Institute of Architecture of Application Systems

Leveraging Pattern Applications via Pattern Refinement

Michael Falkenthal[1], Johanna Barzen[1], Uwe Breitenbücher[1], Christoph Fehling[1], Frank Leymann[1], Aristotelis Hadjakos[2], Frank Hentschel[3], Heizo Schulze[4]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany
<lastname>@iaas.uni-stuttgart.de

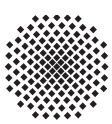
[2]Center of Music and Film Informatics, University of Music Detmold, Germany
hadjakos@hfm-detmold.de

[3]Musicological Institute, University of Cologne, Germany
frank.hentschel@uni-koeln.de

[4]Center of Music and Film Informatics, University of Applied Sciences Ostwestfalen-Lippe, Germany
heizo.schulze@hs-owl.de

BibTeX:

```
@inproceedings{Falkenthal2015,
     author     = {Falkenthal, Michael and Barzen, Johanna and
                   Breitenb{\"u}cher, Uwe and Fehling, Christoph and
                   Leymann, Frank and Hadjakos, Aristotelis and
                   Hentschel, Frank and Schulze, Heizo},
     title      = {Leveraging Pattern Application via Pattern Refinement},
     booktitle  = {Proceedings of the International Conference on Pursuit
                   of Pattern Languages for Societal Change (PURPLSOC)},
     pages      = {38--61},
     year       = {2016},
     publisher  = {epubli GmbH}}
```

Universität Stuttgart
Germany

PURPLSOC

Leveraging Pattern Applications via Pattern Refinement

Falkenthal, Michael; Barzen, Johanna; Breitenbücher, Uwe; Fehling, Christoph; Leymann, Frank, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, Stuttgart, Germany, {lastname}@iaas.uni-stuttgart.de

Hadjakos, Aristotelis, Center of Music and Film Informatics, University of Music Detmold, Hornsche Str. 44, Detmold, Germany, hadjakos@hfm-detmold.de

Hentschel, Frank, Musicological Institute, University of Cologne, Albertus-Magnus-Platz, Cologne, Germany, frank.hentschel@uni-koeln.de

Schulze, Heizo, Center of Music and Film Informatics, University of Applied Sciences Ostwestfalen-Lippe, Liebigstraße 87, Lemgo, Germany, heizo.schulze@hs-owl.de

Abstract:

In many domains, patterns are a well-established concept to capture proven solutions for frequently reoccurring problems. Patterns aim at capturing knowledge gathered from experience at an abstract level so that proven concepts can be applied to a variety of concrete, individual occurrences of the general problem. While this principle makes a pattern very reusable, it opens up a gap between the (i) captured abstract knowledge and the (ii) concrete actions required to solve a problem at hand. This often results in huge efforts that have to be spent when applying a pattern as its abstract solution has to be refined for the actual, concrete use cases each time it is applied. In this work, we present an approach to bridge this gap in order to support, guide, and ease the application of patterns. We introduce a concept that supports capturing and organizing patterns at different levels of abstraction in order to guide their refinement towards concretized solutions. To show the feasibility of the presented approach, we show how patterns detailing knowledge at different levels of abstraction in the domain of information technology are interrelated in order to ease the labor-intensive application of abstract patterns to concrete use cases. Finally, we sketch a vision of a pattern language for films, which is based on the presented concept.

Keywords: *Pattern Refinement; Pattern Application; Cloud Computing Patterns; Costume Patterns*

ISSN (tba)
www.purplsoc.org

1. Introduction

Christopher Alexander explains the essential characteristic of patterns by the following definition: „*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*“ (Alexander, 1979). Hence, pattern authoring focuses on abstracting the essence of concrete problems and solutions within a specific context to formulate reusable knowledge into documents. Beyond that, pattern authoring is supported by systematic approaches, which define methods to collect problem and solution knowledge in order to organize it into patterns and pattern languages (Fehling et al. 2014; Barzen & Leymann, 2015). Such approaches are additionally supported and eased by pattern repositories, which are IT-based libraries – often realized as wiki systems – to author, store, edit, and search patterns (Fehling et al., 2015; Reiners, 2013; Cunningham & Wehaffy, 2013). Such systems typically enable to interrelate patterns with each other to form *pattern languages* that enable to combine patterns to proper solutions (Alexander, 1977; Fehling et al. 2015). In summary, pattern research mainly focuses on pattern authoring and pattern organization. However, detailed knowledge about concrete solutions is lost during this authoring process as use case-specific details are not captured by patterns (Falkenthal et al., 2014, 2015). While this principle of abstraction makes patterns very reusable, because patterns preserve the so called *gestalt* of the investigated concrete solutions, nevertheless, it opens up a gap between (i) the captured *abstract knowledge* and (ii) the *concrete actions* required to solve a concrete and specific problem at hand – as depicted by the abstraction gap in Figure 1: the abstract solution of a pattern has to be refined towards the respective concrete use case each time the pattern is applied.

This gap of abstraction seems to be a domain-independent phenomenon due to the above described characteristics of patterns, which leads to abstract descriptions of problems and solutions within a context in any domain. The patterns of Fehling et al. (2014) are examples from the IT-domain of cloud computing, which is a technology that enables to efficiently share computing resources between different applications. They provide patterns, which help

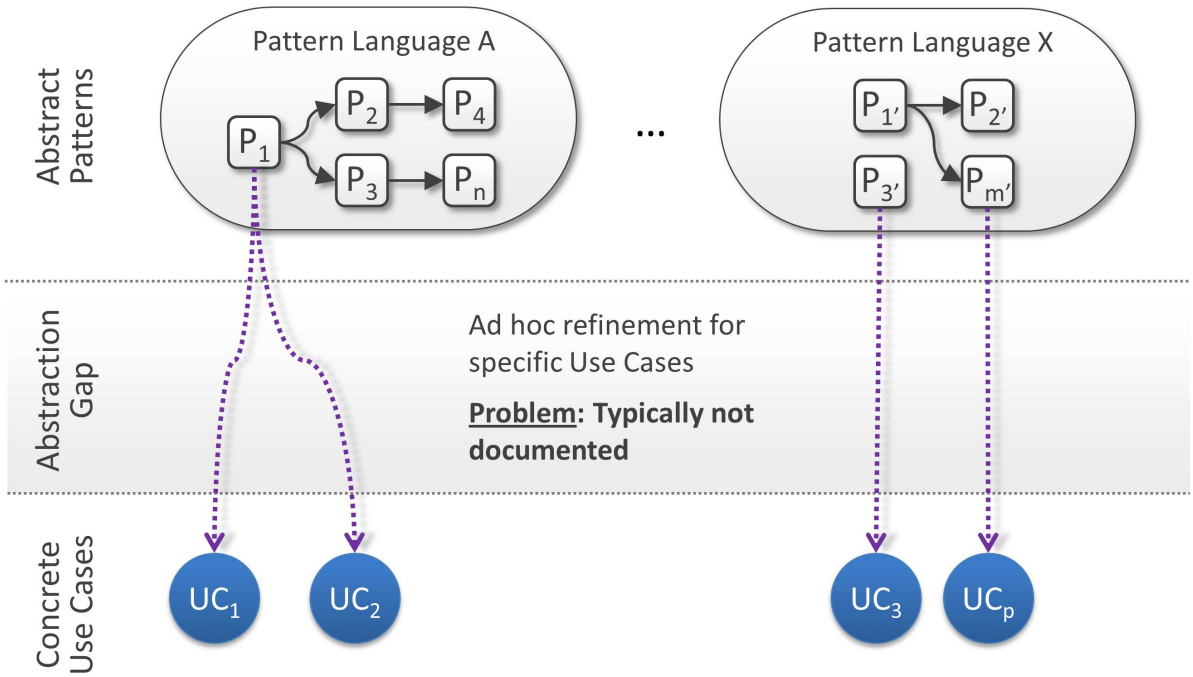


Figure 1: Abstraction Gap

to design applications in order to be streamlined for the technological possibilities of cloud computing. In order to keep these patterns reusable in a manifold of use cases, they are formulated in a technology- and vendor-agnostic manner. Thus, specific knowledge on how to implement a pattern utilizing specific technologies is not provided in detail by these patterns. Software architects are, therefore, not supported at designing software architectures taking account of specific technological constraints, which results in a gap of abstraction between the patterns and their concrete realizations. Hence, a software architect has to transfer the documented abstract solutions of the patterns into technology-specific designs ad hoc, i.e., without guidance and support by the patterns. This *ad hoc refinement* requires either much expertise or immense efforts because additional knowledge beyond the applied patterns has to be researched and adopted to the concrete use case at hand.

For another example, we can observe the gap of abstraction in the domain of costumes in films in the pattern language of Schumm et al. (2012). These patterns provide proven solutions in the area of the so-called vestimentary communication. Vestimentary communication describes the communication taking place through clothes, for example, to communicate a specific character trait by, e.g., the material or the color of the worn costume to the audience. The costume patterns provide general patterns as well as genre-specific patterns, resulting in corresponding different costume pattern languages (Barzen and Leymann, 2015). They need to be combined and refined to support a costume designer in detecting appropriate clothes to communicate the intended character. To get from the abstract solution of a costume pattern to a concrete costume, as stated as the abstraction gap, the domain of film points to a second problem: a film is a complex artifact where many various areas come together and influence each other. The setting, the light, the camera perspective, the film music and the makeup, for example, influence the effect of a costume and the other way around. These difficult dependencies lead to immense efforts in the preparation of a film. So, this example shows that isolating pattern languages covering just one aspect of films from the many other influencing elements leads to a lack of documented deep expertise on how to originate the desired effects in films. Therefore, the lack of systematic, easily accessible, and searchable documentation of such dependencies makes film productions very expensive.

Recapitulating, we can state the following shortcomings in current pattern research: while patterns aim to provide abstract and general solution knowledge, it is time consuming to apply them for concrete use cases and the gap of abstraction between patterns and concrete solutions leads to huge efforts because of the required ad hoc refinements. These issues open up the research question *„How to systematically support, guide, and ease applying abstract patterns to concrete use cases?“.* To answer this question, we argue that pattern languages have to be augmented by explicit relationships between patterns, which indicate the semantics that one pattern refines another pattern towards concrete solutions.

The remainder of this paper is structured as following: we discuss related work in Section 2 and identify the lack of support for actual pattern applications in state of the art pattern research. In Section 3 we introduce a concept to bridge the gap of abstraction in order to ease the application of patterns for concrete use cases. We show the feasibility of the presented approach in Section 4 in terms of a case study focusing patterns of the domain of cloud computing. We sketch a vision for a pattern language of films, which is based on the presented approach in Section 5. Finally, we conclude this paper in Section 6 and show open topics for future work.

2. Related Work

The presented approach bases on work from decades of pattern research and extends the state of the art in this discipline by the capability to manage pattern languages align with the issue on how to efficiently apply patterns considering specific constrains for concrete use cases at hand. To show how the approach presented in this paper extends the state of the art, fundamental related work is discussed in the following and shortcomings are identified, which show the lack of guidance at pattern application.

In his work on pattern languages Alexander (1977) introduces how patterns can be organized as generative chunks of knowledge that inherently connect reoccurring problems with proven and for many use cases reusable solutions. However, the main organization structure of his pattern language is from large-sized things like cities to more fine-grained things like streets, houses and even parts of houses like rooms and configurations of them. Besides, he discusses relations to other patterns within each pattern and describes how patterns are affected from each other and how they solve overall problems in combination. He shows that patterns are powerful means for solving complex problems once they are applied in combination. While he provides an approach to connect patterns on different levels of scale, he lacks guidance on how the presented patterns and, thus, the abstractly provided solutions can be applied for concrete and specific use cases at hand. Further Alexander (1979) describes the power of a pattern language to unfold a new whole solution by applying patterns from a pattern language step by step. Each pattern adds specific new qualities and properties by adapting the structure of the solution. This follows the principle of piecemeal growth and shows how pattern languages unfold their generativity. Nevertheless, Alexander just focusses on how humans can be supported by patterns at the process of creating things and structures of things but his general pattern approach lacks guidance on how already realized concrete solutions can be reused – which is indeed one of the major challenges in the domain of IT reflected by the simplified question on how to develop code so that it can be reused seamlessly in many different programs.

Zimmer (1994) introduces categories of relationships between patterns of object-oriented software design (Gamma et al., 1994). In this pattern collection he identifies the relation categories "X uses Y in its solution", "X is similar to Y" and "X can be combined with Y" between pairs of patterns. These relationship types are used to organize the pattern collection of Gamma et al. in order to indicate, which patterns are related to each other, which ones solve a similar problem but in different valid ways and, finally, which patterns can be applied together to solve an overall problem in combination. Especially the last relationship type is used to indicate that several patterns in combination can be used as larger building blocks to handle design problems, while pattern combinations can encapsulate combined solutions as more abstract building blocks than just single patterns do. Nevertheless, he doesn't provide a means to reflect refinement paths through a collection of patterns towards concrete use cases and, therefore, towards concrete solutions.

Van Welie and van der Veer (2003) present a methodology to organize patterns into a pattern language by focusing on the decomposition of coarse-grained patterns to more fine-grained ones. The level of granularity is defined by the process of user interface design and therefore the hierarchy levels are only domain-dependent. Decomposition in the sense of the presented approach means that high-level respectively coarse-grained design problems are detailed into smaller design problems. Nevertheless, the decomposition focuses on describing more fine-grained structures, which can be used to build up more coarse-grained

ones, i.e., more fine-grained patterns are building blocks from which other patterns can be built. Thus, the presented decomposition does not consider concrete refinements towards pattern implementations for concrete use cases at hand.

Salingaros (2000) describes how patterns are formulated on different levels of granularity align with systems theory. Patterns on more detailed levels act as artifacts, which patterns of more coarse-grained levels are made of. Thus they do not refine solution concepts of coarse-grained patterns towards concrete implementations but show how details of the abstract solution can be described as more fine-grained patterns.

Kubo et al. (2007) introduce a metric to determine the abstraction level of patterns in order to support users to find the most relevant pattern for their problem at hand. Further, besides "Uses" and "Provides Context" they use "Refines" semantics between patterns to distinguish the abstraction levels of different patterns, i.e., these semantics are used to specify the distance between patterns regarding their level of abstraction. Nevertheless, they do not introduce an approach on how to specify refinement by means of semantic links between patterns nor they provide a technical implementation in order to make pattern languages navigable along semantic links to ease the elaboration of concrete realizations of patterns.

Mullet (2002) discusses the organization of patterns within pattern languages by means of different relationship types. Besides the relationship types "Aggregation", which is used to indicate "has-a" relationships between patterns and the general "Association" relationship type, which covers all other non-hierarchical dependencies between patterns, they introduce the type "Derivation". Derivation corresponds to an "is-a" relationship, and, thus, covers semantics of pattern specializations. While specialized patterns inherit functional and non-functional properties described by the more-general pattern and extend them by new properties such specializations do not translate abstract solution principles towards concrete implementations.

Hallstrom and Soundarajan (2009) introduce the concept of design refinement in the IT-domain of object-oriented design and development. They introduce abstraction concepts to formalize patterns and more specialized sub-patterns. The correctness of software designs using patterns and sub-patterns as guidelines is provable because pattern requirements and behavioral guarantees are defined by pattern contracts and subcontracts, respectively. Considering pattern contracts and subcontracts, design refinement leads to pattern hierarchies, which lead from abstract and general patterns to more specialized ones. Nevertheless, specialization in the context of their work leads to variants of solutions in sub-patterns, which are related to general solution concepts in an abstract pattern but the actual refinement towards concrete and specific implementation is not addressed. Further, the approach is tailored to tackle the problem of correctness of design and software and, thus, is not formulated and validated to be feasible for patterns in other domains, i.e., the question is left open if the presented abstraction concepts and contracts are appropriate in general.

Kohls (2011a, 2011b, 2012) provides a detailed discussion on the structure and qualities of patterns and pattern languages. The key elements of patterns – context, forces, problem, solution and consequences – are clearly described and translated into the metaphor of a path. He discusses the qualities of patterns and pattern languages and especially the concepts of wholeness and gestalt utilizing that metaphor. Patterns are described to capture knowledge on a mid-level of abstraction, which means that they are instructive enough to elaborate solutions and, coincidently, are generic enough to be applicable for designing

solutions for many different specific use cases. However, the work does not discuss how already existing pattern languages, which are authored on different levels of abstraction can be linked in order to enable navigation from more abstract patterns to concretized ones in order to guide the design of solutions in a more specific context.

Noble et al. (2006) discuss patterns from a semiotics point of view. They introduce that patterns can be grasped as signs and focus on how meaning of patterns and references between patterns are communicated between recipients. Transferred to the domain of IT signs connect the intended effects that a programmer wants to realize and the actual realizations in program code. In their approach they address research problems like how to identify differences in patterns that are very similar, how to organize pattern variants that lead conceptually to a similar solution but each having a very distinct gestalt than the others, how to it can be expressed that one pattern can solve more than one problem depending on the context, or how relationships between patterns can be carried out that a clear semantics is achieved. Their work provides a basis for this paper by discussing how one pattern can have a multitude of concrete implementations. So, Noble et al. provide a communication theory-centric basis applied to the object-oriented design patterns by Gamma et al (1993).

Zimmermann et al. (2009) discuss the modeling on architecture decisions from a problem-driven point of view. They formalize model entities in order to support the decision making process by clear semantics. Design *Issues* are the leading modeling elements and can be refined from abstract issues to more fine-grained ones. Thus, they discuss the concept of refinement regarding issues in the decision making process for designing application architectures. Issues in their decision models are closely related to problems and forces that are solved and balanced by patterns. The presented approach in this paper extends the concept of refinement of issues to the structure of patterns, which also incorporates the description of a context and a solution.

The meta pattern language by Meszaros and Doble (1996) guides pattern authors at the writing process of patterns and pattern languages. They describe *Pattern Name*, *Problem, Solution, Context* and *Forces* to be mandatory elements of a pattern so that the main idea carried by a pattern is unambiguously communicated. They further describe optional elements, which can be elaborated if they are helpful to understand the essence of the pattern and to provide guidance for the application of a pattern. The elements *Examples* and *Code Samples* closely related to the concept of concrete solutions as presented in this work. However, they do not discuss a systematic approach on how to support and guide the actual application of a pattern. They also miss a discussion on how to interrelate existing pattern languages, especially, for the case that the pattern languages are authored on different levels of abstraction.

3. A Concept to Close the Abstraction Gap

In this work, we answer the raised research question by generalizing findings from the domain of information technology. In this domain, different pattern languages exist, which capture proven knowledge for different contexts and on different levels of abstraction. For the sake of comprehensibility, we foster the concept of pattern refinement by a definition of the term *refinement* in the next subsection, followed by an explanation on how patterns and refined patterns can be linked. Finally, this enables to select a collection of patterns from different levels of abstraction to ease and guide their realization to concrete solutions.

3.1. A View on Pattern Refinement

The approach presented in this paper is based on refining proven solution knowledge from abstract patterns towards concrete solutions. In our former work (Falkenthal et al., 2014, 2015), we introduced the term *Solution Implementation* to circumstantiate a concrete realization of a solution described in a pattern. For the sake of a better interdisciplinary understandability, we revise the IT-inspired term *implementation* and make a rename in the present work to *Concrete Solutions.* Consequently, we define a concrete solution to be an individual, use case-specific realization of the abstract, generic solution principles described by a pattern.

As described above, the abstraction gap between patterns and concrete solutions often leads to time consuming efforts when patterns have to be applied to concrete use cases at hand. However, in the domain of IT, pattern languages exist, which present proven solutions on different levels of abstraction, i.e., there are pattern languages, which provide technology agnostic solution principles (Fehling et al., 2014) while other pattern languages tackle similar topics but describe the solutions considering specific IT environments and technologies (Amazon Web Services, 2012; Microsoft, 2014). To grasp the terminus *refinement* in this constellation, for this discussion we reduce the structure of patterns to a minimal canonical form consisting of the three parts *problem*, *context* and *solution*. While the problem part provides information about the problem, which is solved by the pattern, the context part describes under which circumstances the pattern is applicable and which forces influence the elaboration of concrete solutions. Finally, the solution part of a pattern describes proven solution principles in an abstract textual and human readable form. Since the solution part of a pattern summarizes the essence of a multitude of concrete solutions, i.e., the solution principles that have proven to be good in practice, this opens a whole solution space of concrete solutions, which can be constructed. This characteristic is pointed out by the phrase *"…you can use this solution a million times over, without ever doing it the same way twice*" by Alexander (1979). Therefore, we can state that the size of the solution space of a pattern is directly related to the level of abstraction of the pattern's solution principles: the more abstract the documented problem, context, and solution, the more use cases can be solved. This is because at higher levels of abstraction the number of constraints, which limit the applicability of a pattern, decreases. Kohls (2012) refers to this as the "*abstraction to emergent qualities*". He points out that more abstract solution descriptions are less instructive

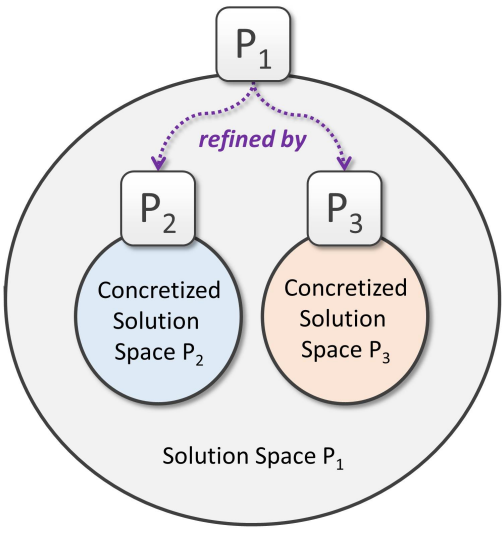


Figure 2: Concretized patterns and solution spaces

but open up more choices at realizing concrete implementations of a pattern. Vice versa, the size of the solution space is significantly smaller if a pattern provides solution principles that are closer to a concrete solution or use case, respectively. This results from *implementation-specific* knowledge and, thus, *implementation-specific constraints* that are part of the more concretized pattern. Nevertheless, this is exactly the kind of knowledge that can be used to close the abstraction gap towards concrete and use case-specific realizations.

In consequence, pattern refinement means that a pattern's problem, context, and solution are getting more specific and precise towards concrete realizations and use cases. Nevertheless, the core of the problem and the provided solutions remain the same. This is conceptually depicted in Figure 2, where abstract pattern P_1 opens a solution space and the patterns P_2 and P_3 refine P_1 to more concrete solutions and, thus, open smaller but *concretized solution spaces* within the one of P_1. Therefore, P_1 can be applied to more use cases than P_2 and P_3, while they document and provide knowledge about how to realize solutions for certain concrete use cases at hand.

3.2. The Concept of Refinement Links

If one pattern refines another pattern, this refinement can be documented by a semantic link, which determines a navigation path from the abstract solution of the former pattern to the more concretized solution of the latter one. This improves the usability of patterns, which were authored isolated from each other – especially, if they are part of different pattern languages. Thereby, formerly isolated knowledge can be connected via explicit refinement links, which bridge the abstraction gap in order to avoid manual, unguided, and ad hoc refinements.

This is conceptually depicted in Figure 3, where patterns (P_i) are organized into pattern languages (L_j). For the sake of simplicity, the depicted layers illustrate that the pattern languages provide solution knowledge on different levels of abstraction - even though pattern languages are not necessarily authored and stuck to just one specific abstraction level.

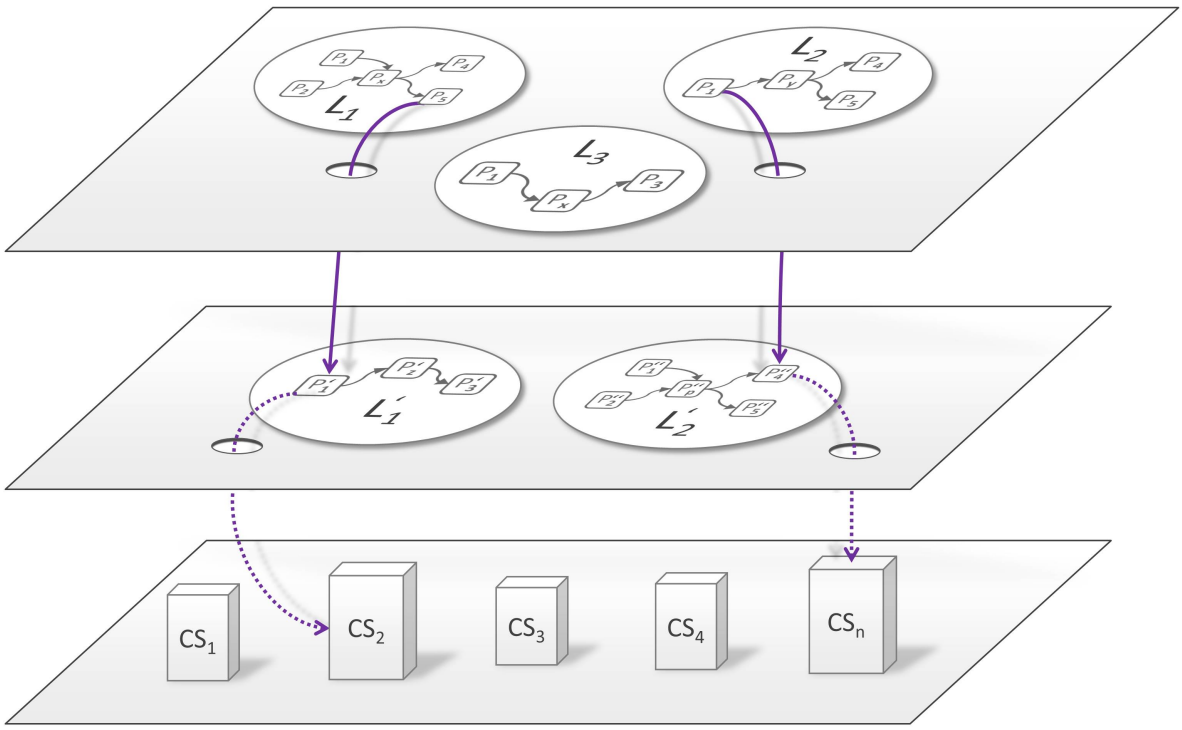


Figure 3: Patterns on different levels of abstraction connected by refinement links

Refinement links from abstract patterns of the languages L_1 and L_2 to more concretized patterns of the languages L'_1 and L'_2 are illustrated by solid arrows, while dotted arrows indicate refinements that have to be done manually.

In this scenario, the pattern languages L'_1 and L'_2 bridge the abstraction gap by providing concretized solution knowledge to refine the more abstract patterns of L_1 and L_2 towards concrete solutions (*CS*) for use cases at hand. So, a pattern reader can choose pattern P_5 of pattern language L_1, which provides a huge solution space and, therefore, only abstract solution principles. Then, he navigates to P'_1 in L'_1, which provides more concretized solution knowledge to elaborate the concrete solution CS_2 that is actually the solution of the specific use case. Nevertheless, it is not obligatory that there is refinement knowledge in the form of more concretized patterns, as depicted by the missing refinement links between other patterns. Especially, there is no refinement guidance for pattern language L_3 as there are no concretized patterns that refine the ones in L_3.

3.3. Easing Pattern Application by Inter-Pattern Language Solution Graphs

The concept introduced above can be leveraged to ease the application of patterns for concrete use cases. Patterns are often used in combination to elaborate concrete solutions, which is supported by references between patterns in different pattern languages. Thus, a pattern language can be grasped as a graph whose nodes represent patterns and whose edges represent references between the patterns. So, a reader can select a pattern that solves a particular part of the problem at hand and then navigate to related patterns, which provide additional solutions that typically need to be combined to elaborate a proper concrete solution. Navigating and selecting patterns in this fashion results in a subgraph, which we call *Solution Graph*. This concept extends the approach of solution paths by Zdun (2007), because subgraphs are not limited to represent just sequences of patterns, but also branching paths within a pattern language.

The concept of solution graphs is specifically important when we extend the expressiveness of pattern languages by the formerly introduced concept of refinement links in order to enable the navigation towards concretized solutions, too. As depicted in Figure 4, references between patterns enable to navigate within a pattern language of a specific level of abstraction. We call this kind of navigation *horizontal navigation*, since navigation remains on the same level of abstraction. Refinement links also support to navigate towards concretized patterns in other languages, which provide refined solution knowledge. This kind of navigation is called *vertical navigation*, since it targets towards concretized solutions. Therefore, horizontal navigation enables to unfold the power of pattern languages to create a "*new whole out of incoherent pieces*" as Alexander (1964) points out. This means, the abstract patterns allow for designing a new solution conceptually, whereby each pattern adds particular new qualities and, therefore, enables piecemeal growth of the solution. On the other hand, the vertical navigation enables to replace abstract parts of a solution by more concrete ones towards concrete solution artifacts. Thus, the *wholeness* of the abstractly designed solution and its *gestalt* is concretized towards an overall concrete solution, which itself is a new whole.

Given the situation that P_5 of pattern language L_1, P_1 of L_3, and P_1 of L_2 are selected by a reader to solve his problem at hand, he can also navigate to the more concretized patterns P'_1 of L'_1 and P''_4 of L'_2 to be guided towards concrete solutions. The selected patterns are part of the solution graph indicated by the surrounding area in Figure 4.

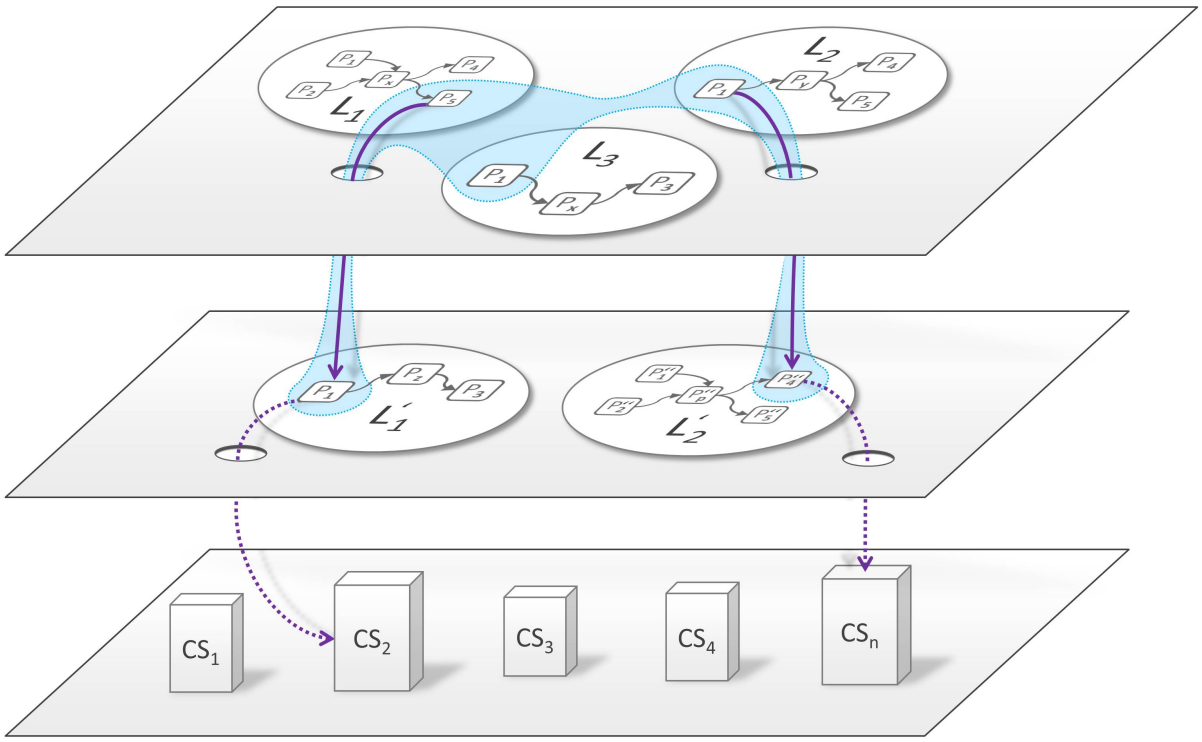


Figure 4: Solution graph spanning multiple pattern languages

Since the refined patterns P'$_1$ of L'$_1$ and P''$_4$ of L'$_2$ are part of the solution graph, this example shows that following refinement links leads to solution graphs that span several pattern languages on different levels of abstraction. Thus, besides the combination of patterns on the same level of abstraction, also refinement of abstract patterns towards concretized ones, and, therefore, more concretized solutions, is represented by the depicted solution graph. Nevertheless, Figure 4 shows that refinement knowledge is not necessarily available for each pattern of a pattern language, which is indicated by the fact that there is no refinement link, e.g., from P_1 of L_3 to a concretized pattern. This means that the solution principles provided by this pattern have to be manually and ad hoc refined to elaborate a concrete solution. But since refinement links are available to navigation from the other two abstract patterns of the solution graph to concretized solutions, the application of the patterns within the solution graph is at least partly guided and eased.

4. Case Study: Cloud Computing Pattern Refinement

We discovered the above presented concepts of pattern refinement and refinement links by investigating pattern languages in the IT-domain of cloud computing. There, the pattern language of Fehling et al. (2014) describes problems and solutions in an abstract, cloud provider- and technology-independent manner. Besides these patterns, the cloud providers Amazon Web Services (2012) and Microsoft (2014) developed their own pattern languages that focus on realizing applications in their concrete cloud environments. Their languages are, therefore, more concrete in contrast to Fehling's abstract pattern language: indeed, they describe similar problems, contexts, and solutions, but on a significantly more detailed level with respect to their proprietary offerings in order to bind customers. Thereby, the ability to reuse these patterns is decreased in comparison to the patterns of Fehling et al. (2014) because the contexts and solutions of these patterns are concretized to the technical circumstances of the respective cloud environments. This reveals that these provider-specific

patterns are not applicable for developing applications for other cloud environments. However, they capture knowledge to realize abstract patterns of Fehling et al. in the environments of Amazon Web Services and Microsoft. Thus, this scenario provides an interesting example of pattern languages on different levels of abstraction, which may be linked to ease applying an abstract pattern by a stepwise refinement via other pattern languages towards concrete solutions.

In total, we discovered 16 patterns in the pattern languages of Amazon Web Services and Microsoft that provide guidance for translating the abstract solution concepts of Fehling et al. (2014) into technological building blocks (and combinations of these) within the respective cloud environment. In Figure 5, we illustrate this exemplarily by the four patterns: *Elastic Load Balancer (ELB)* and *Stateless Component (SC)* from the pattern language of Fehling et al. (2014) as well as *Scale Out Pattern (SO)* and *State Sharing Pattern (SSh)* from the respective pattern language of Amazon Web Services (2012).

The patterns ELB and SO tackle the problem on how the number of application components can be dynamically provisioned and also decommissioned to process changing workloads. This means that, based on measured workloads, new instances of an application component are launched in order to spread and balance high workloads among them. Besides, instances can also be stopped and terminated in situations of low workloads. This behavior is called *elastic scaling* in the terminology of cloud computing. The abstract pattern ELB describes these principles technology-agnostic and does not provide a solution with respect

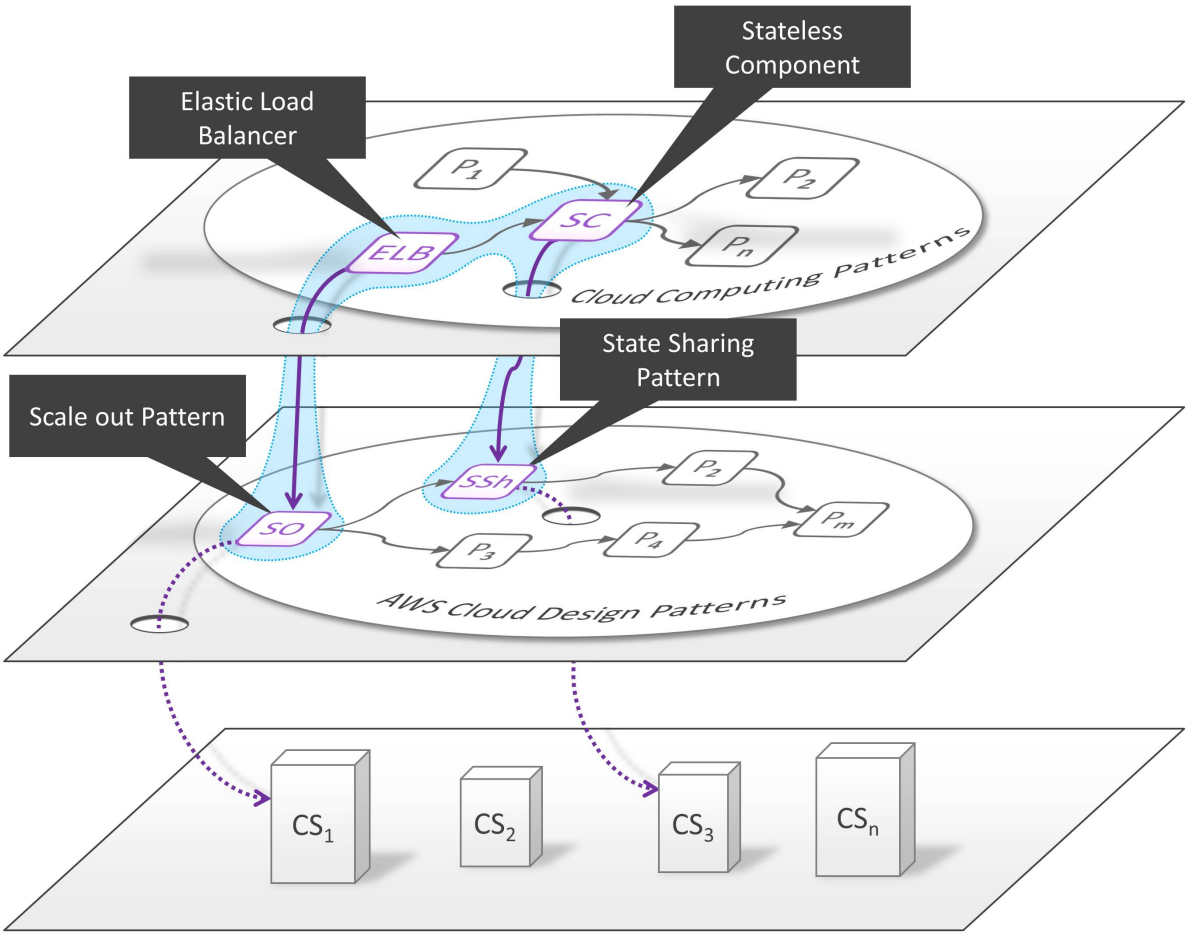


Figure 5: Cloud computing patterns on different levels of abstraction connected by refinement links

to specific cloud offerings like those of Amazon Web Services or Microsoft. On the other hand, SO refines the general solution concepts of ELB into terms of the Amazon Web Services Cloud Offering.

The SC pattern describes how application components should manage state in a cloud application to ease the above-mentioned scaling functionality and enable failure resiliency. The session state, which is the state of the interaction with the component, should be provided with every request to the component. The application state, which is the data handled by the application, such as a customer database, should be handled outside of the component. Preferably, this application state is handled in provider-supplied storage offerings. This ensures that the application components themselves become stateless, thus, they do not manage an internal state that would have to be synchronized or extracted upon component instance provisioning and decommissioning, respectively. The SSh pattern details this concept of stateless components in scope of Amazon Web Services. Components take the form of virtual servers that rely on key-value stores provided by Amazon for external state management.

In detail, the refinement of the Elastic Load Balancer (ELB) pattern to the Scale out (SO) Pattern was identified by analyzing the textual description of both pattern documents for similarities, which are exemplarily shown in Figure 6. The abstractly described concept of a *Load Balancer* – that is the component, which spreads workload across a number of application instances – in the Elastic Load Balancer pattern on the left, is refined in Scale out Pattern to the concept of *Elastic Load Balancer Service*, which is a service in the ecosystem of Amazon Web Services to spread and balance workload across a number of application instances. So, the Scale out Pattern provides a concretized solution about how to implement load balancing at Amazon Web Services. The same applies to the concept of *Monitoring* actual workloads in the Elastic Load Balancer pattern. It is refined in Scale out Pattern by the technological approach of *Cloud Watch*, which is a service of Amazon Web Services that provides capabilities to monitor actual workloads. Finally, the abstract Elastic Load Balancer pattern describes an *Elastic Infrastructure*, which provides functionality to automatically provision and decommission instances of application components.

In the Scale out Pattern, this concept is refined by the description on how *EC2* (service to provision compute resources within the Amazon cloud), the *Auto Scaling* service (service to configure automatic elastic scaling) and *AMIs* (Amazon Machine Images, that contain the actual application components), have to be combined in order to elicit elastic scaling in the cloud environment of Amazon.

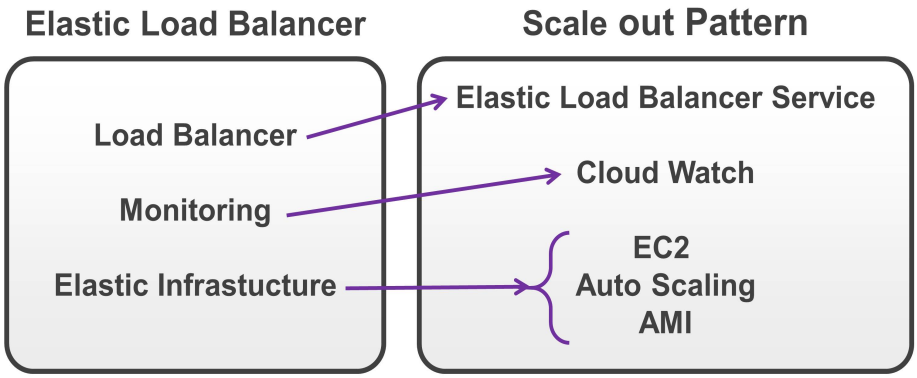


Figure 6: Corresponding concepts in the ELB Pattern and the SO Pattern

To enable navigation from the abstract pattern Elastic Load Balancer towards the concretized pattern Scale out Pattern, we implemented a refinement link between these patterns in our pattern repository PatternPedia (Fehling et al., 2015). There, we collected the abstract cloud computing patterns of Fehling et al. (2014) and also the more concrete patterns of Amazon Web Services (2012). The result is depicted exemplarily in Figure 7, where Elastic Load Balancer is shown as a wiki page within PatternPedia. On the right, all links to other patterns are listed within the groups "*Related To*", "*Consider Next*", "*In Context Of*" and "*Refined By*". Thus, the refinement link to Scale out Pattern is listed under "*Refined By*", which indicates the specific refinement semantics of the link. Technically, these links are implemented by the Semantic MediaWiki (2015) technology, which provides means to define semantic links in PatternPedia.

Since patterns are stored as *wiki pages*, which are build upon the concept of web pages, refinement links are implemented using the technologies of the semantic web – *RDF-Schema* (W3C, 2014) and *RDF* (W3C, 2014a). Via RDF-Schema, patterns can be defined as typed *pattern resources* and refinement links as properties of the type "*refinedBy*" of these resources. Using RDF, a refinement link between two patterns can be expressed as a RDF triple. The abstract pattern is the *subject* of the RDF triple while the concretized pattern is the *object*. Both are identified by unique URIs that correspond to the URLs of the wiki pages of the respective pattern. Conceptually, the refinement link is the *predicate* of the RDF triple, which indicates the relation between *subject* and *object*. It is represented as a *property* of the type "*refinedBy"*, which is part of the subject pattern and contains a reference to the concretized pattern identified by its URI. Since this URI corresponds to the URL of the wiki page of the concretized pattern, the "*refinedBy*" relation can be rendered by Semantic MediaWiki (2015) as a hyperlink, which is indicated in Figure 7. Thus, the creation of

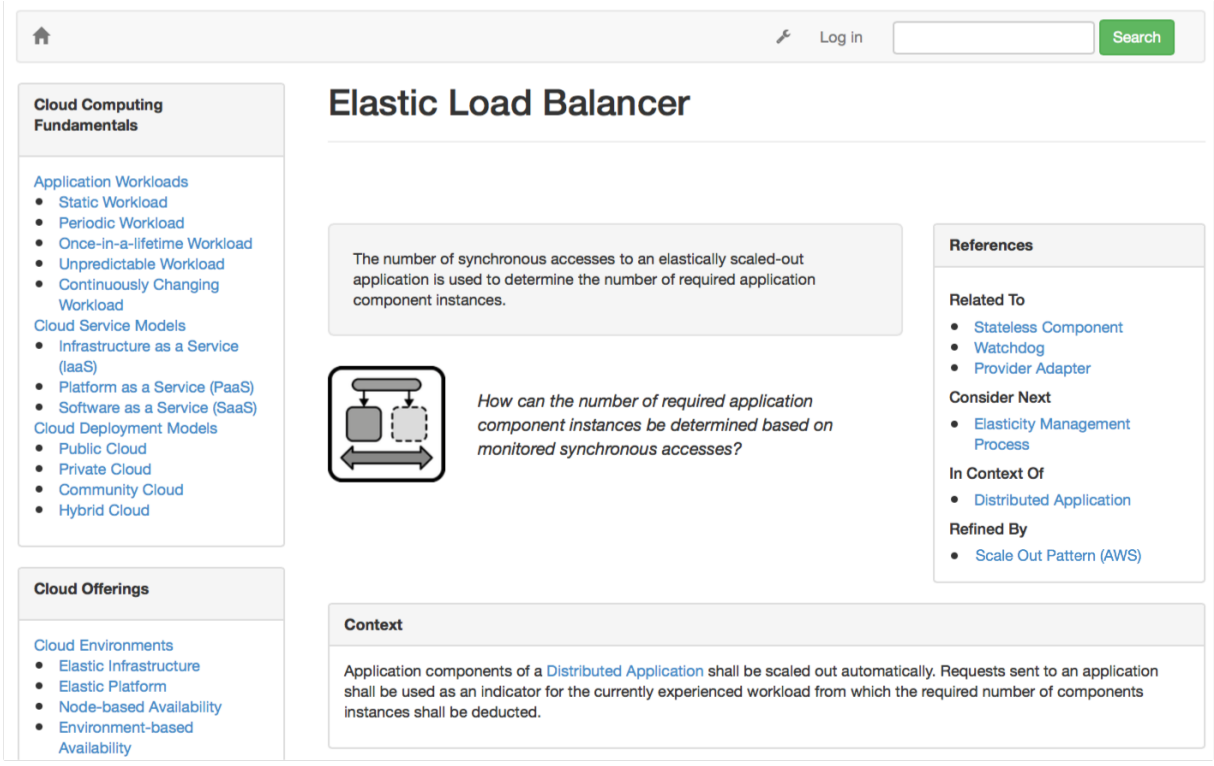


Figure 7: Refinement link from Elastic Load Balancer to Scale out Pattern in PatternPedia

semantic links between patterns is fully supported by Semantic MediaWiki (2015), and, therefore, shows exemplarily how the above introduced concept of refinement links is implemented in PatternPedia.

5. Vision: Towards a Pattern Language for Films based on Costumes, Setting, and Music in Films

To prove the generality of our approach and our method, we present how it can be applied to a domain totally different from cloud computing: the domain of films. Because a film is a complex artifact, whose expressional power arise from many diverse areas like the sound and music, the setting, the light, the costumes, the camera perspective and many more, capturing of knowledge about how films reach an effect on a recipient is a difficult task. Because each of these areas focuses on rather diverse subjects, actions and topics, it is challenging to describe their core issues in a uniform manner that allows comparison and combination of this knowledge. Therefore, the concept of patterns gives a powerful means to identify the relevant parameters, to capture this knowledge, and to store it in a reusable way for others working on films.

The costume pattern language of Schumm et al. (2012) gives an example on how patterns can extract and store the knowledge provided by concrete films, especially for the domain of costumes. The costume pattern language aims at capturing the established conventions in terms of an abstract solution about how, for example, stereotype clothes are used to communicate a certain character to the audience.

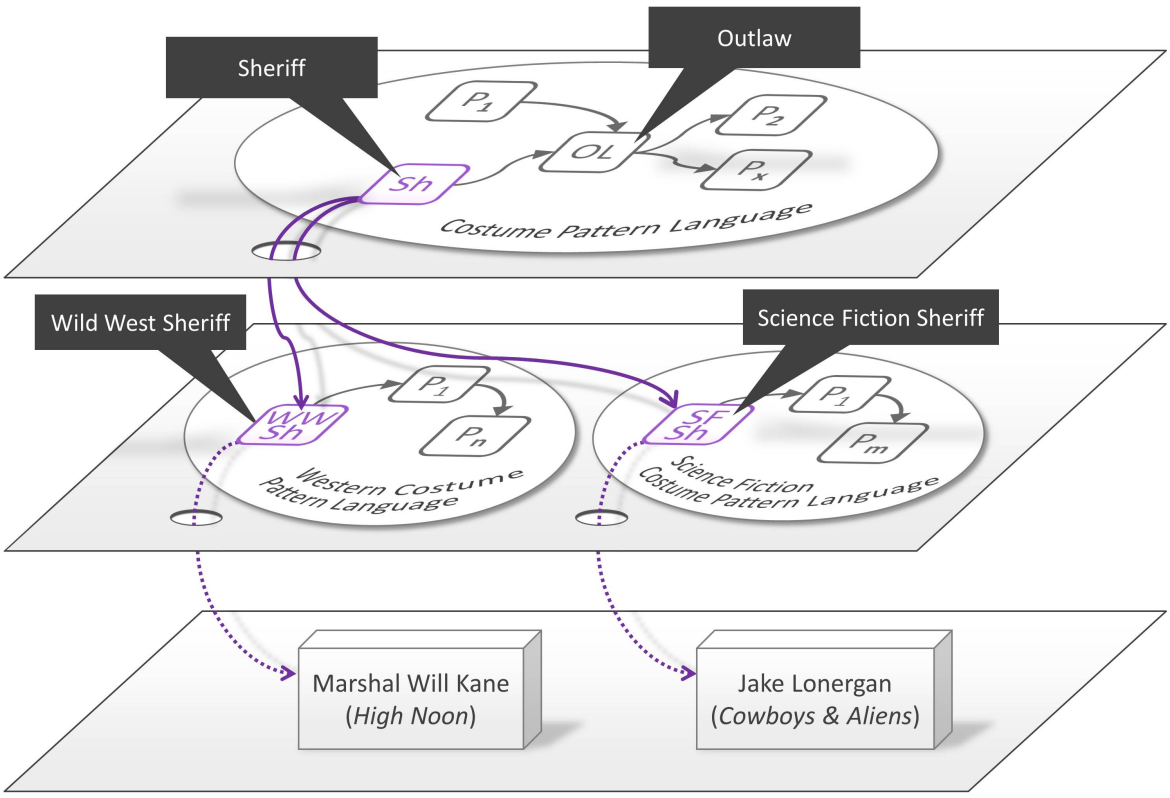


Figure 8: Costume patterns on different levels of abstraction connected by refinement links

When taking a closer look at the Wild West Sheriff Pattern introduced by Schumm et al. (2012), it becomes obvious that the concept of refinement links towards a concrete costume solution is also applicable and valuable in the domain of films. As depicted in Figure 8, there are also different levels of abstraction in the various costume pattern languages in the domain of costume in films. On the highest level the *Costume Pattern Language* is located, containing the highly abstract costume patterns like *Sheriff Pattern* or *Outlaw Pattern*. But, because each film genre has its own Costume Pattern Language (as assumed in Barzen and Leymann (2015)), there are quite different Sheriff Patterns in the genre-specific pattern languages. The genre-specific pattern languages like the *Western Costume Pattern Language* or the *Science Fiction Costume Pattern Language* are less abstract than the Costume Pattern Language and are, therefore, located at the second level. The Sheriff Pattern, thus, can be refined by the *Wild West Sheriff Pattern* from the Western Costume Pattern Language or the *Science Fiction Sheriff Pattern* from the Science Fiction Costume Pattern Language if needed, as indicated by the refinement links in Figure 8. This simplifies the navigation to a suitable concrete solution in terms of a real costume someone wears in a film: The Sheriff Pattern refined by the Wild West Sheriff Pattern can be refined a second time to the concrete solution. This could be, for example, by refining the Wild West Sheriff Pattern to the concrete solution of *Marshal Will Kane* (Gary Cooper) in the Film *High Noon* (1952, Director: Fred Zinnemann) or by refining the Science Fiction Cowboy Pattern to *Jake Lonergan* (Daniel Craig) in the film *Cowboys & Aliens* (2011, Director: Jon Favreau). Using refinement links to navigate through different pattern languages closes the abstraction gap between a very high level and abstract solution to a concrete solution to support, for example, a costume designer to find the right costumes for a certain character.

But, as stated above, a film is a complex artifact, whose expressional power arises through many diverse areas, and the costume area is just one of them. Even more, the expressional power arises through the interaction of these diverse areas, which requires to not only capture the knowledge of a single area, but of many of them as well as their dependencies. For example, a red dress worn in a blue room has a signaling effect, while a character wearing a blue dress in a blue room rather disappears and has a totally different effect than the red dress. The same is true for combining, for example, a forest setting with different music. The effect of the forest can range from seeming a rather safe place to being a very dangerous place according to the music. Therefore, in our vision the diverse pattern languages need to be combined, not only by refinement, but also in dependencies to each other. Therefore, we plan to store all these pattern languages in PatternPedia (Fehling et al., 2014) in order to enable semantic links between patterns from the respective areas. Especially investigating refinements of combinations of these patterns, and representing them in PatternPedia seems to be a promising means to support and ease pattern application from different areas in the domain of films.

6. Conclusion & Future Work

In the present work we introduced the concept of pattern refinement that eases pattern application. Therefore, we showed how pattern languages on different levels of abstraction can be interrelated by refinement links and, thus, navigation from abstract patterns towards concretized solutions is enabled. We further extended the concept of solution paths within pattern languages to solution graphs, which represent a selection of patterns from different pattern languages, also including refined patterns for solving problems at hand. Based on these concepts, we finally sketched a vision of a pattern language for films, which inherently connects solution knowledge from different disciplines in the domain of films.

To realize our vision, we have to do follow up research on how combination of patterns from different pattern languages can be supported. We also work on a more detailed description of the concept of pattern refinement, especially, in respect to situations where the application of a combination of more specific patterns provide a refinement of an abstract pattern. We are also going to further develop our pattern repository PatternPedia to support the selection of concrete solutions from patterns in an unelaborate way. To approach the vision of a pattern language of films that combines the different art domains and their dependencies, we started to work (in addition to our continuing work on costume patters) on music patterns and are currently developing the basic taxonomies to capture music in its relevant parameters. We also started working on a pattern language for film settings that is based on the tool setscene (HZO Film & Medien, 2014), which already provides numbers of concrete solutions of film settings. We also plan to interweave the pattern languages of different areas of the domain of films and support their application by semantic links within PatternPedia.

7. References

Alexander, C. (1964). Notes on the synthesis of form. Cambridge: Harvard University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jakobson, M., Fiksdahl-King, I., & Angel,S. (1977). A Pattern Language: Towns, Buildings, Construction. Oxford: Oxford University Press.

Alexander, C. (1979). The Timeless Way of Building. Oxford: Oxford University Press, 1979.

Amazon Web Services (2012). AWS Cloud Design Pattern. *http://en.clouddesignpattern.org*.

Barzen, J., & Leymann, F. (2015). Costumes Languages as Pattern Languages. *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014.* Neopubli.

Cunningham, W., Mehaffy, M. W. (2013). Wiki as Pattern Language. Proceedings of the 20th Conference on Pattern Languages of Programs (PLoP 2013). Articel No. 32. New York: ACM.

Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., & Leymann, F. (2014). From Pattern Languages to Solution Implementations. *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014). Xpert Publishing Services*.

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

Fehling, C., Barzen, J., Breitenbücher, U., & Leymann, F. (2014). The Process of Pattern Identification, Extraction, and Application. *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP 2014)*. Accepted for publication.

Fehling, C., Barzen, J., Falkenthal, M., & Leymann, F. (2015). PatternPedia - Collaborative Pattern Identification and Authoring. *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014.* Neopubli.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., & Arbitter, P. (2014). *Cloud Computing Patterns*. Wien: Springer.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design patterns: Abstraction and reuse in object-oriented design. *Proceedings of ECOOP'93* (pp. 406-431). Berlin: Springer.

Hallstrom, J. O., & Soundarajan, N. (2009). Reusing Patterns through Design Refinement. In Formal Foundations of Reuse and Domain Engineering (pp. 225-235). Berlin: Springer.

HZO Film & Medien (2014). Setscene - The Compendium of Film Sets. *http://www.setscene.org*.

Kohls (2011a). The structure of patterns. *Proceedings of the 2010 Conference on Pattern Languages of Programming (PLoP 2010)*. New York: ACM.

Kohls (2011b). The structure of patterns – Part II – Qualities. *Proceedings of the 2011 Conference on Pattern Languages of Programming (PLoP 2011)*. New York: ACM.

Kohls (2012). The path to patterns – introducing the path metaphor. *Proceedings of the 17th European Conference on Pattern Languages of Programs (EuroPLoP 2012).* Article No. 9. New York: ACM.

Meszaros, G., & Doble, J. (1996). MetaPatterns: A Pattern Language for Pattern Writing. The 3rd Pattern Language of Program congress. Monticello.

Microsoft (2014). Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. https://msdn.microsoft.com/en-us/library/dn568099.aspx.*

Mullet, K. (2002). Structuring pattern languages to facilitate design. *CHI2002 Patterns in Practice: A Workshop for UI Designers*.

Noble, J., Biddle, R., & Tempero, E. (2006). Patterns as Signs: A Semiotics of Object-Oriented Design Patterns. *International Journal on Communication, Information Technology and Work (Vol. 2 (1))*.

W3C (2014). RDF 1.1 Concepts and Abstract Syntax. *http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/*.

W3C (2014a). RDF Schema 1.1. *http://www.w3.org/TR/rdf-schema/.*

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

Salingaros, N. (2000). The Structure of Pattern Languages. Architectural Research Quarterly (Vol. 4(02)).

Semantic MediaWiki (2015). Semantic MediaWiki. *https://semantic-mediawiki.org*.

Schumm, D., Barzen, J., Leymann, F., Ellrich, L. (2012). A Pattern Language for Costumes in Films. *Proceedings of the 17th European Conference on Pattern Languages of Programs (EuroPLoP 2012)*. New York: ACM.

Van Welie, M., van der Veer, G. C. (2003). Pattern Languages in Interaction Design: Structure and Organization. *Proceedings of Human-Computer Interaction (INTERACT) '03: IFIP TC13 International Conference on Human-Computer Interaction*. IOS Press.

Zdun, U. (2007). Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience* (*Vol. 37*(9)).

Zimmer, W. (1994). Relationships between Design Patterns. Pattern Languages of Program Design. Addison-Wesley.

Zimmermann, O., Koehler, J., Leymann, F., Polley, R., & Schuster, N. (2009). Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules. *Journal of Systems and Software (Vol. 82(8))*. Elsevier.

All links were last accessed on 13.06.2016.

8. About the author/s:

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 as well as cloud computing.

Johanna BARZEN studied media science, musicology and phonetics at the University of Cologne and gained first practical experience while working for some major television channels like WDR and RTL. Next to this she studied costume design at the ifs (international film school Cologne) and worked in several film productions in the costume department in different roles. Currently she is Ph.D. student at the University of Cologne and research staff member at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart doing research on vestimentary communication in films.

Uwe BREITENBÜCHER is a research associate and Ph.D. student at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research vision is to ease the provisioning and management of cloud applications by automating management patterns. Uwe received a diploma in software engineering from the University of Stuttgart.

Christoph FEHLING is a research associate and Ph.D. student at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research interests include IT architecture patterns focused especially on cloud computing. Christoph received a Dipl.-Inf. in computer science from the University of Stuttgart. He is a member of the Hillside Group and author of the book "Cloud Computing Patterns" (Springer, 2014).

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.

Aristotelis HADJAKOS is Professor for Music Informatics at the University of Music Detmold, Germany. He is the head of the Center of Music and Film Informatics, a joint institution of the University of Music Detmold and the University of Applied Sciences OWL, Lemgo. He is conducting research on Human-Computer-Interaction in the area of music and media. His research interests include Digital Humanities in music and film, sensor-based music interfaces and interactivity in music scores.

Frank HENTSCHEL is a full Professor of Musicology at the University of Cologne. He studied musicology, philosophy and German literature in Cologne and London. He received his Ph.D. in musicology in 1999 in Cologne and his habilitation in 2006 in Berlin (Free University). He spent one year at Harvard University Cambridge with a Feodor Lynen fellowship in 2004/2005 and was Professor at the Universities of Jena and Giessen. His recent work focuses on film music and the history of emotions.

Heizo SCHULZE is full Professor for Audiovisual Design at the faculty Media Production at the University of Applied Sciences, Lemgo, Germany. He graduated in Design for Electronic Media. His recent work and research focuses to combine traditional linear media (e.g. film) with the latest interactive and mobile options. He published various essays, papers, linear and nonlinear works, such as films, videos, installations and apps. He is a member of the internal research group Perception Lab and of the Center of Music and Film Informatics, a joint institution with the University of Music Detmold.