000−0001−8397−7973], Uwe Breitenbücher[0000−0002−8816−5541], Michael Falkenthal[0000−0001−7802−1395], Frank Leymann[0000−0002−9123−259X], and Karoline Wild[0000−0001−7803−6386]

Institute of Architecture of Application Systems,
University of Stuttgart, Universitätsstrasse 38, Stuttgart, Germany
[firstname.lastname]@iaas.uni-stuttgart.de

Abstract. Patterns describe proven solutions for recurring problems. Typically, patterns in a particular domain are interrelated and can be organized in pattern languages. As real-world problems often require to combine patterns of multiple domains, different pattern languages have to be considered to address these problems. However, cross-domain knowledge about how patterns of different pattern languages relate to each other is mostly hidden in individual pattern descriptions or not documented at all. This makes it difficult to identify relevant patterns across pattern languages. To address this challenge, we introduce pattern views (i) to document the set of related patterns for a certain problem across pattern languages and (ii) to make this knowledge about combinable patterns available to others. To demonstrate the practical feasibility of pattern views, we integrated support for the concept into our pattern toolchain, the Pattern Atlas.

Keywords: Patterns · Pattern Languages · Cross-Language Relations · Pattern Language Composition · Pattern Graph.

1 Introduction

Patterns describe proven solutions for recurring problems. After the first patterns were published in the domain of city and building architecture by Alexander et al. [1], the concept of patterns has been adopted in various other fields. Especially in software and information technology, publishing patterns has become a popular way to convey expert knowledge in different domains, e.g., object-oriented programming [2], enterprise application architecture [3], messaging [4], or security [5]. Since patterns can often be used in combination or offer alternative solutions, the relations between patterns are essential for identifying all relevant patterns and are therefore often documented. For example, the pattern *Public Cloud* of the cloud computing pattern language by Fehling et al. [6] describes how cloud providers can offer IT resources to a large customer group. It further refers to patterns that describe the different service models for offering resources, e.g., as *Infrastructure as a Service* (IaaS) [6]. When using the *Public Cloud* pattern, those related patterns should also be considered, because they solve

related problems. In conjunction with the relations between them, patterns can be organized in pattern languages [1]. As a result, a pattern language describes how patterns work together to solve broader problems in a particular domain [7].

However, real-world problems often require the use of patterns of different domains. However, in many cases not all relevant patterns belong to the same pattern language. Therefore, some authors include relations to other languages. For example, in the cloud computing pattern language [6], the authors state that the message queues of the *Message-oriented Middleware* pattern are introduced by Hophe & Woolf's [4] *Message Channel* pattern. Unfortunately, not all relevant pattern languages are referred to. Many pattern languages refer only to a few related patterns of other languages or none at all. For example, distributed cloud applications typically have to meet security requirements regarding the communication of the distributed components. To ensure secure communication, Schumacher et al.'s [5] *Secure Channel* pattern could be applied. However, this pattern language is not mentioned in the cloud computing patterns at all [6]. As relations to other pattern languages are often missing, it is difficult to identify related patterns of other languages.

One reason for missing relations is the way pattern languages are documented. Most pattern languages are published in books or scientific publications. Once they are published, they can hardly be changed and, therefore, the pattern languages remain static. This was not intended by Alexander et al. [1], who described them as "*living networks*". Some authors created dedicated websites for their pattern languages (e.g., [8,9,10]), which eases their adaptation. Nevertheless, most websites represent only one particular language. For this reason, pattern repositories have been developed that aim to collaboratively collect patterns of various domains and provide tooling support to edit or extend patterns and relations. Although several pattern repositories support the collection of patterns, patterns of different domains are not organized in pattern languages (e.g., [11,12]) and are thus treated as a single set of patterns and their relations. In contrast to that, a pattern language is more than a collection of patterns and reflects the higher-level *domain* for which the patterns are relevant [13], e.g., for realizing cloud applications. A few repositories organize patterns in pattern languages (e.g., [14,15]), but do not reflect explicit cross-domain relations between patterns in different languages. This knowledge is hidden in individual pattern descriptions. However, without explicit cross-domain relations, and without the context in which these relations are relevant, it is difficult to identify relevant patterns for a given problem. This leads to the overall research question: *"How can relevant patterns and their relations be identified for a problem across different domains?"*

In this paper, we address this problem by introducing a concept to explicitly document cross-domain knowledge relevant for a particular problem. For this, patterns and their relations from different pattern languages can be selected and further relations can be defined as relevant in a specific context. The relations between patterns of different languages are *cross-language relations* that express cross-domain knowledge, i.e. relations between patterns of different pattern languages. Thus, it is possible to combine and extend pattern languages – a truly *living network of patterns*. Based on our previous experience with pattern repositories, we show how support for the concept can be integrated into a pattern repository by adapting our previous toolchain [16,17,18]

which we refer to as the *Pattern Atlas* [19]. The remainder of the paper is structured as follows: Section 2 describes fundamentals and a motivating scenario. In Section 3, we introduce our concept and show how tooling support for it can be integrated into a pattern repository. In Section 4, we present a concrete use case and describe the realization of our prototype. Section 5 describes related work and is followed by a discussion in Section 6. Finally, Section 7 concludes the paper.

2 Background and Motivation

In this section, we first introduce patterns and pattern languages and then motivate that for real-world problems often patterns from multiple domains have to be considered. Based on the motivating scenario, we further refine the research question.

2.1 Patterns and Pattern Languages

As already mentioned, patterns are used to gather knowledge about proven solutions for recurring problems in many different fields, especially in the domain of software and information technology [7] but also in humanities it is a common concept [20]. They describe the core idea of the solution in a general manner, which means in case of software engineering patterns that they are independent of a specific technology or programming language. The general solution concept of a pattern can therefore be applied to a variety of use cases in different ways. Each pattern is identified by its name which we write in italics throughout this paper. Since humans are the targets, patterns are documented as textual descriptions according to a defined pattern format. Even if the pattern formats differ slightly from pattern language to pattern language [21], typical formats for patterns in software and information technology domains contain several common sections [22]: A section about the addressed *problem*, the *context* in which the problem might arise, *forces* which direct the problem, the proposed *solution*, the *resulting context* describing which forces have been resolved, and a section showing a *sketch* of the solution. Often other patterns are only referenced in the textual description of one of these sections. Some authors have introduced explicit sections to describe the relations of the pattern and give them defined semantics [23], such as *"Variations"* [6,9], *"See also"* [5], or *"Next"* [4]. Further examples of semantics for relations between patterns can be found in [24,25,26].

Patterns and relations are the basic building blocks of pattern languages. In this work, we build on the premise that a pattern language is more than a collection of patterns, but a designed system [27]. This means that (i) relations of a pattern language are designed to guide the reader towards suitable patterns and (ii) each pattern solves a specific problem that is related to the overall context of the pattern language [28,13], e.g., in the domain of cloud computing, enterprise integration, or security.

2.2 Motivating Scenario and Problem Statement

Often patterns of several domains have to be considered for a real-world problem. For example, suppose a software developer wants to build a secure elastic cloud application. An elastic application responds to changing workload by adjusting the amount of

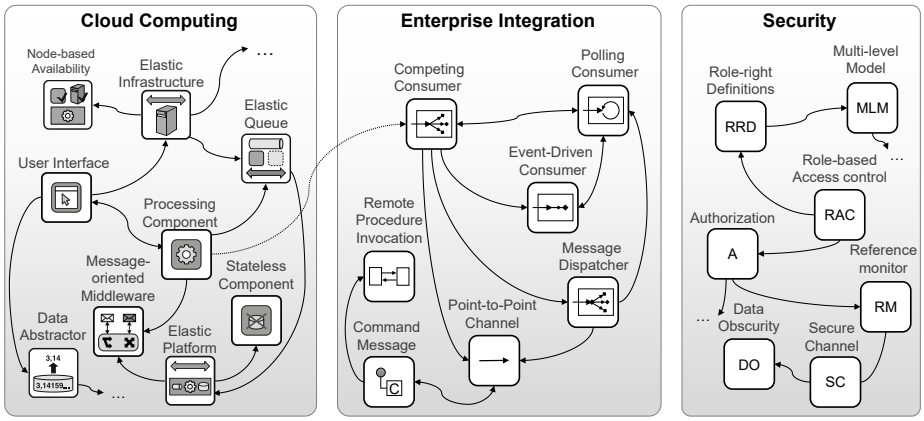


Fig. 1. Patterns and their documented relations of multiple pattern languages: Cloud computing patterns [6], enterprise integration patterns [4], and security patterns [5].

resources allocated to the application [6]. This ensures that neither too many resources (which is costly) nor too few resources are allocated over a long period.

The cloud computing patterns in Figure 1 on the left provide several patterns relevant for an elastic cloud application: For example, an *Elastic Infrastructure* provides a dynamically adjustable infrastructure to a customer to deploy an application and an *Elastic Queue* can be used to monitor the number of messages in a queue and to adjust the number of *Processing Components* handling the requests. In the context of an elastic cloud application, the *Processing Components* are often implemented as *Competing Consumers* as any of the instances can receive and process an incoming request. Therefore, this enterprise integration pattern is explicitly referred to in the *Processing Component* pattern. Since messaging is often used for integrating cloud application components, several cloud computing patterns also refer to other enterprise integration patterns. For example, the authors state that the *Message-oriented Middleware* pattern summarizes selected enterprise integration patterns that are not explicitly listed. However, often explicit references to related pattern languages would be helpful. An example of missing cross-language relations can also be found in our motivating scenario: The enterprise integration patterns were published before the cloud computing patterns and thus never reference them. And although most elastic cloud applications must meet certain security requirements, such as secure communication between application components, as provided by the *Secure Channel* pattern of the security patterns, no security patterns are mentioned and, thus, no cross-language relations exist. It can easily be seen that cross-language relations are also important for pattern languages of other areas than software, e.g., for realizing a film scene, patterns from different domains (costumes, music, and film settings) are needed [26].

But even if cross-language relations exist, they are often not properly documented. The pattern languages depicted in Figure 1 are published in books [6,4,5] or on dedicated websites [8,10]. Besides scientific publications and dedicated websites, patterns are published in repositories that aim to collect patterns in collaboration [16]. However,

even with the tooling support of current repositories, it is challenging to find related patterns for a given problem: Several repositories do not organize patterns in pattern languages [11,12] and treat patterns only as a simple interconnected set. Thus, the domain of the pattern language is not visible and cannot serve as an entry-point for the reader. The few repositories organizing patterns in pattern languages [16,14] list cross-language relations in individual pattern descriptions which are therefore not obvious. None of the repositories known to us enables to document patterns and relations for a specific context (e.g., secure elastic cloud applications). Consequently, finding suitable patterns across pattern languages for a certain problem is a cumbersome, manual process. And especially if a large number of patterns must be considered, this process can be time-consuming. For example, the cloud computing pattern language comprises 74 patterns while the enterprise integration pattern language consists of 65 patterns. Among these patterns, often only a subset is relevant for a certain problem. In Section 1, we questioned how this subset and their relations can be identified across different domains. The domain of the problem may be addressed by either one or multiple pattern languages. In this work, we focus on pattern languages as they also define sophisticated relations and we do not consider simple pattern collections. As a result, the research question can be reformulated as follows:

> ***Research Question I***: *"How can relevant patterns in one or more pattern languages be identified for a certain problem?"*

The main purpose of patterns and pattern languages is to document knowledge. The additional knowledge about which patterns and relations are relevant for a particular cross-domain problem area is also worth documenting. Especially if multiple pattern languages have to be considered, it can be beneficial to share and extend this knowledge in collaboration. Therefore, a second question can be derived from the original question:

> ***Research Question II***: *"How can this knowledge about relevant patterns be documented in a reusable manner?"*

To address these questions, sufficient tooling support is needed to document this knowledge that may span different pattern languages. As mentioned earlier, documenting patterns and relations for a specific problem is currently not supported by any existing pattern repository known to us.

3 Pattern Views

In this section, we introduce our concept and tooling support to tackle the research questions introduced in Section 2. First, we introduce *pattern views* as a concept to document cross-domain knowledge for a particular context that requires patterns and relations across pattern languages. A pattern view can then be used to identify relevant patterns for the problem that is defined by the context of the pattern view (***Research Question I***). Then, a formal definition of pattern views is given. Finally, the integration of pattern views in a pattern repository is shown (***Research Question II***).

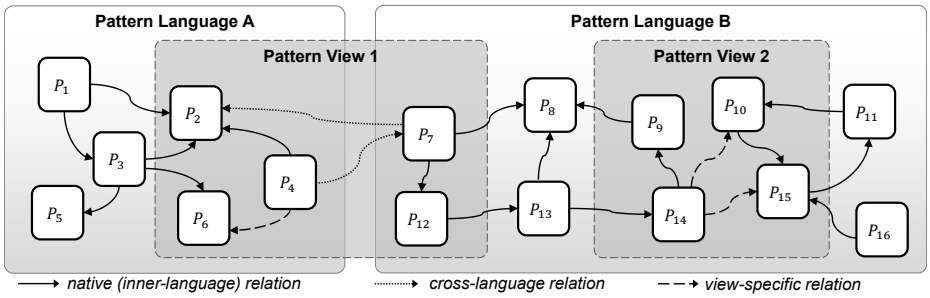


Fig. 2. The concept of pattern views: Pattern views can contain patterns of either multiple pattern languages (pattern view 1) or one single pattern language (pattern view 2).

3.1 The Concept of Pattern Views

Alexander et al. [1] already mentioned in the publication of the first pattern language that if a certain problem requires patterns of different pattern languages, patterns of different domains can be combined as needed. Based on this idea we introduce *pattern views* as a concept (i) to explicitly define the context in which a set of patterns and relations is relevant, (ii) to specify new relations between patterns that are relevant in this specific context, and (iii) to preserve knowledge about the pattern languages from which the patterns originate.

Figure 2 illustrates our concept: A pattern view comprises patterns of either different pattern languages (pattern view 1) or a single pattern language (pattern view 2). For example, patterns from different languages are relevant for a secure elastic cloud application, while only a subset of the cloud computing patterns are relevant for the deployment of a cloud application. The relations between the contained patterns in a pattern view are either those already defined in the original language or newly defined relations that are relevant in the defined context of the pattern view. We distinguish between (i) *native relations*, which are inner-language relations defined by a pattern language, (ii) *cross-language relations*, which are either described in a pattern language or newly introduced by a pattern view, and (iii) *view-specific relations*, which are newly introduced inner-language relations. Especially cross-language relations are often poorly documented in pattern languages. The relevance of a pattern view for a certain use case is determined by its context: The context guides the pattern users, e.g., software architects, to identify a pattern view for his or her particular problem. Thus, pattern views enable to document knowledge about the joint use of patterns and pattern languages for a particular problem explicitly and reusable for other users. In Section 4.1, a pattern view containing patterns relevant in the context of *secure elastic cloud applications* is described in detail as a sample. As a result, an individual pattern can be considered from different perspectives: It is primarily anchored in its original pattern language, but can also be part of different views that place the pattern in a specific context of an overarching problem. As a pattern view can reuse and extend the existing structure of underlying pattern languages, new structures emerge. This supports one aspect of Alexander's *living network of patterns* which allows constant change.

The term *pattern view* is inspired by two existing concepts in computer science: In database management systems, database views can be used to represent subsets of data contained in regular tables. They can join, aggregate, or simplify data from multiple tables and represent them as a single virtual database table. For patterns, the same can be done by our pattern views: Patterns from multiple sources (pattern languages) can be included in a single pattern view. New relations for the pattern view can be defined, just like a database view can refer to other tables. Another analogy to pattern views is the notion of architecture views in architecture descriptions [29]. An architecture view represents the architecture of a system from a specific viewpoint that is in accordance with a certain set of stakeholders' concerns [29]. Depending on the concerns of the different stakeholders, a suitable architecture description can be created, e.g. a process view for process architects or a software distribution view for software developers. Avgeriou & Zdun [30] use this definition to assign architectural patterns to their primary architectural view, e.g., the *Client-Server* pattern to the component-interaction view. We go beyond this and define pattern views as a representation of pattern languages based on problem scopes. Such a scope of a pattern view represents the context in which the patterns and pattern languages may be used to address the concerns of the pattern user.

In Section 4.1, we present a pattern view for *secure elastic cloud applications* that is aimed towards cloud software architects and contains several patterns for the integration of the application components. Although our work is based on information technology pattern languages, our concept does not rely on specific properties of patterns of this domain. Therefore, our concept may be applied, e.g., to patterns for costumes [31] or building architecture [1] in the future.

3.2 Formalization of Pattern Views

Being composed of patterns and relations, pattern languages are commonly formalized as graphs [23,7,32,33]. This notation is also used in Figure 2 for the pattern languages A and B: Patterns are represented by nodes, and edges define the relations between them. Some authors assume a hierarchical ordering of the patterns and restrict the graph to be acyclic [1,7,13]. Because in practice arbitrary relations are used in pattern languages [23], we do not enforce hierarchical ordering. Therefore, we allow cyclic edges in our definition of a pattern language graph that is based on our previous work [23]:

Definition 1 (Pattern Language). *A pattern language is a directed, weighted graph $G = (N, E, W) \in \mathfrak{G}$, where $\mathfrak{G}$ is the set of all pattern languages, N is a set of patterns, $E \subseteq N \times N \times W$ is a set of relations, and W is a set of weights used to reflect the semantics of the relations between patterns. Thereby applies that $W \subseteq \mathfrak{W}$ where $\mathfrak{W}$ is the set of all "meaningful" weights.*

In our concept introduced in Section 3.1, we distinguish three kinds of relations between patterns for which we formally define three categories of relations:

Definition 2 (Pattern Relation Categories). *Let $G = (N, E, W) \in \mathfrak{G}$. Then $e \in E$ is called **native relation** (more precisely $\mathbf{G}$-**native** relation). With $\hat{G} = (\hat{G}, \hat{E}, \hat{W}) \in \mathfrak{G}$ $(\hat{G} \neq G), (n, \hat{n}, w) \in N \times \hat{N} \times \mathfrak{W}$ is called **cross-language relation** (more precisely **cross-**$(\mathbf{G}, \hat{\mathbf{G}})$ relation). Finally, $(n, n', w) \in (N \times N \times \mathfrak{W}) \setminus E$ $(n \neq n')$ is called **view-specific relation** (more precisely **view-**$\mathbf{G}$-specific relation).*

Note that these categories are mutually exclusive. Relations of all three categories can be part of a pattern view which is defined as follows:

Definition 3 (Pattern View). *A graph* (P, R, S) *is called* ***pattern view****:* $\Leftrightarrow \exists \mathfrak{H} \subseteq \mathfrak{G}$:

(i) $P \subseteq \bigcup_{(N,E,W) \in \mathfrak{H}} N$
(ii) $R = R_n \cup R_c \cup R_s$ *with*
  (a) $\forall e \in R_n \; \exists G \in \mathfrak{H} : e$ *is G-native*
  (b) $\forall e \in R_c \; \exists G, \hat{G} \in \mathfrak{H} : e$ *is cross-*$(G, \hat{G})$
  (c) $\forall e \in R_s \; \exists G \in \mathfrak{H} : e$ *is view-G-specific*
(iii) $S \subseteq \mathfrak{W}$

While both pattern languages and pattern views are directed graphs and thus structurally similar, pattern views reuse selected structures of pattern languages (patterns and native relations) and extend by view-specific relations and cross-language relations.

3.3 Tooling for Pattern Views

In previous works, our toolchain has been introduced as a collaborative tool for documenting and managing patterns and pattern languages [16], as well as concrete solutions [34,18] that are implementations of the patterns with a particular technology or in a particular programming language in case of software engineering patterns. Pattern research is actively supported as experts can analyze concrete solutions in collaboration and as a result identify best practices and document patterns [16]. Based on an analogy to cartography we refer to our toolchain as the *Pattern Atlas* [19].

Figure 3 illustrates the abstract architecture of the Pattern Atlas with the newly developed components in black. In the pattern repository, patterns and relations between them are managed. The patterns as well as their relations are organized in pattern languages. The metadata defines the pattern formats for the different pattern languages as well as the semantics of the relations. Analogously, the solution repository stores concrete solutions and their relations, which are organized in solution languages. Concrete solutions are related to patterns: While a pattern captures the essence of multiple concrete solutions, the documentation of a concrete solution eases the application of its pattern to a specific problem [17]. In addition, aggregation descriptors are stored that specify how different concrete solution artifacts can be combined [17]. These combined solutions are especially relevant if multiple patterns (and thus their concrete solutions) need to be applied to a broader problem. The solution repository for managing solution languages highly depends on the domain of the solution, e.g., for concrete solutions of costumes detailed descriptions of clothing pieces are relevant [35] whereas solutions of software patterns can be code snippets [16]. The repositories facilitate to add patterns and solutions as textual descriptions and browse the pattern languages as well as solution languages. For this work, we enriched our previous realization of the toolchain [16,34,18] by the concept of pattern views and added a graphical editor. Further details of the implementation are described in Section 4.

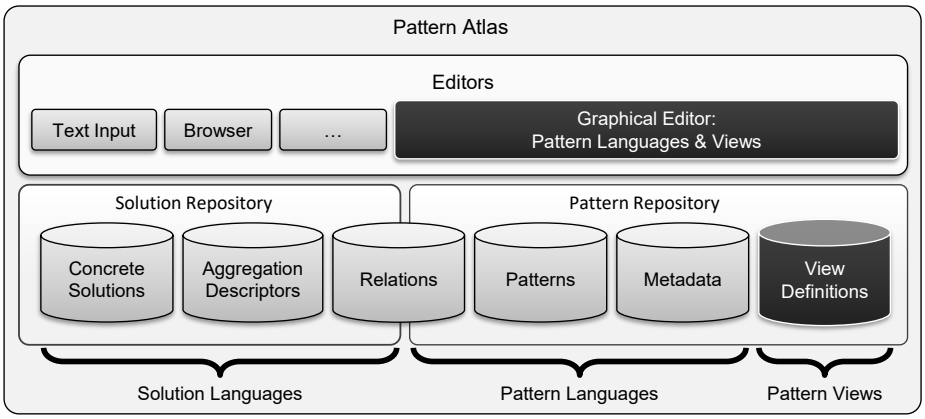


Fig. 3. Abstract system architecture of the Pattern Atlas.

4 Case Study and Prototypical Implementation

In our motivating scenario described in Section 2.1, we stated that patterns from multiple domains are needed for realizing a secure elastic cloud application. In this section, we first present a case study with the pattern view for the context of *secure elastic cloud applications*. We then describe our prototypical implementation which can be used to facilitate the documentation of a pattern view.

4.1 Case Study

Users expect a high availability of certain applications. To fulfill this expectation, cloud providers offer infrastructure and services that can be used to guarantee the availability of an application even for a sudden increase in demand, i.e. scalability of the services. Elastic cloud applications deal with changing demand by adjusting the amount of resources that are used for the application [6]. In addition, data security plays a major role, especially when data is exchanged between communication partners.

Figure 4 depicts the architecture of a secure elastic cloud application. The application consists of a *User Interface* component that communicates with *Processing Components* via messaging. Both components are hosted on an *Elastic Infrastructure*. The number of messages in the channel is monitored to determine the workload of the *Processing Component* instances. Depending on the number of messages, the *Elastic Queue* adjusts the number of instances. As any *Processing Component* instance can answer a request, the component is implemented as *Stateless Component* and its instances act as *Competing Consumers* listening on a *Point-to-Point* channel provided by a *Message-Oriented Middleware*. After consuming and processing a message the *Processing Component* instance can send a response via another *Point-to-Point Channel*. To ensure data security, the communication between the component must be encrypted.

For such an application, there is a number of patterns that should be taken into account during implementation. In Figure 5, the pattern view for secure elastic cloud

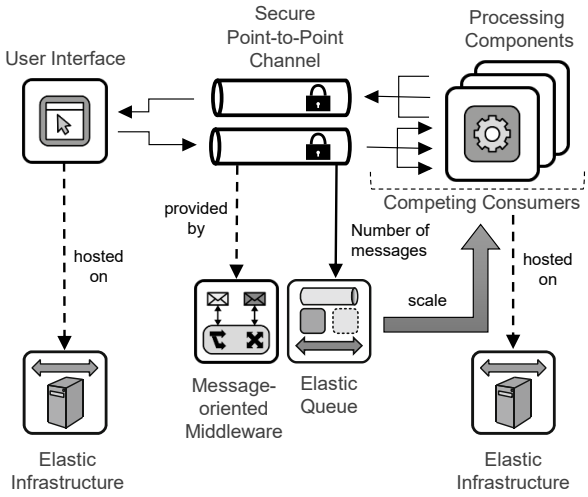


Fig. 4. Architecture of a secure elastic cloud application.

applications is shown. It includes patterns from the cloud computing, enterprise integration, and security pattern languages that are relevant in this specific context. Besides native relations and one cross-language relation from the original pattern languages, three new cross-language relations and two view-specific relations are contained in the pattern view. In addition to the already named patterns also a *Message Dispatcher* can be used to delegate the message to one specific consumer, i.e. one *Processing Component* instance. Each *Competing Consumer* can be implemented as *Polling Consumer*, *Event-Driven Consumer*, or a combination of both [4]. A *Message-oriented Middleware* provides the functionality for communication via messaging and therefore also the secure message channels for the *Competing Consumer* instances. To ensure that a message is consumed only once, the consumers must all listen to the same *Point-to-Point Channel*. As all transferred data of the application must be encrypted, the *Point-to-Point Channel* must also implement the *Secure Channel* pattern. Once defined, this pattern view can be used by other cloud application architects to realize their secure elastic cloud applications. Since the existing knowledge is only enriched by the pattern views, further relevant patterns outside the view can be identified by the native relations in the pattern languages.

4.2 Prototypical Implementation

In the course of this work, we not only extended the pattern repository conceptually but also refactored the implementation of our previous toolchain. The user interface of the pattern repository was implemented as an Angular frontend[1] and we used Spring Boot for implementing a RESTful backend[2].

[1] https://github.com/Pattern Atlas/pattern-atlas-ui
[2] https://github.com/Pattern Atlas/pattern-atlas-api

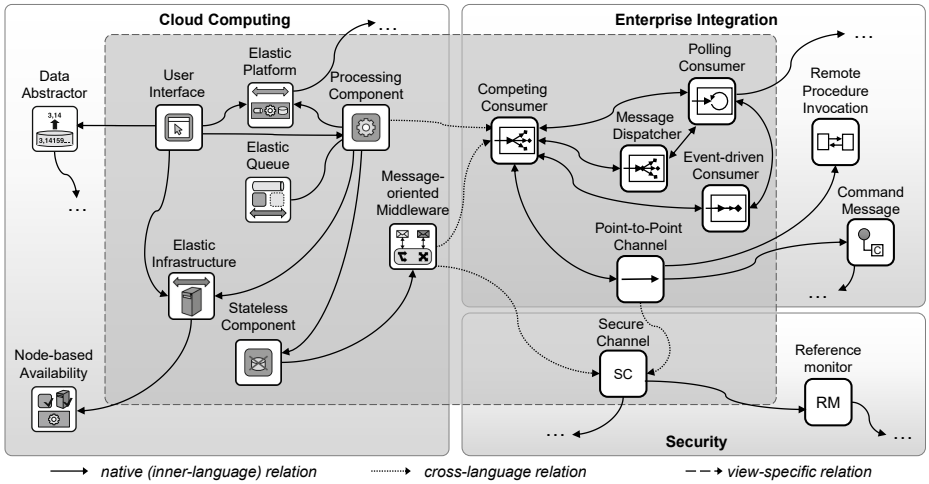


Fig. 5. Pattern view for secure elastic cloud applications.

In the pattern repository, patterns are created in the scope of a pattern language that specifies a certain pattern format. After the creation of a pattern, it can be added to a pattern view. For both, pattern languages and pattern views, we implemented a graphical editor that visualizes their graph structure. In Figure 6, this is shown for the pattern view presented in Section 4.1. Patterns from all pattern languages of the repository can be added via drag and drop. The layout of the graph can be adapted by re-positioning nodes, zooming in and out, and triggering an automatic reformatting of the graph based on the edges. After the selection of a pattern, related patterns are highlighted. New relations can be added by drawing arrows between two pattern nodes and are further defined by a description and specification of the relation type. These new relations are either cross-language relations or view-specific relations. Users can therefore directly edit or interact with the visualized pattern graph and observe how new relations or patterns lead to structural changes as the overall structure of the network of patterns can be grasped immediately. For pattern views, native relations of a pattern can be displayed and selectively imported into the pattern view. This enables the user to reuse the structure that is defined by relations in a pattern language.

5 Related Work

Several authors have examined relations and patterns across multiple pattern languages. Avgeriou & Zdun [30] reviewed architectural views and patterns from different languages. They assigned each architectural pattern to its primary architectural view and defined relations between the patterns. As each of their collection of patterns and relations for an architectural view is worth documenting, we adopted the idea of views as a concept that is not limited to the domain of IT architecture. Caiza et al. [36] standardize the relations of various privacy pattern languages to combine them into a new pattern

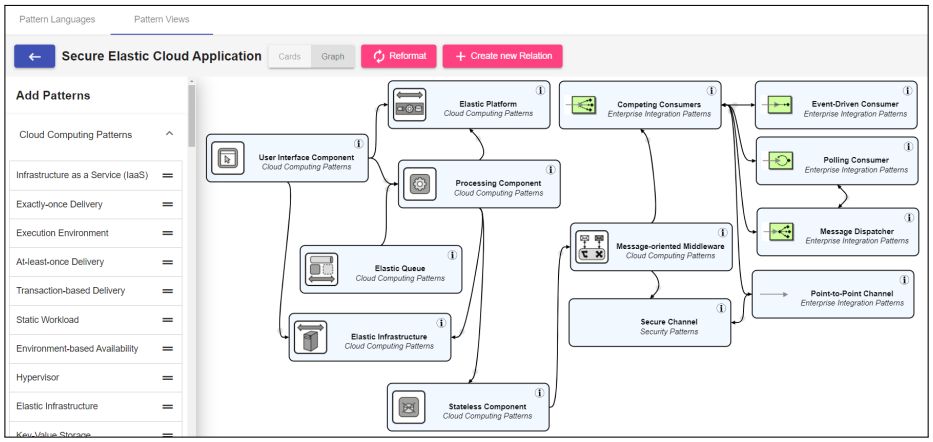


Fig. 6. The secure elastic cloud applications view in the graphical editor. Via drag & drop (left side of the figure), patterns from pattern languages can be added to the pattern view.

language. Porter et al. [32] derived a method to combine pattern languages based on pattern sequences. In contrast, pattern views contain only those patterns of different languages and their relations that are relevant in a certain context. Thus, pattern views are more specific and less complex than potentially large combined pattern languages.

Buschmann et al. [37] introduce pattern stories as textual descriptions that walk a reader through the application of multiple patterns. In an exemplary pattern story, they demonstrate how patterns of multiple languages are used together. The information that is contained in a story - the patterns and the relations described in it - can also be captured in pattern views. However, pattern stories are targeted at illustrating common pattern sequences. Pattern views are not limited to express sequential steps but can express arbitrary relationships.

Reinfurt et al. [38] present an algorithm for finding entry points in pattern languages. Their algorithm can be used to support several manual steps that are needed to document pattern views: For a formalized set of problems related to the context of the pattern view, their algorithm suggests suitable languages and a pattern that serves as a starting point.

Köppe et al. [39] elaborated on requirements for online pattern repositories. They used the term pattern views in the sense that there should be different options (pattern views) for displaying a pattern, e.g., for smaller screens or optimized for printing. Their notion of a pattern view, therefore, defines the visual representation of a pattern whereas we use the term pattern view for a concept to encompass patterns and relations that are relevant for a particular context. Apparently similar terms from other domains are *process views* and *process viewing patterns* [40]. Process views are used to represent a complex business process regarding certain aspects, e.g. by reducing the process to its essentials parts [40]. They are obtained by applying transformations on the process graph of the business process [41]. These transformations have been described by process viewing patterns [40]. In contrast to pattern views that are created by selecting suitable nodes (patterns) and redefine the relations (edges) between them, the former

transformations can be far more sophisticated, e.g., nodes of a process graph can be aggregated.

Pavlič et al. [42] introduced the concept of pattern-containers to represent pattern collections. They formalized how patterns represent knowledge in an ontology. Relations are modeled by specifying that a pattern is related to another pattern. But in their ontology, the relation cannot be described further, and thus, the type of the relation cannot be defined. They define pattern-containers as a way to create pattern collections: Pattern-containers can include patterns and other pattern-containers. A pattern can be included in multiple pattern-containers. But given their ontology, pattern-containers cannot be used to represent pattern views: As it cannot be defined which relations are relevant for a pattern-container, they represent a simple subset of patterns.

Graph-based representations for pattern languages are commonly used to reflect Alexander's description of a network of patterns [37,23,33]. Another pattern repository, *The Public Sphere Project*, mentions that a graph representation of all their patterns and relations was once created [14], but only a sub-graph of it (8 patterns and their relations) can still be found on their website. Nevertheless, even the complete graph is still a static representation of their underlying living pattern network. Schauer & Keller [43] developed a tool for documenting software systems. Although they use patterns to enhance multiple graph-based views for a software system (e.g. as annotations in UML diagrams), they do not offer a general view on patterns. Welicki et al. [44] developed a visualization tool that can be used to search and create relations (including cross-language relations) between patterns in their software pattern catalog. They also implemented different views on a pattern that display e.g. a summary or software-specific views (e.g. source-code of a concrete implementation). The MUSE repository of Barzen [45] offers a graph-based representation of concrete costumes that occur in films and are understood as concrete solutions for costume patterns. However, these tools and repositories do not offer different perspectives on the relations of the patterns or pattern languages. Therefore, no other pattern repository or tool known to us offers graph-based representations of pattern languages and the ability to dynamically combine patterns from different languages to pattern views for a particular problem context.

6 Discussion

Once a pattern view has been documented for a particular problem, the (cross-domain) knowledge about the patterns is made explicit. When initially documenting a pattern view, expert knowledge or experience in the domain of the corresponding pattern languages is required. While this is an additional manual step, it enables other users of the pattern repository to reuse and extend the knowledge captured in the pattern view. In future work, the documentation of pattern views could be further simplified by further tooling support that, e.g., suggests related patterns.

In contrast to pattern languages, patterns of a pattern view can belong to different pattern languages. Thus, independent of the underlying pattern format, each pattern can be integrated into a pattern view. However, since the patterns are unchanged, their pattern descriptions may use different terminologies. For example, the patterns of the pattern view in Section 4.1 use either *"message channel"* [4] or *"message queue"* [6]

to describe the same concept. This is due to the fact that the overall context of the pattern language is not the same: Hohpe & Woolf [4] describe solutions in the context of integrating enterprise applications via a shared message-based communication channel, called message channel. Fehling et al. [6] use the term message queue in the context of cloud computing to emphasize that multiple messages are stored in a queue and can be retrieved by one of the multiple receivers. This behavior of the message queue can be used for scaling as discussed in Section 4.1. Besides the differences in the overall context (or domain) of a pattern language, *pattern primitives* may be used inconsistently by different pattern authors [46]. Pattern primitives are fundamental elements of patterns and can be used as common elements in patterns [47]. However, in the future, the approach may be extended to adapt or standardize pattern descriptions in the context of a pattern view.

Finally, we want to point out that our approach is not based on specific properties of software patterns and, thus, seems to be applicable to pattern languages of other areas.

7 Conclusion and Future Work

In this paper, we introduced the concept of pattern views to explicitly document cross-domain knowledge relevant for a particular problem context. Patterns from either different pattern languages or a single pattern language relevant for a specific problem can be combined into a so-called pattern view. In addition to the patterns and native relations of the underlying pattern languages, view-specific relations can be defined if necessary and cross-language relations can be documented as relevant for the given context. Therefore, cross-domain knowledge expressed by these relations is documented explicitly and within a meaningful context.

We extended the implementation of our pattern repository that was presented in previous works [19,16,17] by the concept of pattern views. Therefore, our repository allows to collect multiple pattern languages and to define pattern views that can combine, reuse, and extend the structure of pattern languages that are given by their patterns and relations. Our repository also offers a graph-based representation for pattern views and pattern languages that visualizes the network of patterns. We plan to collect further pattern languages in the repository, such as Internet of Things patterns [48] or green IT patterns [49] and to extend our collection of pattern views. We will further evaluate if some patterns need to be adapted to be used in the context of a pattern view. For future research, we will especially consider patterns from new research areas such as music [50] or Quantum Computing [51]. Patterns for quantum computing are especially interesting as new technologies need to be integrated into our current software systems (for which we already have patterns at hand). Also, an open access hosting of the pattern repository would offer multiple advantages in the future.

Acknowledgment

This work was partially funded by the BMWi projects *PlanQK (01MK20005N)* and *IC4F (01MA17008G)*. The authors would like to thank Lisa Podszun for her help with the documentation of existing patterns.

References

1. C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, Aug. 1977.
2. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Oct. 1994.
3. M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, Nov. 2002.
4. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2004.
5. M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.
6. C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, Jan. 2014.
7. J. O. Coplien, *Software Patterns*. SIGS Books & Multimedia, 1996.
8. "Cloud computing patterns," https://cloudcomputingpatterns.org/.
9. "Internet of things patterns," http://internetofthingspatterns.com/.
10. G. Hohpe, "Enterprise integration patterns," https://www.enterpriseintegrationpatterns.com/.
11. "Ui patterns," https://ui-patterns.com/.
12. "Pattern catalog," http://designpatterns.wikidot.com/pattern-catalog.
13. J. O. Borchers, "A Pattern Approach to Interaction Design," in *Cognition, Communication and Interaction: Transdisciplinary Perspectives on Interactive Technology*, ser. Human-Computer Interaction Series, S. Gill, Ed. Springer, 2008, pp. 114–131.
14. "The public sphere project," https://www.publicsphereproject.org/.
15. "Open pattern repository for online learning systems," https://www.learningenvironmentslab.org/openpatternrepository/.
16. C. Fehling, J. Barzen, M. Falkenthal, and F. Leymann, "PatternPedia – Collaborative Pattern Identification and Authoring," in *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014.*, Aug. 2015, pp. 252–284.
17. M. Falkenthal, J. Barzen, U. Breitenbücher, and F. Leymann, "Solution languages: Easing pattern composition in different domains," *International Journal on Advances in Software*, pp. 263–274, 2017.
18. M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, and F. Leymann, "Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains," *International Journal On Advances in Software*, vol. 7, no. 3&4, pp. 710–726, Dec. 2014.
19. F. Leymann and J. Barzen, "Pattern Atlas," *arXiv:2006.05120 [cs]*, Jun. 2020, arXiv: 2006.05120. [Online]. Available: http://arxiv.org/abs/2006.05120
20. J. Barzen and F. Leymann, "Patterns as Formulas: Patterns in the Digital Humanities," in *Proceedings of the Ninth International Conferences on Pervasive Patterns and Applications (PATTERNS)*. Athen: Xpert Publishing Services, pp. 17–21.
21. S. Henninger and V. Corrêa, "Software pattern communities: current practices and challenges," in *Proceedings of the 14th Conference on Pattern Languages of Programs - PLOP '07*. ACM Press, 2007, p. 1.
22. J. O. Coplien, *Software patterns*. New York; London: SIGS, 1996.
23. M. Falkenthal, U. Breitenbücher, and F. Leymann, "The nature of pattern languages," in *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*, 10 2018, p. 130–150.
24. J. Noble, "Classifying relationships between object-oriented design patterns," in *Proceedings 1998 Australian Software Engineering Conference (cat. no. 98ex233)*. IEEE, 1998, pp. 98–107.

25. W. Zimmer, “Relationships between design patterns,” *Pattern languages of program design*, vol. 57, pp. 345–364, 1995.
26. M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann, A. Hadjakos, F. Hentschel, and H. Schulze, “Leveraging Pattern Application via Pattern Refinement,” in *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC 2015)*. epubli, Jun. 2015.
27. T. Winn and P. Calder, “A pattern language for pattern language structure,” in *Proceedings of the 2002 Conference on Pattern languages of Programs*, vol. 13, 2003, pp. 45–58.
28. D. J. Meszaros and J. Doble, “A pattern language for pattern writing,” in *Proceedings of International Conference on Pattern languages of program design (1997)*, vol. 131, 1997, p. 164.
29. IEEE Standards Association, *IEEE Std 1471 (2000): IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, Std., 2000.
30. P. Avgeriou and U. Zdun, “Architectural Patterns Revisited – A Pattern Language,” in *In 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*. UVK - Universitaetsverlag Konstanz, Jul. 2005.
31. J. Barzen and F. Leymann, “Costume Languages as Pattern Languages,” in *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014*, P. Baumgartner and R. Sickinger, Eds. Krems: PURPLSOC 2015, Juni 2015, Workshop-Beitrag, pp. 88–117.
32. R. Porter, J. O. Coplien, and T. Winn, “Sequences as a basis for pattern language composition,” *Science of Computer Programming*, vol. 56, no. 1-2, pp. 231–249, Apr. 2005.
33. U. Zdun, “Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis,” *Software: Practice & Experience*, no. 9, pp. 983–1016, Jul. 2007.
34. M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, and F. Leymann, “From Pattern Languages to Solution Implementations,” in *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014)*. Xpert Publishing Services, May 2014, pp. 12–21.
35. J. Barzen, M. Falkenthal, and F. Leymann, *Wenn Kostüme sprechen könnten: MUSE - Ein musterbasierter Ansatz an die vestimentäre Kommunikation im Film*, ser. Digital Humanities. Perspektiven der Praxis. Berlin: Frank und Timme, Mai 2018, pp. 223–241.
36. J. C. Caiza, Y.-S. Martín, J. M. Del Alamo, and D. S. Guamán, “Organizing Design Patterns for Privacy: A Taxonomy of Types of Relationships,” in *Proceedings of the 22Nd European Conference on Pattern Languages of Programs*, ser. EuroPLoP ’17. ACM, 2017, pp. 32:1–32:11.
37. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Oct. 1996.
38. L. Reinfurt, M. Falkenthal, and F. Leymann, “Where to begin: on pattern language entry points,” *SICS Software-Intensive Cyber-Physical Systems*, 2019.
39. C. Köppe, P. S. Inventado, P. Scupelli, and U. Van Heesch, “Towards extending online pattern repositories: Supporting the design pattern lifecycle,” in *Proceedings of the 23rd Conference on Pattern Languages of Programs*, ser. PLoP ’16. USA: The Hillside Group, 2016.
40. D. Schumm, F. Leymann, and A. Streule, “Process Viewing Patterns,” in *Proceedings of the 14th International Conference on Enterprise Distributed Object Computing (EDOC 2010)*. IEEE, Oct. 2010, pp. 89–98.
41. ——, “Process views to support compliance management in business processes,” in *Proceedings of the 11th International Conference on Electronic Commerce and Web Technologies (EC-Web 2010)*, ser. Lecture Notes in Business Information Processing (LNBIP), vol. 61. Springer, 2010, p. 131–142.
42. L. Pavlič, M. Hericko, and V. Podgorelec, “Improving design pattern adoption with Ontology-Based Design Pattern Repository,” Jul. 2008, pp. 649–654.

43. R. Schauer and R. K. Keller, "Pattern visualization for software comprehension," in *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242)*, June 1998, pp. 4–12.
44. L. Welicki, O. Sanjuán, J. Manuel, and J. Cueva Lovelle, "A Model for Meta-Specification and Cataloging of Software Patterns," *Proceedings of the 12th Conference on Pattern Languages of Programs (PLoP 2012)*, Jan. 2005.
45. J. Barzen, "Wenn Kostüme sprechen - Musterforschung in den Digital Humanities am Beispiel vestimentärer Kommunikation im Film," Ph.D. dissertation, Universität zu Köln, 2018.
46. C. Fehling, J. Barzen, U. Breitenbücher, and F. Leymann, "A Process for Pattern Identification, Authoring, and Application," in *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP 2014)*. ACM, Jan. 2014.
47. U. Zdun, P. Avgeriou, C. Hentrich, and S. Dustdar, "Architecting as Decision Making with Patterns and Primitives," in *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge (SHARK 2008)*. ACM, May 2008, pp. 11–18.
48. L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of things patterns for devices," in *Ninth international Conferences on Pervasive Patterns and Applications (PATTERNS) 2017*. Xpert Publishing Services (XPS), 2017, pp. 117–126.
49. A. Nowak, F. Leymann, D. Schleicher, D. Schumm, and S. Wagner, "Green Business Process Patterns," in *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP 2011)*. ACM, Oct. 2011.
50. J. Barzen, U. Breitenbücher, L. Eusterbrock, M. Falkenthal, F. Hentschel, and F. Leymann, "The vision for MUSE4Music. Applying the MUSE method in musicology," *Computer Science - Research and Development*, pp. 1–6, November 2016.
51. F. Leymann, "Towards a pattern language for quantum algorithms," in *Quantum Technology and Optimization Problems*, ser. Lecture Notes in Computer Science (LNCS), vol. 11413. Cham: Springer International Publishing, 2019, pp. 218–230.