PURPLSOC

# The Nature of Pattern Languages

Falkenthal, Michael, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany, falkenthal@iaas.uni-stuttgart.de

Breitenbücher, Uwe, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany, breitenbuecher@iaas.uni-stuttgart.de

Leymann, Frank, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany, leymann@iaas.uni-stuttgart.de

**Abstract**

*Patterns and pattern languages have emerged in many disciplines to capture deep domain expertise and knowledge about solving frequently recurring problems by proven solutions. Thereby, patterns capture the essence of many implementations along with descriptions about how to apply them in combination with other patterns, which manifests in pattern languages.*

*Although pattern languages are a powerful means to preserve and reuse expertise, a clear definition is missing about what a pattern language actually is. Pattern languages are primarily described as being networks of patterns which does not provide a clear and unambiguous foundation to reveal their nature. This lack of rational about the structure behind pattern languages hinders reasoning about them to grasp what connections between patterns are and how the interplay of patterns from different pattern languages can be authored and managed.*

*Therefore, we present a formal notion of pattern languages as node-coloured and edge-weighted directed multigraphs. We show how this model can be used to sharpen Alexander's idea of pattern languages. Thereby, we illustrate how pattern languages can be authored and adapted to establish living networks of patterns. We further introduce that patterns are specific renderings of such a graph depending on actual problems and use cases at hand. This manifests in the fact that our graph concept extracts relationships between patterns from the patterns themselves, which enables easily adaptable networks of patterns. This can be leveraged as the formal meta-model for developing tool support for authoring and sharing pattern languages among communities via IT-based systems.*

**Keywords:** *Pattern Language; Formalization; Pattern Language Composition; Pattern Graph*

ISBN (tba)
www.purplsoc.org

# 1. Introduction

The comprehensive documentation and efficient reusability of knowledge has been one of the most important challenges for many decades. In 1977, Christopher Alexander and his colleagues published their pioneering idea of *pattern languages*, which are linked documents describing proven solutions for problems that frequently occur in certain contexts (Alexander, Ishikawa, & Silverstein, 1977). Originally their idea of pattern languages was born in the domain of architecture and urban design with the aim of supporting architects in creating well-designed buildings and landscapes. As a proof of the brilliant nature of this idea, we can see the many pattern languages that have emerged in the meantime: Pattern languages can be found in various domains such as, for example, education, systemic transformations, and information technology. They range from languages that capture the essence of learning and teaching (Iba & Miyake, 2010), to languages that provide compelling guidance at emergency situations such as earthquakes (Furukawazono et al., 2013). Also in technical domains such as information technology, pattern languages have been successfully authored and applied, e.g., for designing cloud applications (Fehling, Leymann, Retter, Schupeck, & Arbitter, 2014) or to integrate different systems of an enterprise (Hohpe & Woolf, 2004). Moreover, besides the documentation of proven solutions, pattern languages are also often used to foster the comprehensibility of domains by acting as a domain-specific jargon or as lingua franca.

Pattern languages consist of *patterns* that are linked with each other. A pattern is a human readable document that describes a general solution principle to solve a frequently recurring problem in a certain context. The solution a pattern describes is typically documented in an *abstract* manner in order to enable solving many *concrete* instances of the conceptual problem. For example, Christopher Alexander and his colleagues documented the general principles about how to build well-designed Farmhouse Kitchens in the form of a pattern (Alexander et al., 1977). This pattern is documented abstractly enough to be applied to many concrete buildings, but nevertheless describes all the key solution principles and best practices the authors gained during their many years of architecting houses. Thus, other architects may take such a pattern to get an idea of a proven conceptual solution, which they can refine to solve their concrete problem at hand. Therefore, patterns typically provide a certain degree of freedom for applying the solution principles to a vast amount of concrete instances of the conceptual problems (Alexander et al., 1977).

To unfold the actual generativity, patterns are typically organized as *pattern languages*, which provide a comprehensive means to connect patterns for solving different problems that often occur together. Organized as pattern language, patterns are not just isolated junks of proven solution knowledge but support the navigation through the language along relevant problems that may occur together with the original problem that needs to be solved (Zdun, 2007). For example, if a farmhouse kitchen needs to be designed, typically also the *cooking layout* must be considered. By documenting related patterns, pattern languages form a *network of patterns* that reveals generative combinability of an entire set of patterns, which are typically applied in combination (Alexander et al., 1977; Alexander, 1979; Buschmann, Henney, & Schmidt, 2007). Based on this concept, readers can navigate through the pattern language and select a pattern that solves a particular part of the problem at hand and then navigate to the next patterns along references from the formerly selected one.

Since people constantly create new knowledge, pattern languages typically evolve over time and increase in their size. Therefore, a pattern language cannot be seen as a static result of documenting all important knowledge about a certain domain, but is rather subject to constant change – it's a *living network of patterns.* Moreover, due to this ongoing process of documenting proven solutions in the form of new patterns, also interrelations between different pattern languages become more and more important: Two pattern languages, which originally had different areas of application, may converge through new related patterns that are part of the two languages. Therefore, also the *dependencies* between pattern languages constantly change and must be documented to support users in solving all related problems.

However, especially this key concept of living pattern networks is ironically in contradiction with the typical way pattern languages are documented: Many languages are published in books, papers, or journals, which are static documents that are hard to change. Thus, these forms of documentation only provide *static snapshots* of proven solutions at a certain point in time, but do not reflect the *liveliness of knowledge* in general. To tackle these issues, many authors publish their pattern languages also on webpages that allow for constant changes. Unfortunately, also these webpages often consider only one single pattern language and do not document the dependencies to other languages. Therefore, our overall vision is to support realizing Christopher Alexander's main idea of living pattern networks by information technology. Pattern authors need intuitive means to publish, adapt, and interrelate pattern languages via globally accessible media such as webpages that are linked with each other. To realize this vision, a clear definition of the concept of a pattern language is required.

However, the definition given by Alexander is a summarization of characteristics in natural language: Although he gives a clear mathematical definition of the decomposition of problems into diagrams of forces based on the mathematical rigor of set theory in *Notes on the Synthesis of Form* (Alexander, 1964), there is no such formal definition of a pattern language.

Therefore, in this paper, we translate the ideas of Christopher Alexander and his colleagues, which have been documented only in natural language, into a formal mathematical definition of pattern languages as directed node-coloured and edge-weighted directed multigraphs. This reveals *the formal nature of pattern languages* and provides the basis for further mathematical considerations that are required to realize our vision of globally accessible, living pattern networks based on information technology. We further introduce that patterns are specific renderings of such formal multigraphs depending on actual problems and use cases at hand. This manifests in the fact that our multigraph concept extracts relationships between patterns from the patterns themselves, which enables easily adaptable networks of patterns. In the following, we explain in detail and step by step how the characteristics of pattern languages as described by Alexander in natural text can be translated into formal mathematical definitions. Moreover, we show how our formal definition can be used to interrelate different pattern languages by the original concepts described in *The Timeless Way of Building.*

The remainder of this paper is structured as following: In Section 2 we reveal the nature of pattern languages by developing a formal model stepwise on the basis of the fundamental theory of graphs. In Section 3 we discuss related work, which our approach is built on. We conclude this work in Section 4 by pointing out future research topics we are going to tackle.

# 2. Revealing the Nature of Pattern Languages

Alexander (1964) clearly describes in *Notes on the Synthesis of Form*, the preceding work of *A Pattern Language* (Alexander et al., 1977), how complex design problems can be decomposed into subproblems that are more easily to grasp and solve. Thereby, he provides a fundamental formal notion of design as the process of remedying identified misfits. He specifies his approach with mathematical rigor by means of set theory and the analysis of correlations between misfits. This is especially remarkable because he translates thoughts, concepts, and an overall method into a mathematical model, which allows to logically reason about the presented approach. Thus, the concept of *patterns* he introduced in the succeeding works goes back to this clearly described model. The main aspect about patterns is that they typically are not isolated but are organized into pattern languages that unfold generative power by the combined expressiveness of all interrelated patterns. However, the model of a pattern language as described in *A Pattern Language* (Alexander et al., 1977) and *The Timeless Way of Building* (Alexander, 1979) is pretty much a summary of qualitative statements without the preciseness used for the fundamental concepts patterns build on. In the following, we discuss the characteristics of pattern languages by descriptive quotes from both mentioned works. We incorporate additional characteristics of pattern languages that have arisen in the domain of IT, where pattern languages are widely used. These identified characteristics describe the concepts behind pattern languages, which we then translate stepwise into an emerging general formal, mathematical meta-model of pattern languages.

## 2.1. A Network of Patterns

The general understanding of what a pattern language is can be ascribed to the fact that patterns are not just isolated proven solutions for common non-trivial problems. Alexander refers to this by pointing out the collaborative character of patterns. In a *Timeless Way of Building* he states that *"the structure of a pattern language is created by the fact that individual patterns are not isolated"* (Alexander, 1979, p. 311). Thus, there exist inherent relations between different patterns according to the *things* they represent. Moreover, he describes this structural characteristic of a pattern language to be a network of patterns by the quotes *"[a] pattern language has the structure of a network"* (Alexander et al., 1977, p. xviii) and *"[t]he structure of the language is created by the network of connections among individual patterns […]"* (Alexander, 1979, p. 305). Based on the idea of a network of patterns, he introduces the concept of *completeness* of patterns and, thereby, substantiates the inherent relationships between patterns even more. This is because a pattern does not provide a single finalized solution, but rather the actual solution is completed by other patterns it is related to. He underpins this by the statements *"[e]ach pattern sits at the center of a network of connections which connect it to certain other patterns that help to complete it. […] And it is the network of these connections between patterns which creates the language"* (Alexander, 1979, p. 313). Moreover, Alexander states that *"[i]t is, indeed, the structure of the network which makes sense of individual patterns, because it anchors them, and helps make them complete."* (Alexander, 1979, p. 315). Thus, he raises the relationships between patterns to the backbone and core of a pattern language.

The relations between the patterns, i.e., the *paths* through the network of patterns restricts the combinability of patterns to only suitable and relevant ones. Hence, we can grasp the connections between patterns and so the paths through the network of patterns as necessary constraints that eliminate variations of pattern combinations that do not lead to meaningful good solutions. Although, at a first glimpse this seems to limit a pattern language exactly these

restrictions increase the usability and expressiveness of a pattern language immensely to elaborate good solutions. Alexander points this out by the following statement:

*"At this stage, we have defined the concept of a pattern language clearly. We know that it is a finite system of rules which a person can use to generate an infinite variety of different buildings – all members of a family – and that the use of language will allow the people of a village or a town to generate exactly that balance of uniformity and variety which brings a place to life."* (Alexander, 1979, p. 191)

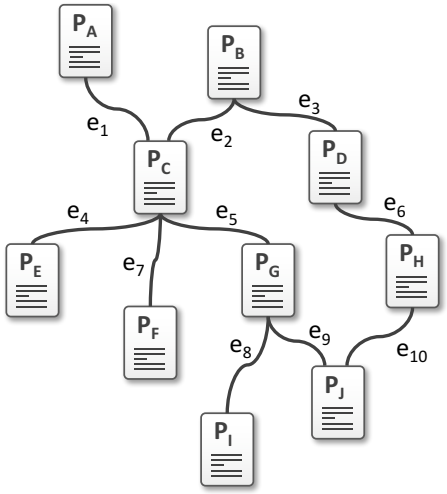


**Figure 1: Network of Patterns as a Graph of Patterns consisting of Nodes and Edges**

From this statement we can derive that a pattern language consists of a finite set of patterns and connections between patterns due to the fact that it is a "*finite system of rules*" (Alexander, 1979, p. 191). Thus, if we draw a figure based on the above identified characteristics we result in a mathematical structure that is called a *graph*. Such a graph is exemplarily depicted in Figure 1. Therein the patterns $P_A$ – $P_J$ are the nodes of the graph and the connections between patterns $e_1$ – $e_{10}$ the edges of the graph. Therefore, we refer to a pattern language as being a *pattern graph* because a graph inherently consists of nodes and edges, which provide us key entities to form networks of patterns as described by Alexander. Based on this interpretation of a pattern language, we provide the following definition of its basic structure.

**Definition 1** (*Pattern Graph*): We define that a pattern language is a pattern graph $\mathcal{G}$ specified as a tuple $\mathcal{G} = (N, E)$. The finite non-empty set $N$ is the set of all patterns of the pattern language and the set $E$ is the set of all edges connecting patterns to form an overall network. Sequences of patterns connected by edges form so-called paths through the graph indicating combinations of patterns that are relevant.

We define

$$\mathcal{G} = (N, E)$$

with

(i) $N$ is a set of patterns
(ii) $card(N) \in \mathbb{N}$
(iii) $E \subseteq \wp(N)$
(iv) $\forall e \in E: card(e) = 2$

(v) $n_1, n_2, \dots, n_k \in N$ is a path from $n_1$ $to$ $n_k$: $\Leftrightarrow \{n_1, n_2\}, \{n_2, n_3\}, \dots, \{n_{k-1}, n_k\} \in E$
(vi) A path $n_1, n_2, \dots, n_k \in N$ is a simple path : $\Leftrightarrow \forall 2 \leq i, j \leq k-1: n_i \neq n_j$ ■

By this definition we enable the analysis of pattern languages by mathematics and well-known algorithms operating on graphs. For example, by this definition we can explain the concept of adjacency of patterns, which plays a core role in Alexander's theory, because it is the network a pattern is centered in that helps to develop complete solutions of the pattern. For this reason, if we want to investigate how to complete a pattern, we now have a formalism that explicitly defines which patterns are adjacent to each other, namely by the edges of the graph that connect them. Referring to the abstract pattern language depicted in Figure 1 we can easily determine that the patterns $P_A$, $P_B$, $P_E$, $P_F$, $P_G$ are required to complete pattern $P_C$ because $P_C$ resides in the center of the network connecting all these patterns via edges.

Further, if we want to extract parts of a pattern language which are relevant to solve a concrete problem at hand, we can determine them by searching for simple paths between patterns we are interested in. For instance, if we require the patterns $P_A$ and $P_J$ to be part of a solution to a problem at hand, then we can determine the path $P_A$, $P_C$, $P_G$, $P_J$ that contains relevant patterns in this case. As a consequence, the selected patterns can now be used to investigate all patterns they are connected with in order to complete them. The problem of determining paths through a graph is well investigated and, thus, further motivates the understanding of pattern languages by a mathematical formalism because many mathematical approaches dealing with graphs can be used to support the application of pattern languages (Schöning, 2001).

## 2.2. Organizing the Network of Patterns

In the following, we further refine our meta-model of pattern languages by considering more statements by Alexander. First of all, we incorporate one important aspect which can be derived from the characteristics of patterns. According to Alexander, patterns step in to a process of creating things starting from larger structures to more and more fine-grained structures. Thus, as part of a pattern language each pattern creates so-called *morphological structures*, which are then filled in by other patterns. Thereby, when creating a solution from a pattern it is important to also consider the structure, which the pattern is contained in because it influences respectively restricts the freedom of elaborating the solution. The solution is constrained by the other patterns and their solutions with which it has to form an overall whole. Alexander emphasizes this need by the phrasings:

*"[…] when you build a thing you can not merely build that thing in isolation, but must also repair the world around it, and within it, so that the larger world at that one place becomes more coherent, and more whole; and the thing which you make takes its place in the web of nature as you make it."* (Alexander et al., 1977, p. xiii)

*"Each pattern then, depends both on the smaller patterns it contains, and on the larger patterns within which it is contained."* (Alexander, 1979, p. 313)

*"And you see then what a beautiful structure a pattern language has. Each pattern is itself a part of some larger pattern [...] And each pattern itself gives birth to smaller patterns [...]"* (Alexander, 1979, p. 322)

As a result, the ordering of the network of patterns is based on the fact that there exist patterns that are *larger* than other patterns and, vice versa, there are patterns that are *smaller* than other ones. However, the concept of morphological structures, which arose from the domain

of architecture as discussed above can be generalized if we also consider the application of the pattern approach in other domains. For instance, in the domain of information technology many pattern languages have been successfully authored and are widely used. There are pattern languages for software architecture such as the cloud computing patterns by Fehling et al. (2014), the enterprise integration patterns by Hohpe and Woolf (2004) but also currently emerging research fields such as the internet of things (Reinfurt, Breitenbücher, Falkenthal, Leymann, & Riegg, 2016, 2017) are tackled with the pattern approach, to name just a few. Interestingly, the concept of morphological structures is not equally present in these pattern languages, though we can also discover the concept of ordering of patterns in the network. Thereby, patterns allow to create designs, software artifacts, or components that can be combined and refined with solutions provided by other patterns. Buschmann et al. (2007) call this characteristics *Sustainable Progression* and *Tight Integration* and explain it by the quotes: "*Sustainable progression. Pattern languages must connect their patterns appropriately to ensure that challenges are addressed in the right order, which is essential to creating sustainable designs incrementally and via stable intermediate steps […]"* (Buschmann et al., 2007, p. 269) and *"Tight integration. Pattern languages must integrate their constituent patterns tightly, based on the roles each pattern introduces and the inter-relationships between them."* (Buschmann et al., 2007, p. 270). Hence, we can deduce that the relations between patterns only have proper meaning if we give them direction, so that we can express that one pattern is larger than other patterns or that one pattern has to be applied earlier in the process of design than others. This is depicted in Figure 2 as an example of larger and smaller patterns.

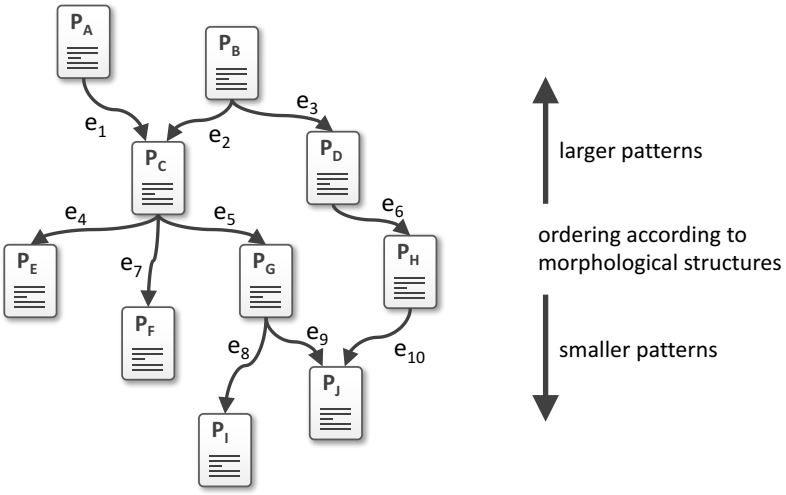


**Figure 2: Pattern Language as Directed Graph**

Therefore, these characteristics of patterns, be it their effect on morphological structures or on the process of creating artifacts in the domain of information technology, have explicit impact on the network of patterns and, thus, on the pattern graph. This means that the fundamental organization of patterns in a pattern language needs to be reflected by more specific relationships in the structure of the pattern graph. Thus, we refine $E$ to be the set of ordered pairs of patterns. This refines $\mathcal{G}$ to be a directed graph (digraph). Consequently, we can express the semantics of a relationship between two patterns by defining that they are connected by an edge, which connects a *larger pattern* with a *smaller pattern*, i.e., the edge between two patterns directs from the larger pattern to the smaller one. Directed edges are commonly drawn as arrows, thus, a directed edge between two patterns is drawn as an arrow whose tail is connected to the larger pattern and whose head is connected to the smaller pattern. Based on

this we can refine our model of a pattern graph to be an Alexandrian pattern graph capturing the expressiveness elucidated in the original pattern language theory by Alexander. Of course this differently applies to domains like information technology where the direction provides other semantics because there are no morphological structures present. Therefore, we first define the Alexandrian pattern graph covering morphological structures and afterwards relax this formalism towards a graph that contains arbitrary semantics on edges between patterns.

**Definition 2** (Alexandrian Pattern Graph): We define an Alexandrian Pattern Graph as a directed acyclic graph (DAC) by refining the set of edges $E$ to contain ordered pairs of nodes expressing the direction from larger to smaller patterns.

We define

$$\mathcal{G} = (N, E)$$

with

(i) $N$ is a set of patterns
(ii) $card(N) \in \mathbb{N}$
(iii) $E \subseteq N \times N$
(iv) $e \in E$, then $\pi_1(e)$ is the starting point or tail of an edge, respectively, the arrow connecting two patterns and $\pi_2(e)$ the endpoint or head
(v) $\forall e \in E: \pi_1(e) \neq \pi_2(e)$
(vi) $n_1, n_2, \dots, n_k \in N$ is a path from $n_1$ $to$ $n_k: \Leftrightarrow (n_1, n_2), (n_2, n_3), \dots, (n_{k-1}, n_k) \in E$
(vii) A path $n_1, n_2, \dots, n_k \in N$ is a simple path $: \Leftrightarrow \forall 2 \leq i, j \leq k-1: n_i \neq n_j$
(viii) $\forall\, n_i, n_{i+1}, \dots, n_k \in N$ that are simple paths holds $n_i \neq n_k$ ■

Such a pattern graph describes the fundamental structure behind a pattern language according to Alexander because it incorporates structural order of patterns. This kind of order is the core expressiveness of a pattern language that allows to navigate purposefully from things that have to be present before other things can be created, from larger structures to the things that are contained in them. Thus, we lift the relations between the patterns to first-class elements in the understanding of a pattern language that carry implicit semantics, which is underpinned by the phrase by Alexander *"[i]n this network, the links between the patterns are almost as much a part of the language as the patterns themselves."* (Alexander, 1979, p. 314).

### 2.3. Semantics of Relations between Patterns

So far, we introduced in Section 2.2 the concept of directed edges to represent the implicit semantics of larger and smaller patterns and, hence, purposeful navigation structures through pattern languages. In the following, we will investigate the semantics of relations between patterns in more detail. As described above, the directed edges in an Alexandrian pattern graph do not only employ navigation structures with a direction but are means to express the semantics that one larger pattern *contains* several smaller patterns as it creates a structure, which gets filled by the smaller patterns. Thus, directed edges are the elements that add actual semantics to the model of a pattern language. This is explained by Alexander in the statement:

*"However, when we use the network of a language, we always use it as a sequence, going through the patterns, moving always from the larger patterns to the smaller, always from the ones which create structures, to the ones which then embellish those structures, and then to those which embellish the embellishments."* (Alexander et al., 1977, p. xviii)

However, in domains besides towns and building architecture the morphological structures are not as obvious and comprehensible. Thus, in these domains the structuring of pattern languages via different link types has emerged. For example, further introduced relationships focus on different types of references between patterns such as "*see also*" and "*consider after*", which are used in the pattern languages on cloud computing (Fehling et al., 2014). Reiners (2013) describes link types that support to express AND, OR, and XOR semantics to specify the interaction of patterns. Further, we have so-called composite patterns, which describe the interplay of different other patterns. They do not aim for describing morphological larger structures but rather describe combinations of patterns for solving strongly-related problems, i.e., they provide an aggregated solution to such composite problems (cf. Buschmann, Henney, & Schmidt, 2007; Fehling, Leymann, Retter, Schupeck, & Arbitter, 2014). These patterns can be linked with the patterns they are composed of via a link type "*composed of*" to indicate the patterns that are part of the composition. In addition to this, the pattern community has created a lot more domain-specific dependencies between patterns, however, for the sake of brevity we focus on the above mentioned as evident examples to further refine our meta-model.

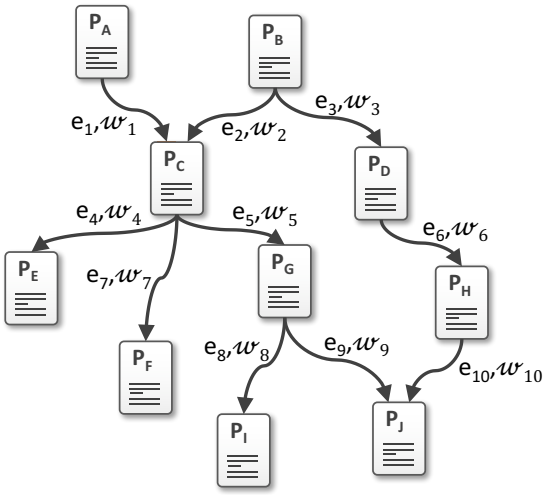


**Figure 3: Directed Weighted Pattern Graph**

Referring to the mentioned examples, we can generalize different link semantics to be arbitrary weights assigned to edges. Thus, we can refine our model of a pattern graph as depicted in Figure 3. Therein different weights are assigned to edges. In the following we transfer this concept to our formalism.

**Definition 3** (Directed Weighted Pattern Graph): To assign weights to edges we associate the pattern graph $\mathcal{G}$ with the set of weights $\mathcal{W}$:

$$\mathcal{G} = (N, E, \mathcal{W})$$

with

(i) $N$ is a set of patterns
(ii) $card(N) \in \mathbb{N}$
(iii) $E \subseteq N \times N \times \mathcal{W}$
(iv) $\mathcal{W} \neq \emptyset$
(v) $e \in E$, then $\pi_1(e)$ is the starting point of the edge $e$, $\pi_2(e)$ is the endpoint of the edge $e$ and $\pi_3(e)$ is the weight assigned to an edge $e$

(vi) $\forall e \in E: \pi_1(e) \neq \pi_2(e)$
(vii) $n_1, n_2, \dots, n_k \in N$ is a path from $n_1$ *to* $n_k: \Leftrightarrow (n_1, n_2), (n_2, n_3), \dots, (n_{k-1}, n_k) \in E$
(viii) A path $n_1, n_2, \dots, n_k \in N$ is a simple path $: \Leftrightarrow \forall 2 \leq i, j \leq k-1: n_i \neq n_j$
(ix) $\forall\, n_i, n_{i+1}, \dots, n_k \in N$ that are simple paths holds $n_i \neq n_k$
(x) $\forall e_i, e_k \in E: \pi_1(e_i) = \pi_1(e_k) \wedge \pi_2(e_i) = \pi_2(e_k) \Rightarrow \pi_3(e_i) \neq \pi_3(e_k)$ ■

It is obvious that a pattern graph according to Alexander corresponds to this definition if and only if all edges represent the semantics *contains*, i.e., $\forall e \in E: \pi_3(e) = contains$. Thus, the weight of all edges in the Alexandrian pattern language is "*contains*".

Besides Alexanders pattern language that only uses a single implicit relationship, many pattern languages use arbitrary kinds of relationships between patterns to interrelate them. Most often, important details about relationships to other patterns are described directly in the pattern document in natural language. This is exemplarily depicted in Figure 4 where pattern documents are depicted that reference other patterns by means of specific labels on edges that indicate the semantics of the references and text passages the provide more information about these references However, having such additional descriptions that detail the references directly captured in pattern documents makes it really hard to extend or adapt them if new patterns join the language. Therefore, we discuss this problem in detail in the next section.

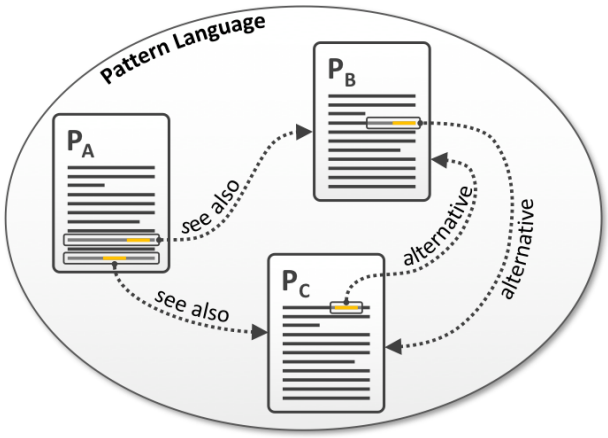


**Figure 4: Different Semantics on Edges**

## 2.4. The Combination of Patterns to Form an Overall Whole

For some pattern languages, it is important to provide a defined structure for the weights along with predefined domains of allowed values. This is for example important if programs should be used to automatically traverse pattern graphs parsing and processing the weights on the edges. For these cases, we allow the annotation of edges by *domain-specific types*. These types further enable to add arbitrary type-specific information describing the relation of two patterns in detail, while the allowed values are defined by type-specific domains. Thus, this concept enables to extract the formerly described details about references to other patterns out of the pattern document to the edges themselves, which separates concerns and eases adapting and extending the pattern language in regard to its liveliness.

An example for this concept can be studied by the sophisticated dependencies between *Remoting Patterns* as presented by Zdun, Hentrich, & Dustdar (2007), which are further extended by Zdun (2007). There, the semantics of the relations between patterns are enriched by additional descriptors that describe the effect of the application of a pattern to the quality

attributes of an IT application architecture. This means that following an edge from one pattern to another pattern it is annotated to the edge that for instance quality attribute A increases but quality attribute B decreases. For instance, such quality attributes can be used, e.g., to document the negative side effect that combining two patterns decreases the overall performance of an application. However, for our investigation it is not of importance what a quality attribute exactly is or what "increases" and "decreases" specifically means in this context. Much more important is that the edges between patterns are used to not only describe how two patterns are related to each other by means of a semantic keyword but also provide more detailed description about how this relation is defined in the specific case. For example, if two patterns are interrelated with a reference having the type "*can be combined with*", additional descriptions directly on the edge provide the user all information required to actually combine the two patterns. Please note, that today this information is often contained in the pattern documents themselves, which makes it hard to maintain the pattern language as all related pattern documents must be adapted if something changes. Therefore, our formalization is not bound to certain semantics but supports the generative nature of pattern languages as arbitrary information can be annotated.

**Definition 4** (Domains of Edge Types): There is a set $\mathfrak{D}$ that is the set of all domains of types used to specify value ranges for type-specify descriptions on edges. The domains of types are used to define reusable structures to add type-specific descriptions to edges. ■

**Definition 5** (Directed Pattern Graph with Types): The weights $\mathcal{W}$ are refined to represent types assigned to edges between patterns whereby each type specifies a reusable structure to add type-specific descriptions to edges to detail the relationship of two patterns. Thus, for all cases that require such expressiveness on edges, we refine the directed weighted pattern graph $\mathcal{G}$ to be a directed pattern graph with types as weights.

We define

$$\mathcal{G} = (N, E, \mathcal{W}, \mathfrak{D}, \alpha, \beta)$$

with

(i) $N$ is a set of patterns
(ii) $card(N) \in \mathbb{N}$
(iii) $E \subseteq N \times N \times \mathcal{W}$
(iv) $\mathcal{W} \neq \emptyset$
(v) $e \in E$, then $\pi_1(e)$ is the starting point of the edge $e$, $\pi_2(e)$ is the endpoint of the edge $e$ and $\pi_3(e)$ is the type assigned to an edge $e$
(vi) $\forall e \in E: \pi_1(e) \neq \pi_2(e)$
(vii) $n_1, n_2, \dots, n_k \in N$ is a path from $n_1$ *to* $n_k: \Leftrightarrow (n_1, n_2), (n_2, n_3), \dots, (n_{k-1}, n_k) \in E$
(viii) A path $n_1, n_2, \dots, n_k \in N$ is a simple path $: \Leftrightarrow \forall 2 \leq i, j \leq k-1: n_i \neq n_j$
(ix) $\forall\, n_i, n_{i+1}, \dots, n_k \in N$ that are simple paths holds $n_i \neq n_k$
(x) $\forall e_i, e_k \in E: \pi_1(e_i) = \pi_1(e_k) \wedge \pi_2(e_i) = \pi_2(e_k) \Rightarrow \pi_3(e_i) \neq \pi_3(e_k)$
(xi) $\alpha: \mathcal{W} \rightarrow \wp(\mathfrak{D})$
(xii) $\beta: E \rightarrow \bigcup_{e \in E} \times_{D \in \mathfrak{D}_{\alpha(\pi_3(e))}} D$
(xiii) $\forall e \in E: \beta(e) \in \times_{D \in \mathfrak{D}_{\alpha(\pi_3(e))}} D$

where $\alpha$ is a map that assigns subsets of all domains to weights and $\beta$ is a map that assigns type-specific descriptions to edges. ■

This definition allows to enrich edges between patterns by arbitrary information about the dependency of the connected patterns. Thus, this formalism can be used to add descriptions about how to combine two patterns to edges that is yet part of the plain text of pattern documents. We argue that this formalism is now powerful enough to represent the real nature of pattern languages because we are capable of moving all information describing dependencies between patterns to those entities of the network of patterns which are meant to reflect them – the edges. Let's investigate this in more detail. Pattern languages are today not present as networks of patterns but the network exists implicitly because pattern documents reference each other and, thus, the edges of the pattern graph are not authored as entities. However, as a consequence of our formalism the pattern documents as authored today are just renderings of a directed pattern graph with types. This means that by rendering the graph in order to produce human readable plain text the information provided on the edges of the graph is inserted into the pattern documents. Thus, our formalism reveals the real nature of pattern languages that was pointed out by Alexander by the statement *"[i]n this network, the links between the patterns are almost as much a part of the language as the patterns themselves"* (Alexander, 1979, p. 314).

Beyond this, our formalism also supports to efficiently extend pattern languages by new patterns. This is because we conceptually overcome the problem that adding new patterns to a pattern language leads to the rephrasing of already present patterns in order to reference the newly added one, which is depicted in Figure 5. This is motivated by Alexander's statements that *"[w]e must […] invent new patterns, whenever necessary, to fill out each pattern which is not complete"* (Alexander, 1979, p. 319) and *"[a] living language must constantly be re-created in each person's mind"* (Alexander, 1979, p. 338).

Our formalism enables that references to other patterns are no longer contained in pattern documents in the form of describing text but are part of the actual references themselves. Thereby, adding new patterns only requires to add new references between the present patterns and the new one. Then, the created edges can be annotated with the information that describes the dependency of two patterns.

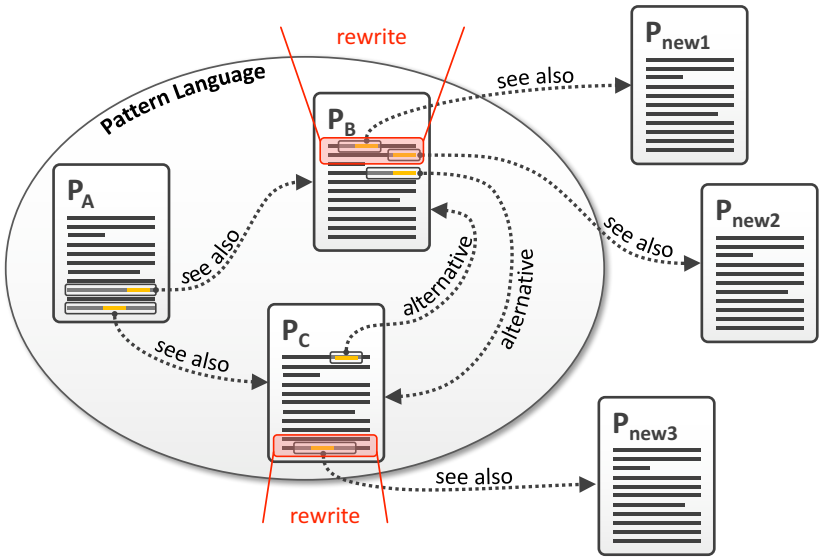


**Figure 5: Problem of rephrasing patterns when new patterns are added to a language**

## 2.5. Living Networks of Patterns

Alexander describes pattern languages as living networks of patterns. This means that they are not just static structures but are liable to change, be it because of the addition of new patterns to a pattern language or also the combination with other pattern languages. In the first case, an existing pattern graph $\mathcal{G}$ is extended by new nodes and edges as described in the former section. This means new patterns are added to the set of patterns $N$ and connections to other patterns are expressed by new edges added to the set of edges $E$ of $\mathcal{G}$.

However, the second case, the combination of pattern languages requires more investigation. Often the knowledge about different aspects of a domain is spread among different pattern languages. For instance, in the domain of information technology the remoting patterns (Zdun et al., 2004), the cloud computing patterns (Fehling et al., 2014) and enterprise integration patterns (Hohpe & Woolf, 2004) are often used together to design application architectures although they were authored almost isolated from each other in the first place. This is conceptually depicted in Figure 6 where patterns from one pattern language do not reference patterns from another pattern language.

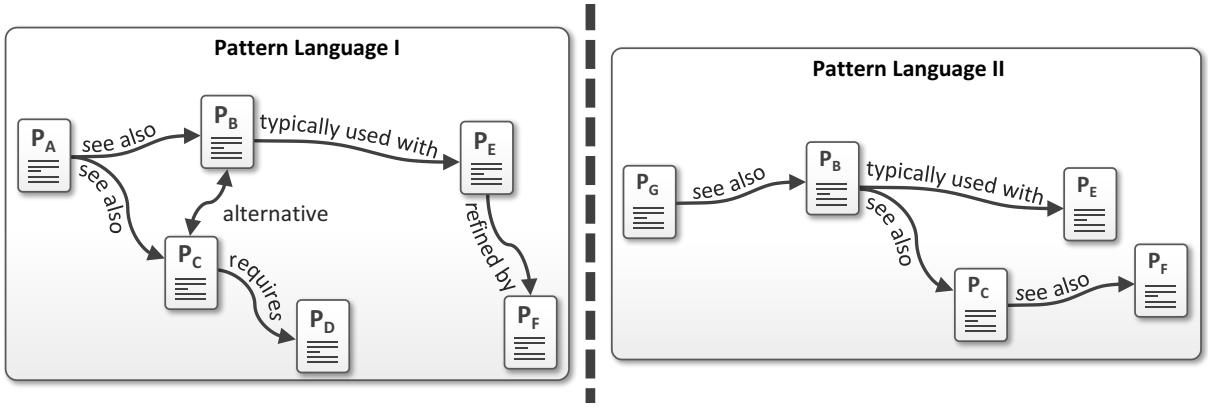


**Figure 6: Isolated Pattern Languages**

This leads to time consuming elaborations of solutions based on the isolated pattern languages because no guidance is provided by means of references between them, although they are used often in combination. Therefore, this scenario motivates that we need a means to combine the pattern languages somehow to support and ease their combined usage. Also Alexander motivates this by discussing that pattern languages that are shared among communities typically diverge and thus, have to be integrated again in order to create structures that inherently work together. He points this out by the quote:

"*We see then, that a language which is shared within a town is a vast structure, far more complex than an individual language.*

*Not merely a network, but a network of networks, a structure of structures, a vast pool of changing, varying, languages which people create for themselves as they take on their different building tasks.*

*And once this kind of structure exists, we have a living language in a town, in just the same sense that our common speech is living"* (Alexander, 1979, p. 341f).

Therefore, based on our formal notion we introduce the operator $\odot$ to support the combination of pattern languages, which combines two graphs to a single one.

**Definition 6** (Pattern Language Aggregator): The set $\mathfrak{G}$ is the set of all pattern language graphs. Two pattern languages are aggregated to a single one by aggregating the underlying graphs based on a set of new edges $\mathcal{E}$:

$$\odot: \mathfrak{G} \times \mathfrak{G} \times \mathcal{E} \rightarrow \mathfrak{G}$$

is the *pattern language aggregator function* that aggregates two pattern language graphs to a single one containing new edges $\mathcal{E}$. ■

Exemplarily, two pattern graphs $\mathcal{G}_1 = (N_1, E_1)$ and $\mathcal{G}_2 = (N_2, E_2)$ are aggregated to $\mathcal{G}_3$ based on new edges $\mathcal{E}$ connecting patterns from $\mathcal{G}_1$ with patterns from $\mathcal{G}_2$ as following:

$$\mathcal{G}_3 = \odot\,(\mathcal{G}_1, \mathcal{G}_2, \mathcal{E}) = (N_1 \cup N_2, E_1 \cup E_2 \cup \mathcal{E})$$

For the sake of simplicity, we write $\mathcal{G}_3 = \mathcal{G}_1 \odot_{\mathcal{E}} \mathcal{G}_2$ to indicate that $\mathcal{G}_1$ is aggregated with $\mathcal{G}_2$ via the new edges $\mathcal{E}$.

The pattern language aggregator $\odot$ is associative and commutative according to the fundamental union operation of set theory because it unions the sets of nodes and the sets of edges and new edges of two pattern graphs.

**Proof:** $\mathcal{G}_1 \odot_{\mathcal{E}_{1,2}} \mathcal{G}_2 = \left(N_1 \cup N_2, (E_1 \cup E_2) \cup \mathcal{E}_{1,2}\right) = \left(N_2 \cup N_1, E_1 \cup E_2 \cup \mathcal{E}_{1,2}\right) = \left(N_2 \cup N_1, E_2 \cup E_1 \cup \mathcal{E}_{1,2}\right) = \left(N_2 \cup N_1, (E_2 \cup E_1) \cup \mathcal{E}_{1,2}\right) = \mathcal{G}_2 \odot_{\mathcal{E}_{1,2}} \mathcal{G}_1$ ■

Therefore, the aggregation of different pattern languages can be written in general as following, whereby the order of the graphs to be aggregated can be changed ad libitum:

$$\mathcal{G}_{n+1} = \mathcal{G}_1 \odot_{\mathcal{E}_{1,2}} \mathcal{G}_2 \odot_{\mathcal{E}_{2,3}} \mathcal{G}_3 \ldots \odot_{\mathcal{E}_{n-1,n}} \mathcal{G}_n$$

Based on this operator we can also explain Alexander's idea of pattern language aggregation as mentioned in *A Timeless Way of Building* (Alexander, 1979). He describes that two yet isolated pattern languages can be aggregated into an overall pattern language by authoring a larger pattern that is more general than at least two patterns from the pattern languages to be combined. Then this pattern can be used to connect both pattern languages into one larger structure because it is used to connect to at least one smaller pattern from each of the pattern languages to be combined. He summarizes this concept as following:

*"And, more subtly, we find also that different patterns in different languages, have underlying similarities, which suggest that they can be reformulated to make them more general, and usable in a greater variety of cases."* (Alexander, 1979, p. 330)

*"Gradually it becomes clear that it is possible to construct one much larger language, which contains all the patterns from the individual languages, and unifies them by tying them together in one larger structure."* (Alexander, 1979, p. 330)

As a consequence, we can apply the introduced aggregation operator to this scenario. Let's assume that two pattern languages $\mathcal{G}_1$ and $\mathcal{G}_2$ have to be combined. Further, a newly authored larger pattern can be grasped as a pattern graph $\mathcal{G}_l$ that contains just one pattern and, hence, no edges. Then $\mathcal{G}_l$ can be aggregated with $\mathcal{G}_1$ via a set of edges $\mathcal{E}_{l,1}$ connecting the larger pattern in $\mathcal{G}_l$ with patterns in $\mathcal{G}_1$. Likewise, $\mathcal{G}_l$ can be aggregated with $\mathcal{G}_2$ via a set $\mathcal{E}_{l,2}$. Therefore, we can specify the aggregation of $\mathcal{G}_1$ and $\mathcal{G}_2$ by a larger pattern into an aggregated pattern graph as $\mathcal{G}_l \odot_{\mathcal{E}_{l,1}} \mathcal{G}_1 \odot_{\mathcal{E}_{l,2}} \mathcal{G}_2 = \mathcal{G}_{aggregated}$.

## 3. Related Work

Alexander et al. (1977) introduce a pattern language as a web of patterns, which supports the navigation through a set of patterns by references between them. The important point is that the relations between the patterns establish so-called pattern sequences, which comprise the generative power of the whole pattern language. Such sequences unfold paths through the pattern language that are typically used to start building a solution based on one pattern, while then, the solution is refined stepwise utilizing the following patterns in the sequence. This is what they call the piecemeal growth that is inherently provided by patterns that build upon others in order to create a whole solution. In contrast pattern sequences the solution paths discussed in this paper are not just navigation structures contained in a pattern language but the actually selected patterns at the time a user chooses applicable patterns in order to solve a concrete problem at hand. Thus, if a pattern sequence allows to choose either one pattern or another at some point, the actual choice of a user is reflected in a solution path. Hence, modelling pattern languages mathematically as presented in this work allows to reflect the nature, i.e., the structure and coherence of a network of patterns appropriately.

Porter et al. (2005) have shown that selecting patterns from a pattern language is a question of temporal ordering of the selected patterns. They show that combinations and aggregations of patterns rely on the order in which the patterns have to be applied. This leads to so called pattern sequences which are partially ordered sets of patterns reflecting the temporal order of pattern application. Their approach requires clear structures and semantics that enables the navigation through a pattern language. Thus, the presented formalization in this paper lays a proper base for the navigation through a pattern language to enable the selection of patterns.

Henney (2006) show how stories can be the nucleus in order to find valuable sequences in a pattern language. Investigating the pattern *Encapsulate Context* by Allan Kelly he shows how a single pattern can be split and integrated into a brief pattern language. Starting from typically occurring problems in the form of stories that reveal a chain of problems and design considerations he shows how sequences through the authored pattern language can ease the application of the knowledge formerly contained in just one pattern more efficiently und clearly. Further, he shows practically how sequences can be related to pattern languages in order to ease their application and to bring out their generative power by piecemeal growth. Both aspects can be covered by the expressiveness of our introduced formalization for pattern languages by means of directed weighted graphs and the selection of proper subgraphs.

Building upon Henney, Zdun (2007) introduces to formalize sequences through pattern languages by means of pattern language grammars. The selection of a pattern is seen as an event in the design process of solutions. According to the temporal order of applying patterns from a pattern language one after the other based on the work of Porter et al. (2005), their formalism considers patterns to be terminal symbols in a formal language, while relationships between patterns are grasped as the production rules of the pattern language grammar. The proven sequences through a pattern language correspond to words, which can be derived by the production rules of the grammar. The effects of a pattern regarding on how it refines and changes the present solution once it is applied is annotated to the grammar in terms of effects on the quality goals of a software architecture. Thus, in contrast to the pattern language by Alexander et al. (1977) the approach by Zdun requires the capability of annotating relationships between patterns by additional semantics. This can be enabled via attaching arbitrary weights on edges. The understandability and unambiguity of attached weights can be maintained by means of ranges of allowed values, which we introduced as type-specific domains.

Mikkonen (1998) tackles the issue on how to formalize the temporal behaviour of components introduced by design patterns in system design. He shows how to overcome the lack of clear semantics in the informal description of pattern solutions, especially focusing on communication aspects between system components described by a pattern. Besides formalizing single patterns also combinations and instantiations of patterns are formalized by means of temporal logic of actions. Although he focusses on formalizing the combination of patterns, i.e., their combined usage in software systems the approach lacks support for clearly specifying clear navigation structures through pattern languages, which our approach allows.

Mirnig & Tscheligi (2014) introduce a general pattern framework based on set theory. This framework provides a general theory of patterns in order to explicate knowledge in pattern structures and relate patterns into pattern languages. Their approach is general due to the definition of patterns and pattern languages by means of set theory and, therefore, provides a domain independent fundamental method to define patterns and pattern languages. Further, they introduce a conceptual mechanism by means of descriptors and targets to combine patterns from different domains and pattern languages, respectively. In contrast to our approach, they do not introduce the concept of pattern relations as first-class citizens in their meta model as we do by means of weighted edges that can carry arbitrary semantics and descriptions. Thus, while our formalization focusses on the structure and characteristics of a pattern language and reveals it as a certain kind of graph their approach unfolds from a pattern centric view. Thereby, their approach lacks concreteness about how it supports and guides to create structures that conform to what is known as a pattern language, because the introduced concepts of descriptors and targets are specified to vaguely.

Bayley & Zhu (2010) describe the composition of patterns via operators. These operators can be used to formally specify the relation between patterns and, thus, form a pattern language. The introduced operators are examples of specific weights of edges as described in this work.

Salingaros (2000) describes the structure of pattern languages. He mentions that patterns can be grasped as nodes of a graph that are connected to each other but does not give a clear formalism about that graph as we do in this work. He further describes that ordering of patterns depends on hierarchical levels, i.e., the structure that evolves by smaller and larger patterns. However, he does not clearly describe how such semantics can be expressed via the concept of a graph.

Falkenthal, Barzen, Breitenbücher, Fehling, and Leymann, (2014a), (2014b) and Falkenthal et al. (2016) have shown that specific link types can be used to establish navigation structures through pattern languages towards concrete realizations of the contained patterns. Falkenthal and Leymann (2017) further introduced the concept of solution languages to organize concrete implementations of patterns similarly to pattern languages to ease and guide their reuse. Since all of these concepts build uppon the core ideas and characteristics of pattern languages also the introduced formalism in this work can be applied to them which lays a basis for the development of integrated pattern and solution repositories with sophisticated user support.

Barzen and Leymann (2015) derive a formalization of pattern languages based on the interplay of clothing in the domain of costumes in films. They present an ontological description of effects of clothing, which is used to express patterns as clothing with similar effects regarding impressions on the audience of films. Thereby, they define a set of domain-specific relations that allows to create a graph representing a costume language. An environment that supports this language is outlined by Fehling et al. (2015)

Finally, different authors (Borchers, 2000; Coplien, 1996; Porter et al., 2005; Reiners, 2012) have mentioned that a pattern language can be grasped as a directed acyclic graph. However, none of them derived the actual semantics of such a model clearly from the core ideas and thoughts by Alexander and refined them towards a sophisticated meta model as expressive as ours.

## 4. Conclusion and Future Work

In this paper, we derived a formal notion of pattern languages on the basis of fundamental mathematical concepts from graph theory. Thus, we provide an explanation about what pattern languages are on the basis of structural dependencies that can be interpreted, i.e., we revealed the underlying nature of pattern languages. Thereby, the general formal model of a pattern language is developed stepwise so that we believe that our formalism covers the structural backbone of most, if not even all present pattern languages. We presented different manifestations of our formalisation differing in their expressiveness. Hence, readers from different domains can step into the formalization that is mostly suitable for developing pattern languages in his domain.

However, we grasp this work as the basis for our research about how to efficiently support the authoring, the maintenance, and development of pattern languages across communities by means of appropriate pattern repositories. Thus, we envision to leverage the whole expressiveness of the presented formalism in the form of collaborative pattern language repositories that inherently incorporate and support the features as shown in this work on the basis of mathematical structures. We especially plan to force investigations on enabling the application of graph algorithms that are now applicable on the basis of graph theory.

## 5. References

Alexander, C. (1964). *Notes on the Synthesis of Form*. London: Oxford University Press.

Alexander, C. (1979). *The Timeless Way of Building*. New York: Oxford University Press. http://doi.org/10.1080/00918360802623131

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A pattern language: towns, buildings, construction*. New York: Oxford University Press.

Barzen, J., & Leymann, F. (2015). Costume Languages as Pattern Languages. In *Pursuit of Pattern Languages for Societal Change (PURPLSOC) - The Workshop 2014: Designing Lively Scenarios With the Pattern Approach of Christopher Alexander* (pp. 88–117).

Bayley, I., & Zhu, H. (2010). A Formal Language of Pattern Compositions. *PATTERNS 2010, The Second International Conferences on Pervasive Patterns and Applications*, (c), 1–6.

Borchers, J. O. (2000). A pattern approach to interaction design. *Proceedings of the Conference on Designing Interactive Systems Processes, Practices, Methods, and Techniques - DIS '00*, 369–378. http://doi.org/10.1145/347642.347795

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. Wiley & Sons.

Coplien, J. O. (1996). *Software Patterns*. SIGS Books & Multimedia.

Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., & Leymann, F. (2014a). Efficient Pattern Application : Validating the Concept of Solution Implementations in Different Domains. *International Journal On Advances in Software*, *7*(3&4), 710–726.

Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., & Leymann, F. (2014b). From Pattern Languages to Solution Implementations. In *Proceedings of the 6th International Conferences on Pervasive Patterns and Applications - PATTERNS 2014* (pp. 12–21). Xpert Publishing Services (XPS).

Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F., Hadjakos, A., … Schulze, H. (2016). Leveraging Pattern Applications via Pattern Refinement. In *Proceedings of Pursuit of Pattern Languages for Societal Change*.

Falkenthal, M., & Leymann, F. (2017). Easing Pattern Application by Means of Solution Languages. In *Proceedings of the 9th International Conferences on Pervasive Patterns and Applications - PATTERNS 2017*. Xpert Publishing Services (XPS).

Fehling, C., Barzen, J., Falkenthal, M., & Leymann, F. (2015). PatternPedia – Collaborative Pattern Identification and Authoring. In *Pursuit of Pattern Languages for Societal Change (PURPLSOC) - The Workshop 2014: Designing Lively Scenarios With the Pattern Approach of Christopher Alexander* (pp. 252–284). epubli GmbH.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., & Arbitter, P. (2014). *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer.

Furukawazono, T., Studies, I., Seshimo, S., Studies, I., Muramatsu, D., & Iba, T. (2013). Survival Language : A Pattern Language for Surviving Earthquakes. In *Proceedings of the 20th Conference on Pattern Languages of Programs* (p. Article No. 30). ACM.

Henney, K. (2006). Context Encapsulation: Three Stories, a Language, and Some Sequences. In *Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*. Irsee.

Hohpe, G., & Woolf, B. (2004). *Enterprise Integration Patterns: Designing, Building, And Deploying Messaging Systems*. Addison-Wesley.

Iba, T., & Miyake, T. (2010). Learning patterns: a pattern language for creative learners II. In *Proceedings of the 1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010)* (p. I-41--I-58). New York, USA: ACM Press. http://doi.org/10.1145/2371736.2371742

Mikkonen, T. (1998). Formalizing Design Patterns. In *International Conference on Software Engineering* (pp. 115–124). Kyoto.

Mirnig, A. G., & Tscheligi, M. (2014). Building a General Pattern Framework via Set Theory : Towards a Universal Pattern Approach. In *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS).* (pp. 8–11). Xpert Publishing Services (XPS).

Porter, R., Coplien, J. O., & Winn, T. (2005). Sequences as a basis for pattern language composition. *Science of Computer Programming*, *56*(1–2), 231–249. http://doi.org/10.1016/j.scico.2004.11.014

Reiners, R. (2012). A Pattern Evolution Process - From Ideas to Patterns. In *Lecture Notes in Informatics - Informatiktage 2012* (pp. 115–118). Gesellschaft für Informatik (GI).

Reiners, R. (2013). *An Evolving Pattern Library for Collaborative Project Documentation*. RWTH Aachen University.

Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., & Riegg, A. (2016). Internet of Things Patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM.

Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., & Riegg, A. (2017). Internet of Things Patterns for Devices. In *Proceedings of the 9th International Conferences on Pervasive Patterns and Applications - PATTERNS 2017* (pp. 117–126). Xpert Publishing Services (XPS).

Salingaros, N. a. (2000). The structure of pattern languages. *Arq: Architectural Research Quarterly*, *4*(2), 149–161. http://doi.org/10.1017/S1359135500002591

Schöning, U. (2001). *Algorithmik*. Springer.

Zdun, U. (2007). Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, *37*(9), 983–1016. http://doi.org/10.1002/spe.799

Zdun, U., Hentrich, C., & Dustdar, S. (2007). Modeling process-driven and service-oriented architectures using patterns and pattern primitives. *ACM Transactions on the Web*, *1*(3), 14–es. http://doi.org/10.1145/1281480.1281484

Zdun, U., Kircher, M., & Völter, M. (2004). Remoting patterns: Design reuse of distributed object middleware solutions. *IEEE Internet Computing*, *8*(6), 60–66. http://doi.org/10.1109/MIC.2004.70

All links were last accessed on 2017-09-28.

## 6. About the authors

Michael FALKENTHAL is a research associate and Ph.D. student at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. He studied business information technology at the Universities of Applied Sciences in Esslingen and Reutlingen focusing on business process management, services computing and enterprise architecture management. Michael gained experience in several IT transformation and migration projects at small- to big-sized companies. His current research interests are fundamentals on pattern language theory, cloud computing and the internet of things.

Uwe BREITENBÜCHER is a research staff member and postdoc at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research vision is to improve cloud application provisioning and application management by automating the application of management patterns. Uwe was part of the CloudCycle project, in which the OpenTOSCA Ecosystem was developed. His current research interests include cyber-physical systems, patterns, and microservices.

Frank LEYMANN is a full professor of computer science and director of the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research interests include service-oriented architectures and associated middleware, workflow- and business process management, cloud computing and associated systems management aspects, and patterns. Frank is co-author of more than 300 peer-reviewed papers, more than 40 patents, and several industry standards. He is on the Palsberg list of Computer Scientists with highest h-index.