Easing Pattern Application by Means of Solution Languages

Michael Falkenthal, Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany,
lastname@iaas.uni-stuttgart.de

Easing Pattern Application by Means of Solution Languages

Michael Falkenthal and Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
Email: {lastname}@iaas.uni-stuttgart.de

***Abstract*—Patterns and pattern languages are a pervasive means to capture proven solutions for frequently recurring problems. They capture the expertise of domain specialists, as well as the essence of concrete solutions in an abstract and generic manner. These characteristics guarantee that patterns and pattern languages are applicable for many concrete use cases. However, due to this nature the knowledge about applying them to concrete problems at hand is lost during the authoring process. The lack of guidance on how to implement a pattern in a specific technical or environmental context leads to immense manual efforts and unnecessary reimplementations of already existing solutions. In our previous work, we presented the concept of linking concrete solutions to patterns in order to ease the pattern application. In this work, we extend this concept and present an approach to organize concrete solutions into Solution Languages, which are means to structure the solution space of a pattern language. We show how Solution Languages can be used to systematically collect specific implementation knowledge to purposefully navigate through a set of concrete solutions to ease and guide the realization of patterns. We validate the approach of Solution Languages in the domain of cloud application architecture and illustrate its technical feasibility by a wiki-based prototype.**

Keywords–Pattern Language; Solution Language; Pattern Application; Solution Selection.

I. Introduction

In many domains, expertise and proven knowledge about how to solve frequently recurring problems are captured into patterns. Originated by Alexander et al. [1] in the domain of building architecture, the pattern concept was also heavily applied in many disciplines in computer science. Patterns were authored, e.g., to support object-oriented design [2], for designing software architectures [3], for human-computer interaction [4], to integrate enterprise applications [5], for documenting collaborative projects [6], or to foster the understanding of new emerging fields like the Internet of Things [7]. Triggered by the successful application of the pattern concept in computer science, it is also gaining momentum in the humanities, especially as a result of collaborative endeavors and research in the field of the *digital humanities* [8].

In general, patterns segment domain knowledge into *nuggets of advice*, which can be easily read and understood. They are interrelated with each other to form pattern languages, which ease and guide the navigation through the domain knowledge. This is often supported by links between patterns, which carry specific semantics that help to find relevant other patterns based on a currently selected one [9]. In previous work, we showed that this principle can be leveraged to organize patterns on different levels of abstraction into pattern languages [10]. *Refinement links* can be used to establish navigation paths through a set of patterns, which lead a user from abstract and generic patterns to more specific ones that, e.g., provide technology-specific implementation details about the problem – often presented as implementation examples. We further showed that also concrete solutions, i.e., concrete artifacts that implement a solution described by a pattern, can be stored in a solution repository and linked to patterns [11] [12]. Thus, we were able to show that pattern-based problem solving is not only limited to the conceptual level, but rather can be guided via pattern refinement towards technology-specific designs and, finally, the selection and reuse of concrete solutions.

However, this approach still lacks guidance for navigation through the set of concrete solutions. Navigation is only enabled on the level of pattern languages, while it is not possible to navigate from one concrete solution to others, due to missing navigation structures. This hinders the reuse of available concrete solutions especially in situations when many different and technology-specific concrete implementations of patters are available. As a result, it is neither easily understandable which concrete solutions can be combined to realize an aggregated solution, nor which working steps actually have to be done to conduct an aggregation. Thus, an approach is missing that allows to systematically document such knowledge in an easily accessible, structured and human-readable way.

Therefore, we present the concept of Solution Languages, which introduces navigable semantic links between concrete solutions. A Solution Language organizes concrete solution artifacts analogously to pattern languages organize patterns. Their purpose is to ease and guide the navigation through the set of concrete solutions linked to patterns of a pattern language. Thereby, knowledge about how to aggregate two concrete solutions is documented on the semantic link connecting them.

The remainder of this paper is structured as following: we provide background information and give a more detailed motivation in Section II. Then, we introduce the concept of Solution Languages and a means to add knowledge about solution aggregation in Section III. We validate our approach by a case study and discuss how presented concept can be applied in domains besides information technology (IT) in Section IV. We show the technical feasibility of Solution Languages by a prototype based on wiki-technology and by implementing the presented case study in Section V. We discuss related work in Section VI and, finally, conclude this work in Section VII by a summary of the paper and an outlook to future work.

II. Background and Motivation

Patterns document proven solutions for recurring problems. They are human-readable documentations of domain expertise. Thereby, their main purpose is to make knowledge about how to effectively solve problems easily accessible to readers. According to Meszaros and Doble [13], they are typically written and structured using a common format that predefines sections such as the *Problem*, which is solved by a pattern, the *Context* in which a pattern can be applied, the *Forces* that affect the elaboration of concrete solutions, the *Solution*, which is a description of how to solve the exposed problem, and a *Name* capturing the essence of a pattern's solution.

Patterns are typically not isolated from each other, but they are linked with each other to enable the navigation from one pattern to other ones, which are getting relevant once it is applied. In this manner, a navigable network of patterns is established – a pattern language. Often, a pattern language is established by referring other patterns in the running text of a pattern by mentioning them. This applies, especially, to pattern languages that are published as a monograph. Using wikis as platforms for authoring and laying out a library of patterns has further enabled to establish semantic links between patterns [6] [14]. This allows to enrich a pattern language to clearly indicate different navigation possibilities by different link types. Such link types can, e.g., state *AND*, *OR*, and *XOR* semantics, describing that after the application of a pattern more than one other patterns are typically also applied, that there is a number of further patterns, which can be alternatively applied, or that there is an exclusive choice of further patterns that can be applied afterwards, respectively [6]. Further, they can tell a reader, e.g., that a pattern is dealing with the equivalent problem of another pattern, but gives solution advice on a more fine-grained level in terms of additional implementation- and technology-specific knowledge [10]. Thus, the navigation through a pattern language can be eased significantly.

Since patterns and pattern languages capture the essence and expertise from many concrete solutions of recurring problems, implementation details, such as technology-specific or environmental constraints, which affect the actual application of a pattern for specific problems at hand, are abstracted away during the pattern authoring process [15] [16]. As a result, this abstraction ensures that the conceptual core ideas of how to solve a problem in a context are captured into a pattern, which makes a pattern applicable for many concrete use cases that may occur. In the course of this, the application of patterns for specific use cases is getting harder because concrete solutions, i.e., implementations of a pattern, are lost during the authoring process. Thus, we showed that connecting concrete solutions to patterns in order to make them reusable when a pattern has to be applied is a valuable concept to save time consuming efforts [11] [12]. This concept is depicted in Fig. 1, where a pattern language is illustrated as a graph of connected patterns at the top. Based on the conceptual solution knowledge, the pattern language opens a solution space, illustrated as an ellipse below the pattern language, which is the space of all possible implementations of the pattern language. Concrete solutions that implement individual patterns of the pattern language are, consequently, located in the solution space and are illustrated as circles. They are related to the pattern they implement, which enables to directly reuse them once a pattern is selected from the pattern language in order to be applied.

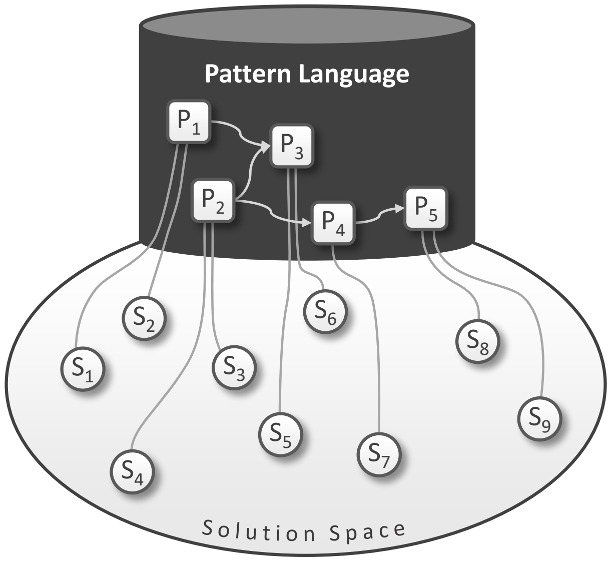


Figure 1. Missing Navigation Support through the Space of Concrete Solutions connected to a Pattern Language

However, while navigation through conceptual solutions is provided by pattern languages in terms of links between patterns, such navigation capabilities are currently not present on the level of concrete solutions, due to the absence of links between the concrete solutions. Thus, if a concrete solution is selected, there is no guidance to navigate through the set of all available and further relevant concrete solutions. Navigation is only possible via the conceptual level of patterns by navigation structures of the pattern language. This is time consuming if experts have their conceptual solution already in mind and want to quickly traverse through available concrete solutions in order to examine if they can reuse some of them for implementing their use case at hand. Further, if a set of concrete solutions is already present that provides implementation building blocks for, e.g., a specific technology, it is often necessary to quickly navigate between them in order to understand their dependencies for reusing them. This is specifically the case, if concrete solutions cannot be reused directly, but need to be adapted to a specific use case. Then, they still can provide a valuable basis for starting adaptions instead of recreating a concrete solution from scratch. Finally, if some concrete solutions have proven to be typically used in combination it is valuable to document this information to ease their reuse. While this could be done on the level of a pattern language, we argue that this is bad practice because implementation details would mix up with the conceptual character of the pattern language. This would require to update a pattern language whenever implementation insights have to be documented. This can get cumbersome, if concrete solutions are collected over a long period of time and technology shifts lead to new implementations and approaches on how to aggregate them, while the more general pattern language stays the same.

Therefore, to summarize the above discussed deficits, there is (i) a lack of organization and structuring at the level of concrete solutions, which (ii) leads to time consuming efforts for traversing concrete solutions, and that (iii) prevents the documentation of proven combinations of concrete solutions.

III. Solution Languages: Means to Structure and Organize Concrete Solutions

To overcome the presented deficits, we introduce the concept of *Solution Languages*. The core idea of Solution Languages is to transfer the capabilities of a pattern language to the level of concrete solutions having the goal of easing and guiding the application of patterns via reusing concrete solutions in mind. Specifically, the following capabilities have to be enabled on the level of concrete solutions: (i) navigation between concrete solutions, (ii) navigation guidance to find relevant further concrete solutions, and (iii) documentation capabilities for managing knowledge about dependencies between concrete solutions, e.g., how to aggregate different concrete solutions to elaborate comprehensive solutions based on multiple patterns.

A. Ease and Guide Traversing of Concrete Solutions

To realize the requirements (i) and (ii), a Solution Language establishes links between concrete solutions, which are annotated by specific semantics that support a user to decide if a further concrete solution is relevant to solve his or her problem at hand. Thereby, the semantics of a link can indicate that concrete solutions connected to different patterns *can be aggregated* with each other, that individual concrete solutions are *variants* that implement the same pattern, or if exactly one of more *alternative* concrete solutions can be used in combination with another one. Depending on the needs of users, also additional link semantics can be added to a Solution Language. To give one example, semantic links can be introduced that specifically indicate that selected concrete solutions *must not be aggregated*. This is useful in cases, when concrete solutions can be technically aggregated on the one hand, but, on the other hand, they implement non-functional attributes that prevent to create a proper aggregated solution. Such situations might occur, e.g., in the field of cloud computing, where applications can be distributed across different cloud providers around the world. Then, this is also implemented by the concrete solutions that are building blocks of such applications. Different concrete solutions can force that individual parts of an application are deployed in different regions of the world. In some cases, law, local regulations, or compliance policies of a company can restrict the distribution of components of an application to specific countries [17]. In such situations, it is very valuable to document these restrictions on the level of concrete solutions via the latter mentioned link type. This can prevent users from unnecessarily navigating to concrete solutions that are irrelevant in such use cases. Nevertheless, the concrete solutions that are not allowed to be used in a specific scenario can be kept in a Solution Language, e.g., for later reuse if preventing factors change or as a basis for adaptions that make them compliant.

While (i) and (ii) can be realized by means of semantically typed links between concrete solutions as introduced above, (iii) requires to introduce the concept of a *Concrete Solution Aggregation Descriptor (CSAD)*. A CSAD allows to annotate a link between concrete solutions by additional documentation that describes the dependency of concrete solutions in a human-readable way. This can, e.g., be a specific description of the working steps required to aggregate the concrete solutions connected by the annotated link. Beyond that, a CSAD can also contain any additionally feasible documentation, such as a sketch of the artifact resulting from the aggregation, which supports a user. The actual content of a CSAD is highly specific for the domain of the concrete solutions. The aggregation of concrete solutions that are programming code can, e.g., often be described by adjustments of configurations, by manual steps to be performed in a specific integrated development environment (IDE), or by means of additional code snippets required for the aggregation. In other domains, such as the non-technical domain of costumes in films, the required documentation to aggregate concrete solutions looks quite different and can, e.g., be a manual about how to combine different pieces of clothing, which in this case are concrete solutions, in order to achieve a desired impression of a character in a movie. So, a CSAD can be leveraged to systematically document concrete implementation knowledge about how to create aggregated overall solutions. Thus, CSADs are the means to add arbitrary documentation about how to aggregate concrete solutions to a Solution Language. Hence, a Solution Language can be iteratively extended over time to preserve the expert knowledge of a domain on the implementation level, the same way as pattern languages do on the conceptual level. Especially in situations, when technologies are getting outdated and experts, which are required to maintain systems implemented in such technologies are getting only scarcely available, Solution Languages can be valuable instruments that preserve technology-specific implementation expertise and documentation. Since concrete solutions are also connected to patterns, which they implement, conceptual, as well as implementation knowledge can be kept easily accessible and inherently connected.

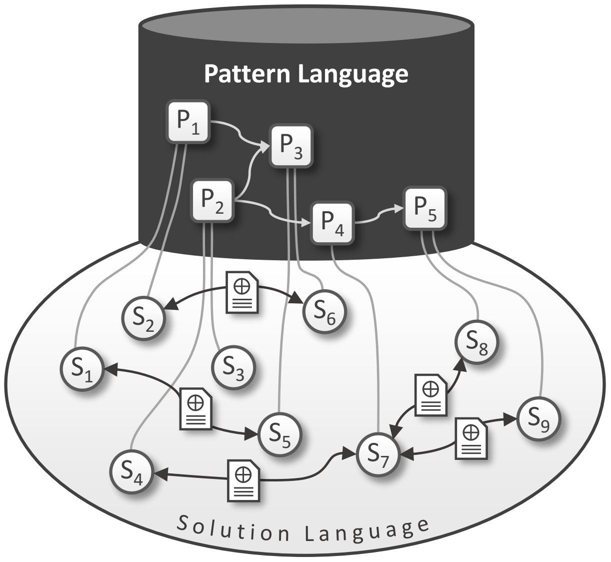


Figure 2. A Solution Language Structures the Solution Space of a Pattern Language and Enables Navigation through Relevant Concrete Solutions

The overall concept of a Solution Language is illustrated in Fig. 2. There, concrete solutions are linked to the patterns they implement. This enables a user to navigate from patterns to concrete implementations that can be reused, as described in our earlier work [11] [12]. Additionally, the concrete solutions are also linked with each other in order to allow navigation on the level of concrete solutions. For the sake of simplicity and clarity, Fig. 2 focusses on links that represent *can be aggregated with* semantics, thus, we omitted other link types. Nevertheless, the relations between the concrete solutions can capture arbitrary semantics, such as those presented above. The semantic links between concrete solutions and the fact that they are also linked to the patterns, which they implement, enables to enter the Solution Language at a certain concrete solution and allows to navigate among only the relevant concrete solutions that are of interest for a concrete use case at hand. For example, if concrete solutions are available that implement patterns in different technologies, then they typically cannot be aggregated. Thus, entering the Solution language at a certain concrete solution and then navigating among only those concrete solutions that are implemented using the same technology, using semantic links indicating this coherence (e.g., *can be aggregated with*), can reduce the effort to elaborate an overall solution significantly. Finally, Fig. 2 depicts CSADs attached to links between concrete solutions in the form of documents. These enrich the semantic links and provide additional arbitrary documentation on how to aggregate the linked concrete solutions.

This way, a Solution Language delegates the principles of pattern languages to the level of concrete solutions, which helps to structure and organize the set of available concrete solutions. While a pattern language guides a user through a set of abstract and conceptual solutions in the form of patterns, a Solution Language provides similar guidance for combining concrete solutions to overall artifacts, all provided by semantic links between concrete solutions and additional documentation about how to aggregate them. Navigation support between concrete implementations of patterns cannot be given by a pattern language itself, because one pattern can be implemented in many different technologies, even in ones that did not exist at the time of authoring the pattern language. Thus, the elucidated guidance is required on the solution level due to the fact that a multitude of different and technology-specific concrete solutions can implement the concepts provided by a pattern language.

B. Mapping Solution Paths from Pattern Languages to Solution Languages

Since pattern languages organize and structure patterns to a navigable network, they can be used to select several patterns to solve a concrete problem at hand by providing conceptual solutions. A user typically tries to find a proper entry point to the pattern language by selecting a pattern that solves his or her problem at least partially. Starting from this pattern, he or she navigates to further patterns in order to select a complete set of patterns that solve the entire problem at hand in combination. This way, several patterns are selected along paths through the pattern language. Thus, the selected patterns are also called a *solution path* through the pattern language [10] [18]. Fig. 3 shows such a solution path by the selected patterns P_2, P_4, and P_5. If several solution paths proof to be successful for recurring use cases, this can be documented into the pattern language to present stories that provide use case-specific entry points to the pattern language [19]. Further, if several concrete solutions are often aggregated by means of the same CSAD, then this can reveal that there might be a candidate of a composite pattern that can be added to the pattern language by abstracting the underlying solution principles, which might be supported and automated by data mining techniques in specific domains [20].

Due to the fact that concrete solutions are linked with the patterns they implement, solution paths through a pattern language can support to find suitable entry points to the corresponding Solution Language. Accordingly, a user can

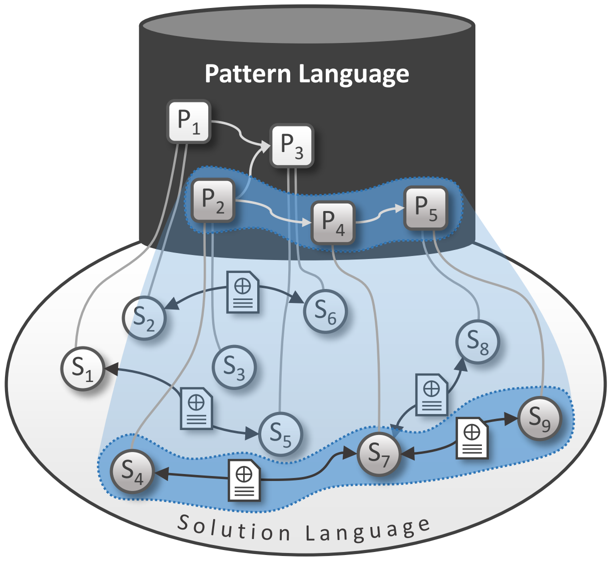


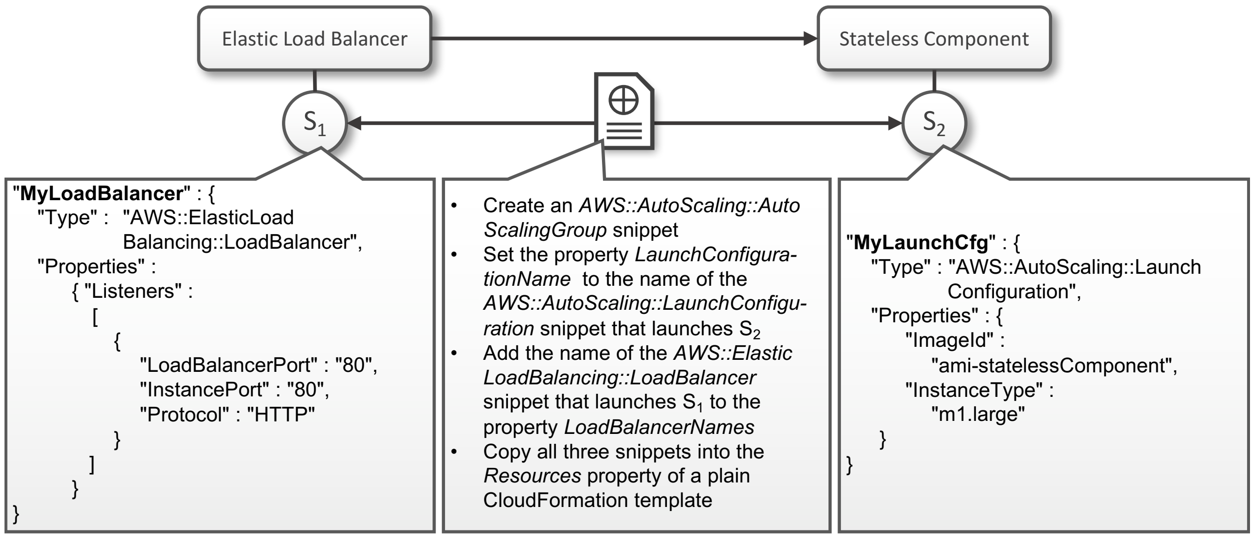
Figure 3. Solution Path from a Pattern Language projected to a Solution Language

navigate from P_2 to the concrete solution S_4. From there, the Solution Language provides navigation support to find further concrete solutions that can be aggregated with S_4. If concrete solutions are available for all patterns contained in the solution path, and if these can be aggregated with each other, then the solution path can be mapped to the Solution Language. This is illustrated in Fig. 3 by the highlighted path from S_4 via S_7 to S_9 through the Solution Language. This allows to translate design decisions that are taken on the conceptual level of the pattern language to reusable concrete solutions that are organized into the Solution Language. The mapping of the solution path to a corresponding set of concrete solutions of the Solution Language can, consequently, provide knowledge about how to elaborate an aggregated solution of the selected patterns by CSADs of the Solution Language, which can significantly speed up the elaboration of an overall concrete solution.

IV. Application in the Domain of Cloud Computing

The pattern language of Fehling et al. [21] provides knowledge about tailoring applications to leverage the capabilities of cloud environments, such as Amazon Web Services (AWS) [22]. One important capability in terms of cloud computing is the automatic and elastic scaling of compute resources. To enable this, the pattern language provides the patterns *Elastic Load Balancer (ELB)* and *Stateless Component (SC)*. ELB describes, how workloads of an application can be distributed among multiple instances of the application. If workload increases, additional instances are added to keep the application responsive. Once, workload decreases, no longer required instances are decommissioned, i.e., to save processing power and expenses. The ELB pattern links to the SC pattern, which describes how components that contain the business logic of an application can manage their state externally, e.g., in an additional database. These patterns are depicted at the top of Fig. 4.

Realizations of these patterns can be connected to them, as depicted in the figure by S_1 and S_2. These concrete solutions implement the patterns by means of AWS CloudFormation [23]



```
"MyLoadBalancer" : {
    "Type" :   "AWS::ElasticLoad
               Balancing::LoadBalancer",
    "Properties" :
        { "Listeners" :
            [
                {
                    "LoadBalancerPort" : "80",
                    "InstancePort" : "80",
                    "Protocol" : "HTTP"
                }
            ]
        }
}
```

- Create an *AWS::AutoScaling::Auto ScalingGroup* snippet
- Set the property *LaunchConfigurationName* to the name of the *AWS::AutoScaling::LaunchConfiguration* snippet that launches S_2
- Add the name of the *AWS::Elastic LoadBalancing::LoadBalancer* snippet that launches S_1 to the property *LoadBalancerNames*
- Copy all three snippets into the *Resources* property of a plain CloudFormation template

```
"MyLaunchCfg" : {
    "Type" : "AWS::AutoScaling::Launch
              Configuration",
    "Properties" : {
        "ImageId" :
            "ami-statelessComponent",
        "InstanceType" :
            "m1.large"
    }
}
```

Figure 4. Concrete Solution Aggregation Descriptor Documenting how to Aggregate Concrete Solutions in the Form of two CloudFormation snippets

snippets, which allows to describe collections of AWS-resources by means of a java script object notation (JSON)-based configuration language. Such configurations can be uploaded to AWS CloudFormation, which then automatically provisions new instances of the described resources. An excerpt of the CloudFormation snippet that describes a load balancer is shown on the left of Fig. 4. The *MyLoadBalancer* configuration defines properties of the load balancer, which are required to receive and forward workload. The corresponding CloudFormation snippet, which implements the concrete solution S_2 is shown on the right. So-called *Amazon Machine Images (AMI)* allow to package all information required to create and start virtual servers in the AWS cloud. Therefore, the *MyLaunchCfg* snippet of S_2 contains a reference to the AMI *ami-statelessComponent*, which is able to create and start a new virtual server that hosts an instance of a stateless component. The link between S_1 and S_2 illustrates that they *can be aggregated* in order to obtain an overall solution, which results in a complete configuration that allows the load balancer instance to distribute workload over instances of virtual servers hosting the stateless component.

If a user wants to aggregate both snippets, he or she can study the CSAD attached to the link between both concrete solutions, which is outlined in the middle of the figure. It provides detailed information about the actual working steps that have to be executed in order to bring both CloudFormation snippets together. Therefore, the CSAD describes that both snippets have to be aggregated via a so-called *AutoScalingGroup*, which is itself also a CloudFormation snippet. The AutoScalingGroup references both, the *MyLoadBalancer* and the *MyLaunchCfg* snippets via the properties *LaunchConfigurationName* and *LoadBalancerNames*. Finally, all three snippets have to be integrated into the property *Resources* of a plain CloudFormation template. By documenting all this information into the Solution Language, (i) the link from concrete solutions in form of CloudFormation snippets to the patterns they implement, (ii) the semantic link between these concrete solutions indicating that they *can be aggregated*, and (iii) the detailed documentation about how to perform the aggregation, can significantly ease the application of the ELB and SC pattern in combination.

Besides the domain of IT, which deals with concrete solutions that are intangible in the sense that they are often programming code or other forms of digital artifacts, there are also domains, exemplarily the domains of building architecture [1] or costumes in films [24], which deal with concrete solutions that are tangible artifacts. While the aggregation of intangible solutions can often be automated [12], e.g., by merging code snippets to an aggregated solution, the aggregation of tangible solutions, such as concrete construction plans of buildings or costumes in a wardrobe, has to be done manually. Especially, in the latter case of tangible solutions, the concept of Solution Languages can be used for documenting knowledge about how to combine concrete solutions. Such knowledge is typically not systematically captured, because of a missing methodical approach. *CSADs* can be used to overcome this problem by documenting procedures and manuals describing the working steps to combine tangible solutions. For the case of costumes in films [15], a Solution Language can be authored that allows to reuse already present costumes for dressing actors. A *CSAD* then can describe, e.g., how roles have to be dressed in a specific scene of a film in order to create the intended expression of how these roles relate to each other to create a perfect immersion.

V. Prototype

To proof the technical feasibility of the presented approach of Solution Languages, we implemented a prototype on the basis of PatternPedia [14]. PatternPedia is a wiki that is built upon the MediaWiki [25] technology and the Semantic MediaWiki extensions [26]. We implemented the case study presented in the previous section. Therefore, we captured the cloud computing patterns in form of wiki pages into PatternPedia and added links between them accordingly to the pattern language of Fehling et al. [21]. We also added the concrete solutions in the form of AWS CloudFormation snippets to PatternPedia so that each AWS CloudFormation snippet is represented by a separate wiki page that references a file containing the corresponding JSON-code. Then, we linked the wiki pages of the concrete solutions with wiki pages representing the patterns they implement to enable the navigation from abstract solution principles captured in patterns to technology-specific implementations in the form of concrete solutions. So, we were able to navigate from patterns to concrete solutions and select them for reuse once a pattern has to be applied. To establish a Solution Language we declared a new property *can be aggregated with* using the Semantic MediaWiki extensions. Properties can be used to define arbitrary semantics, which can be added to wiki pages. The defined property accepts one parameter as a value, which we used to reference wiki pages that represent concrete solutions. This way, concrete solutions can be semantically linked with each other by adding the property into the markdown of their wiki pages and providing the link to the wiki page of the concrete solution, which the *can be aggregated with* semantics holds.

To annotate the link between two specific concrete solutions with information required for their aggregation, we added a CSAD as a separate wiki page containing a detailed description of the working steps required for aggregating them. Finally, we used the query functionality of the Semantic MediaWiki extensions to attach the CSAD to the semantic link between two concrete solutions. We utilized the parser function *#ask* of the Semantic MediaWiki extensions to query the two concrete solutions that are semantically linked with each other via the *can be aggregated with* property. This allowed us to also navigate from one concrete solution to other relevant concrete solutions based on the information of the semantic links, by also providing information about how to aggregate both concrete solutions to an overall one in a human-readable way.

VI. Related Work

The term pattern language was introduced by Alexander et al. [1]. They use this term metaphorically to express that design patterns are typically not just isolated junks of knowledge, but are rather used and valuable in combination. At this, the metaphor implies that patterns are related to each other like words in sentences. While each word does only sparsely provide any information only the combination to whole sentences creates an overall statement. So, also patterns only unfold their generative power once they are applied in combination, while they are structured and organized into pattern languages in order to reveal their combinability to human readers.

Mullet [9] discusses how pattern catalogues in the field of human-computer interaction design can be enhanced to pattern languages to ease the application of patterns in combination. He reveals the qualities of pattern languages by discussing structuring elements in the form of different semantics of pattern relations. Further, the possibility to connect artifacts to patterns, such as detailed implementation documentation or also concrete implementations is identified as future research.

Zdun [18] formalizes pattern language in the form of pattern language grammars. Using this approach, he tackles the problem of selecting patterns from a pattern language. He reflects design decisions by annotating effects on quality attributes to a pattern language grammar. Relationships between patterns express semantics, e.g., that a pattern *requires* another pattern, a pattern is an *alternative* to another one, or that a pattern is a *variant* of another pattern. Thus, he describes concepts of pattern languages, which are transferred in this work to the level of concrete solutions and Solution Languages.

Reiners et al. [27] present a requirements catalogue to support the collaborative formulation of patterns. These requirements can be used as a basis to implement pattern repositories. While the requirements mainly address the authoring and structuring of pattern languages, they can also be used as a basis to detail the discussion about how to design and implement repositories to author Solution Languages. Pattern Repositories [6] [14] [28] have proven to support the authoring of patterns. They enable to navigate through pattern languages by linking patterns with each other. Some (c.f. [6] [14]) also enable to enrich links between patterns by semantics in order to further ease the navigation. Also, conceptual approaches exist that allow to connect a pattern repository with a solution repository, which can be the foundation to implement the concepts introduces in this work. These concepts and repository prototypes can be combined with our approach to develop sophisticated solution repositories.

Barzen and Leymann [15] present a general approach to support the identification and authoring of patterns based on concrete solutions. Their approach is based on research in the domain of costumes in films, where they formalize costume languages as pattern languages. Costumes are concrete solutions that solve specific design problems of costume designers. They enable to hark back to concrete solutions a pattern is evolved from by keeping them connected. They also introduce the terminus Solution Language as an ontology that describes types of clothes and their relations in the form of metadata, as well as instances of these types. This completely differs from the concept of a Solution Language as described in this work.

Fehling et al. [16] present a method for identifying, authoring and applying patterns. The method is decomposed into three phases, whereby, in the pattern application phase, they describe how abstract solutions of patterns can be refined towards concrete implementations. To reduce the efforts to spend for implementing patterns, they apply the concept of concrete solutions by means of code repositories that contain reference implementations of patterns. While our approach is designed and detailed for organizing concrete solutions the argumentation in their work is mainly driven by considerations about patterns and pattern languages. Thus, the method does not introduce how to systematically combine semantics and documentation in order to organize concrete solutions for reuse, which is the principal contribution of our work.

VII. Conclusion and Future Work

In this work, we presented the concept of Solution Languages that allows to structure and organize concrete solutions, which are implementations of patterns. We showed how Solution Languages can be created and how they can support the navigation through the solution space of pattern languages based on semantic links, all targeting to ease and guide pattern application. We further presented the concept of Concrete Solution Aggregation Descriptors, which allows to add arbitrary human-readable documentation to links between concrete solutions.

In future work, we are going to conduct research on how to analyze Solution Languages in order to derive new pattern candidates based on Concrete Solution Aggregation Descriptors annotated to links between concrete solutions, but also on the question if a Solution Language can indicate new patterns in a pattern language, for instance, in the case if links between concrete solutions are missing or if aggregation documentation cannot be clearly authored. We are also going to apply the concept of Solution Languages to domains besides cloud computing, e.g., to the emerging field of the Internet of Things.

Acknowledgment

This work is partially funded by the BMWi project SePiA.Pro (01MD16013F) as part of the Smart Service World.

References

[1] C. Alexander, S. Ishikawa, and M. Silverstein, A pattern language: towns, buildings, construction. New York: Oxford University Press, 1977.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Abstraction and reuse of objectoriented design," in European Conference on Object-Oriented Programming, 1993, pp. 406–431.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and P. Stal, Pattern-oriented software architecture: A system of patterns, 1996, vol. 1.

[4] M. van Welie and G. C. van der Veer, "Pattern Languages in Interaction Design : Structure and Organization," in Human-Computer Interaction '03: IFIP TC13 International Conference on Human-Computer Interaction. IOS Press, 2003, pp. 527–534.

[5] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, And Deploying Messaging Systems. Addison-Wesley, 2004.

[6] R. Reiners, "An Evolving Pattern Library for Collaborative Project Documentation," PhD Thesis, RWTH Aachen University, 2013.

[7] L. Reinurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of things patterns," in Proceedings of the 21th European Conference on Pattern Languages of Programs, 2016.

[8] J. Barzen et al., "The vision for MUSE4Music," Computer Science - Research and Development, vol. 22, no. 74, 2016, pp. 1–6.

[9] K. Mullet, "Structuring pattern languages to facilitate design. chi2002 patterns in practice: A workshop for ui designers," 2002. [Online]. Available: https://www.semanticscholar.org/paper/Structuring-Pattern-Languages-to-Facilitate-Design-Mullet/2fa5e4c25eea30687605115649191cd009a8f33c

[10] M. Falkenthal et al., "Leveraging pattern application via pattern refinement," in Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change, in press.

[11] M. Falkenthal, J. Barzen, U. Breitenbuecher, C. Fehling, and F. Leymann, "From Pattern Languages to Solution Implementations," in Proceedings of the 6th International Conferences on Pervasive Patterns and Applications, 2014, pp. 12–21.

[12] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, and F. Leymann, "Efficient Pattern Application : Validating the Concept of Solution Implementations in Different Domains," International Journal On Advances in Software, vol. 7, no. 3&4, 2014, pp. 710–726.

[13] G. Meszaros and J. Doble, "A Pattern Language for Pattern Writing," in Pattern Languages of Program Design 3. Addison-Wesley, 1997, ch. A Pattern Language for Pattern Writing, pp. 529–574.

[14] C. Fehling, J. Barzen, M. Falkenthal, and F. Leymann, "PatternPedia Collaborative Pattern Identification and Authoring," in Pursuit of Pattern Languages for Societal Change - The Workshop 2014: Designing Lively Scenarios With the Pattern Approach of Christopher Alexander. epubli GmbH, 2015, pp. 252–284.

[15] J. Barzen and F. Leymann, "Costume Languages as Pattern Languages," in Pursuit of Pattern Languages for Societal Change - The Workshop 2014: Designing Lively Scenarios With the Pattern Approach of Christopher Alexander, 2015, pp. 88–117.

[16] C. Fehling, J. Barzen, U. Breitenbücher, and F. Leymann, "A Process for Pattern Identification, Authoring, and Application," in Proceedings of the 19th European Conference on Pattern Languages of Programs, 2015, article no. 4.

[17] U. Breitenbücher et al., "Policy-Aware Provisioning and Management of Cloud Applications," International Journal On Advances in Security, vol. 7, no. 1 & 2, 2014, pp. 15–36.

[18] U. Zdun, "Systematic pattern selection using pattern language grammars and design space analysis," Software: Practice and Experience, vol. 37, no. 9, jul 2007, pp. 983–1016.

[19] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley & Sons, 2007, vol. 5.

[20] M. Falkenthal et al., "Pattern research in the digital humanities: how data mining techniques support the identification of costume patterns," Computer Science - Research and Development, vol. 22, no. 74, 2016.

[21] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer, 2014.

[22] Amazon, "Amazon Web Services," 2017. [Online]. Available: http://aws.amazon.com/

[23] ——, "Amazon Cloud Formation," 2017. [Online]. Available: https://aws.amazon.com/cloudformation/

[24] D. Schumm, J. Barzen, F. Leymann, and L. Ellrich, "A Pattern Language for Costumes in Films," in Proceedings of the 17th European Conference on Pattern Languages of Programs, 2012, article no. 7.

[25] Wikimedia Foundation, "MediaWiki," 2017. [Online]. Available: https://www.mediawiki.org/

[26] M. Krötzsch, "Semantic MediaWiki," 2017. [Online]. Available: https://www.semantic-mediawiki.org/

[27] R. Reiners, M. Falkenthal, D. Jugel, and A. Zimmermann, "Requirements for a Collaborative Formulation Process of Evolutionary Patterns," in Proceedings of the 18th European Conference on Pattern Languages of Programs, Irsee, 2013, article no. 16.

[28] U. van Heesch, "Open Pattern Repository," 2009. [Online]. Available: http://www.cs.rug.nl/search/ArchPatn/OpenPatternRepository

All links were last accessed on 14.01.2017