PURPLSOC

# PatternPedia – Collaborative Pattern Identification and Authoring

Fehling, Christoph, Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany, Fehling@iaas.uni-stuttgart.de

Barzen, Johanna, Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany, Barzen@iaas.uni-stuttgart.de

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

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

**Abstract:**

*The process to identify and author patterns often involves multiple domain experts. This paper introduces PatternPedia – a collaborative tool chain to document existing solutions and manage patterns abstracted from them. We present an extensible pattern metamodel specified in UML to enable this tool support. Sample metamodel extensions are covered for the domain of cloud computing and costumes in films to capture concrete existing solutions and patterns in these domains. Respective solution repositories and pattern repositories have been implemented based on these metamodel extensions. Support for pattern document display, pattern reference visualization, as well as queries on the costume solution repository are presented.*

**Keywords:** *Pattern Repository; Solution Repository; Pattern Authoring; Wiki; Collaboration*

ISSN (tba)
www.purplsoc.org

# 1. Introduction

Patterns are human-readable documents that describe proven solutions in a certain domain, such as building architecture, IT applications, or education. Patterns reference each other, to express common combinations, alternatives etc. The set of patterns and their interrelations are often referred to as a pattern language for the considered domain. In Barzen & Leymann (2014), we covered the identification of patterns in the domain of costumes in films: by documenting and analyzing costumes in different movies, the identification of patterns is systemized. The description format of existing solutions (costumes) is formalized, existing solutions are captured, and patterns are identified in the set of solutions. Following such an approach, the origin of a pattern becomes clearly visible and traceable to ensure pattern provenance. Such a systematic approach to pattern discovery often involves many participants, for example, to review films and capture costumes scene-by-scene. In other domains, where we conducted pattern research, such as cloud computing, collection and classification of information sources was also handled by many parties (Fehling, Ewald, Leymann, Pauly, Rütschlin & Schumm, 2012). These collaborative tasks for information collection, pattern identification, and pattern authoring should, therefore, be supported by a collaborative tool chain. While Barzen & Leymann (2014) introduced a formal model for costumes and costume patterns, this paper provides (i) an extendable *pattern metamodel* for the formal model and (ii) a tool chain (*PatternPedia* - http://www.patternpedia.net) for the collaborative documentation of solutions and identification of patterns. Both, the extendable pattern metamodel and tool chain are used in the domain of cloud computing (IT applications) and costumes in films.

We first cover the relevant related work on wiki-based pattern management and pattern authoring in Section 2. We introduce the pattern research methodology supported by PatternPedia, cover its abstract system architecture, and its usage by pattern researches (Section 3). The remaining sections describe the formal basis of the PatternPedia data structure and the technical implementation of PatternPedia:

The pattern metamodel and its extensions for the domains of cloud computing and costumes is expressed by the Unified Modeling Languge (UML) (Object Management Group, 2011) (Section 4). This modeling language is commonly used for IT applications to describe the abstract structure of data elements and their instances. The metamodel, therefore, specifies a data structure that can be instantiated multiple times to represent multiple instances of the described entity. The description of IT systems using such modeling activities is described as Model Driven Design (Raistrick, Francis, Wright, Carter & Wilkie, 2004). Therefore, the pattern metamodel describes the reoccurring data structure of patterns and solutions as they become manifested in the supporting tool chain. It coordinates the collaborative word of participating researchers. The concrete implementation of PatternPedia is governed models (Section 5) and based on MediaWiki[1], the wiki software that supports WikiPedia[2], and semantic extensions provided by SemanticMediawiki[3]. We chose to use and extend this well-established wiki software to benefit from existing collaboration functionality that can support the pattern research domain without adaptation. The paper is concluded by a summary and outlook in Section 6.

[1] http://www.mediawiki.org
[2] http://www.wikiepedia.org
[3] http://semantic-mediawiki.org/

# 2. Related Work

Existing pattern language formalizations and pattern repositories have been considered as the foundational work for PatternPedia. Wikis have been used early on in the pattern research domain to coordinate the work of multiple pattern authors. Existing pattern repositories, however, are often not been available as open source and often did not support types for the references among patterns. Through reference typing, the tool chain explicitly supports different links between patterns, to express that two patterns are related, are often combined with each other etc. While reference typing seems to be a small detail, the magnitude of references within a pattern catalog can quickly become unmanageable if reference types are omitted. Some existing pattern repositories support this typing and their functionality has been investigated especially to deduct requirements on the PatternPedia tool chain. Existing methodologies to author and improve patterns without any tool support have also been considered, especially, how they can be integrated with the PatternPedia tool chain.

## 2.1. Existing Pattern Languages and Pattern Language Formalizations

The pattern metamodel describes the structure of the pattern documents and references among them. To find an adequate extendable metamodel, the format of existing patterns and pattern languages has been investigated especially how they reference other patterns.

Zdun (2007) formalized pattern descriptions and the use of formal grammars describing the selection of patterns. Pattern primitives are covered in Zdun & Avgeriou (2005) as a means to capture domain-specific concepts that are not patterns themselves, but smaller artifacts forming larger patterns. In the domain of costumes, such primitives are the pieces of clothing comprising a costume (Schumm, Barzen, Leymann & Ellrich, 2012). In general, when authoring patterns domain-specific primitives should be defined to be used in solution descriptions and pattern descriptions. Hanmer (2012) covers the structure of a pattern catalog in the domain of IT applications. He describes a means for pattern users to collect their individual set of patterns and interconnect these patterns based on their own experience and development processes. The pattern language of Alexander, Ishikawa & Silverstein (1977) was considered as the most influential source of patterns for the domain of building architecture. These patterns describe the design of cities, neighborhoods, houses, individual rooms etc. A clear order of consideration and refinement of details is visible in this pattern language. Even though the pattern format used by Alexander et al. (1977) does not use section headings and does not label references, a clear semantic is visible in the separate sections of each pattern describing the problem, context, solution etc. Regarding reference types the order of consideration and the use of alternatives are visible.

Regarding the domain of IT applications, the patterns of Hohpe & Woolf (2004), Gamma, Helm, Johnson & Vlissides (1994), Buschmann, Meunier, Rohnert, & Stal (1996), Fowler (2002), and Hanmer (2013) have been reviewed. Hohpe and Woolf describe patterns for enterprise application integration. Their format highly influenced the cloud computing patterns developed by us (Fehling, Leymann, Retter, Schupeck & Arbitter, 2014). Gamma et al. and Buschman et al. describe patterns in the domain of object-oriented application design. Fowler covers patterns for enterprise applications that incorporate the business case, its domain model, and how it is refined to application architectures. Hanmer covers patterns in the domain of fault-tolerant distributed applications. All of these pattern catalogs have

been analyzed with respect to their individual pattern formats and references among patterns. The pattern metamodel introduced in this work aims to support all of these formats through extension.

## 2.2. Existing Pattern Repositories

The need for a central pattern repository and, especially, the use of wiki software for pattern management has long been identified as promising (Leuf & Cunningham, 2001). Pattern repositories have since been established for many different topics. In many cases, the motivation for these repositories is publishing the contained patterns and not the reuse of the software for other patterns, which may have different formats. Many consider collaboration aspects, such as rating or recommending patterns. Table 1 summarizes the aspects analyzed for some of these pattern repositories. It shows the name and internet address of the repository and gives the main topic considered by contained patterns if applicable. While this list may not be exhaustive, we argue that it represents a significant portion of the pattern repositories available today. A more detailed evaluation of existing pattern repositories can be found in Fürst (2013) and Willig (2014).

| Name | Address | Topic | Software Available | License | Adjustable | Comments | Rating | Authoring |
|---|---|---|---|---|---|---|---|---|
| A Pattern Library for Interaction Design | http://www.welie.com/ | User Interface Design | No | N/A | N/A | Yes | No | No |
| BRIDGE Pattern Library | http://pattern-library.sec-bridge.eu/pattern-library/ | Various | No | N/A | N/A | Yes | Yes | Yes |
| Open Pattern Repository | https://code.google.com/p/openpatternrepository/ | Various | Yes | GPL | Yes | Yes | Yes | Yes |
| Portland Pattern Repository | http://c2.com/ppr/ | Various | Yes | Various | Yes | Yes | No | Yes |
| Yahoo Design Pattern Library | https://developer.yahoo.com/ypatterns/ | User Interface Design | No | N/A | N/A | No | No | No |

**Table 1: Overview of considered Pattern Repositories**

The pattern library for interaction design covers user interaction patterns and mainly focuses on the presentation of these patterns. The BRIDGE pattern library was developed in a European research project. It mainly focuses on the collaborative authoring of patterns. Requirements for a collaborative pattern repository that are implemented in the BRIDGE pattern library are published in Reiners, Falkenthal, Jugel & Zimmermann (2013). The open pattern repository is a customized wiki implementation for pattern management. Especially, the addition of existing pattern documents is well supported by the user interface. Links in this pattern repository are similarly typed as those of PatternPedia. The Portland pattern repository is possibly the oldest pattern repository and also based on wiki software: WikiWiki[4]. The Yahoo design pattern library again focuses on user interface design and its main purpose is providing these patterns to the public. Conceptually, the pattern format and reference types of these existing pattern repositories have been considered during the design of the extensible pattern metamodel.

The BRIDGE pattern library and open pattern repository seemed most accessible regarding their existing source code implementations. The implementation of PatternPedia described in Section 5, nevertheless, was realized using MediaWiki as a basis for two main reasons. First, this open source wiki software addresses most of the desired collaborating features directly. Second, the semantic extensions enable the desired reference typing and queries based on

[4] http://c2.com/cgi/wiki?WikiWikiClones

these reference types. By relying on such an existing code base combined with some extensions, the need for custom implementations and future maintenance effort could be reduced. Again, refer to Fürst (2013) and Willig (2014) for a detailed evaluation of wiki software and available extensions relevant to pattern research.

### 2.3. Existing Best Practices for Pattern Authoring

Existing methodologies to identify, author, and improve patterns still have to be considered regardless whether or not a tool chain is used to support pattern researchers. Wellhauser & Fießler (2011) cover how pattern documents should be designed to be accessible to readers. Meszaros (1997) covers patterns to be considered during pattern writing. These best practices again describe the structure of pattern documents and good writing practices. Harrison (1999) provides patterns for the following review cycle of initially written pattern documents. Lucrédio, de Almeida, Alvaro, Garcia & Piveta (2004) describe how to review and improve patterns during research conferences. Iba & Isaku (2012) present patterns on how to conduct interactive workshops in order to find and author patterns collaboratively. PatternPedia does not aim to replace these methods and techniques. Collaboration during pattern identification shall be supported during the times when face-to-face interaction is impossible. Also, the traceability of patterns shall be increased by documenting existing solutions in the same tooling environment. This also enables pattern users to learn from existing solutions to a higher degree. However, using PatternPedia will not alleviate the need to conduct face-to-face discussions and interactive workshops to identify and author patterns.

## 3. Pattern Research Methodology and Architecture of PatternPedia

To describe how PatternPedia is used and which functional components it is comprised of, we now cover the research methodology that is supported by the PatternPedia tool chain. Then the abstract functional components are described to give an overview of the architecture of PatternPedia. A central artifact governing the use of this tool chain is the pattern metamodel, which homogenizes the data created by different users. The structure and adaptation of this metamodel to the domains of cloud computing and costumes in films is covered separately in Section 4.

### 3.1. Pattern Research Methodology supported by PatternPedia

The tasks to identify, abstract, and apply patterns depicted in Figure 2 are supported by the PatternPedia tool chain. The figure furthermore shows the data generated by these tasks as well as the involved user roles. Each role can be fulfilled by one or more users of PatternPedia. One user or user group can also fulfill multiple roles.

The initial activity is to *document solution*s in the domain considered for pattern research, i.e., to capture how existing IT applications work, how costumes look like etc. This activity collects such information from various sources, such as written documents, mindmaps, spreadsheets etc. and homogenizes this format as well as abstracts the content to describe *concrete solutions*. A *domain expert*, who has created the existing solutions and a *solution collector*, who is familiar with the abstraction of required information, collaborate on this task.

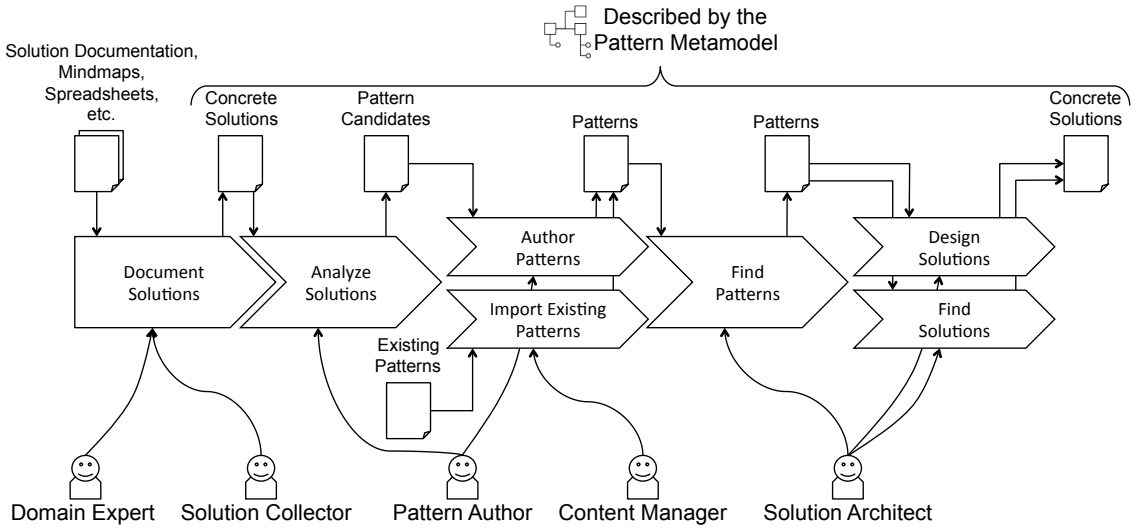


**Figure 1 – Pattern Research Methodology**

As a second step, solutions are analyzed for similarities by the pattern author to identify pattern candidates. A *pattern candidate* proposes a pattern, which is then reviewed by a larger group of pattern authors as part of the author patterns task. Especially, the best practices for pattern writing and improvement (see Section 2.3) shall be used here. Of course, there may be existing patterns in the considered domain. In parallel to the author patterns task, such patterns are imported to PatternPedia in order to be interconnected with newly found patterns. A *content manager* handles this task of transforming exiting patterns from books, conference proceedings etc. into the same format as the newly created patterns.

After patterns have been found in the researched domain, a *solution architect* can use this set of patterns to create new solutions. This could be a new IT application, a costume etc. In scope of PatternPedia, such a concrete solution is considered the description of the solution that shall be managed by PatternPedia. Thus, this is a written document following the format specified by the pattern metamodel that describes the solution. The solution itself, i.e., the code of an application or the tangible costume is not managed by PatternPedia. A solution architect starts to create such new solutions by identifying patterns that are applicable to his or her problem. This identification is part of the find patterns tasks.

Applicable patterns are then refined by the solution architect as part of two parallel activities. The solution design is created, thus, details are added with respect to the abstract pattern description and multiple patterns are combined into a new solution design. During this task, the solution architect may also analyze existing solutions to not only learn from the pattern itself but also from all of its previous applications. As a result, new concrete solutions are created that may again be analyzed to identify new patterns. This research methodology, therefore, describes a continuous process that is followed iteratively.

## 3.2. Functional Architecture and Use of PatternPedia

In every domain where patterns shall be researched, the information is commonly available in a much unstructured format. As shown on the left of Figure 2, such information sources may subsume movies, documents, mindmaps etc. These information sources are provided by the role of a *domain expert*, who is familiar with the currently existing solutions and best-practices. These information sources are not managed by the PatternPedia tool chain whose components are depicted in the center of Figure 2.

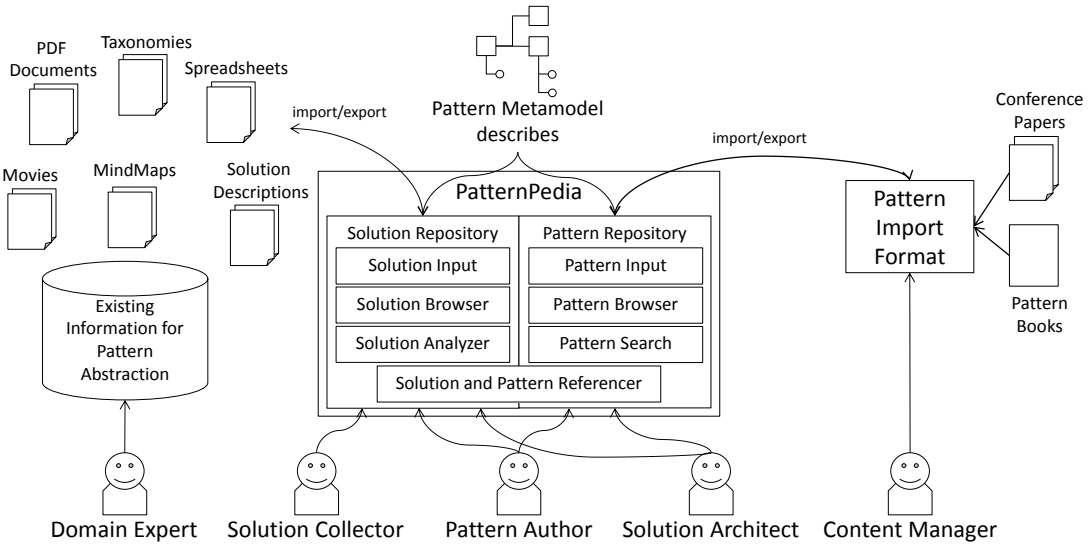


**Figure 2 - Abstract Architecture of PatternPedia**

The *solution input* component is used by the solution collector to enter information about existing solutions obtained from the domain expert and information sources into the solution repository. The pattern metamodel ensures that such information contained in the solution repository follows a homogenous data format. Solutions may then be browsed freely, but more analysis functionality may also be offered to query solutions. While the solution repository can be supported by the wiki-based implementation covered in Section 5, it is often realized as a separate custom tool. This is due to the fact that domains differ greatly regarding the aspects of solutions that shall be captured to support pattern identification. For example, in the domain of cloud computing, architectural goals, such as availability or performance have been used to categorize the documented solutions. In the domain of costumes, detailed ontologies and taxonomies have been used to describe garments of clothes, their colors, structure etc. in order to homogenize the solution format (Barzen & Leymann, 2014). Section 4 covers the concrete metamodel extensions used in these domains. A custom tool can better support such extensions, especially, regarding the enforcement of the desired solution format and queries to the solutions in order to find patterns. Such queries are especially helpful to the pattern author who accesses the *solution repository* to query and analyze solutions in order to identify new patterns.

Patterns are documented in the *pattern repository*. Multiple pattern authors may work on a pattern simultaneously during its authoring. Again, functionality is provided to input and browse these patterns. The pattern metamodel describes the format of the pattern document, i.e., its sections and their semantic, as well as the types of references between multiple patterns. Such pattern references are used by pattern authors to express that patterns form alternatives, are often used together, should be considered in a certain order etc. The component for *solution and pattern referencing* shown in Figure 2 supports the creation of such pattern references: as the number of patterns increases, those patterns written later tend to have fewer incoming references. By the time other patterns have been written, newer patterns did not exist for referencing. Such conditions may be identified automatically by the tool to suggest references that should be added.

The pattern references are fundamental to browse and find applicable patterns. The *solution architect* uses the references among patterns to navigate the catalog of patterns as part of the functionality provided by the pattern search component. The references provide the order

of pattern consideration, related patterns, alternatives etc. as the solution architect traverses the catalog. Furthermore, the references between solutions and the patterns identified from them are also maintained. Such references between patterns and solutions enable the solution architect to access information about existing solutions to refine his or her design. He or she may learn from the existing solutions while the pattern provides the abstract best practices. After design and implementation of a new solution, the solution architect documents his or her design in the solution repository. A reference to the implemented pattern ensures that this new solution may also be accessed in all future applications of the pattern. In order to integrate existing patterns published in books, conference proceedings etc. into the pattern repository, the *content manager* converts existing pattern documents into an import format that can be understood by the pattern repository. We use the Extensible Markup Language (XML) (World Wide Web Consortium, 2008) for this purpose as described in Fehling & Leymann (2014).

## 4. Formalizing the Pattern Metamodel

Patterns and solutions are documented in a well-defined data model in PatternPedia. This ensures format homogenization if multiple authors participate in the documentation process of identifying and authoring patterns (Section 3). Also, it enables a later querying of solutions and pattern documents. The metamodel seen in Figure 3 is described in the Unified Modeling Language (UML) (Object Management Group, 2011). The central entity in this model is an abstract *Referenceable Document*, which is associated with a *Category*. Such Categories can be used to organize the referenceable documents managed in PatternPedia. A referenceable document manifests either in form of a *Solution* or a *Pattern*. Solutions are the documents from which patterns are abstracted. The pattern language metamodel does not enforce a solution to be structured, thus, it is constituted by arbitrary *ImageElements* and *TextualElements*. These two entities are also used in patterns, but patterns are structured additionally. In the pattern metamodel, patterns are constituted by *Intro* entities and *Section* entities. Sections describe the patterns format and, thus, have a distinct name – their heading. Intro entities are used for textual or graphical elements that are included in a more loosely fashion, often in the beginning of pattern documents. For example, a picture or icon that readers shall associate with the pattern. Two different reference entities can be used between referenceable documents. *Textual References* are used in pattern sections or intros. The reference itself is then displayed as the textual element in the pattern documents. *Global References* are not displayed in pattern documents as discrete textual elements. Instead, they are often used for visualization purposes outside of pattern documents, for example, to render a graph indicating the order in which patterns should be considered.

In the generic pattern language metamodel two *Reference Types* are included. The *RelatedTo* reference type is used to point to patterns that are relevant in context of the described pattern, that are often used together with the described pattern etc. In itself, the RelatedTo reference type is, thus, very generic. The *InContextOf* reference type is used to point to patterns forming the setting in which another pattern can be applied. These two reference types are the intersection of the sets of reference types used in pattern languages considered as related work in Section 2. Commonly, this set is extended for a pattern language, for example, to denote alternatives and compositions between patterns or to point to solutions via a known use reference. The concrete extensions for the cloud computing patterns and costume patterns are described in the following sections.

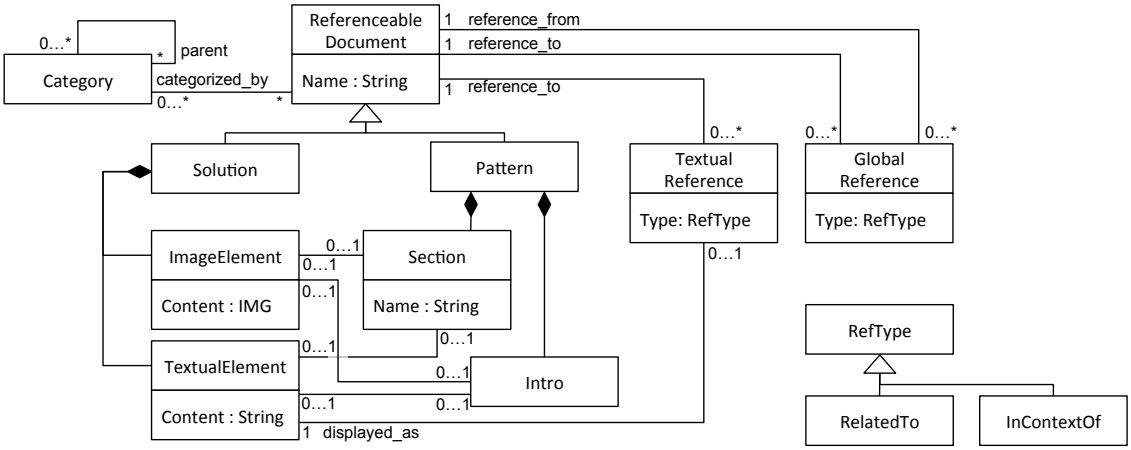


**Figure 3 - Pattern Metamodel in UML**

## 4.1. Extending the Pattern Metamodel for Cloud Computing Patterns

Given the architectural goals of cloud computing applications, information sources have been analyzed to identify and extract cloud computing patterns (Fehling et al., 2014). Therefore, it was documented how existing applications achieve the architectural goals. For provider documentations, guidelines to achieve the goals were documented. The extracted pattern documents use the pattern metamodel with the following extensions. It has been extended at four locations as shown in Figure 4. Extension points one and two describe a more specific pattern format. Extension point three introduces additional reference types that are used to interconnect the patterns of the cloud computing pattern language. Extension point four refines the information captured about solutions in order to identify patterns. The specific model extensions are detailed in the following.

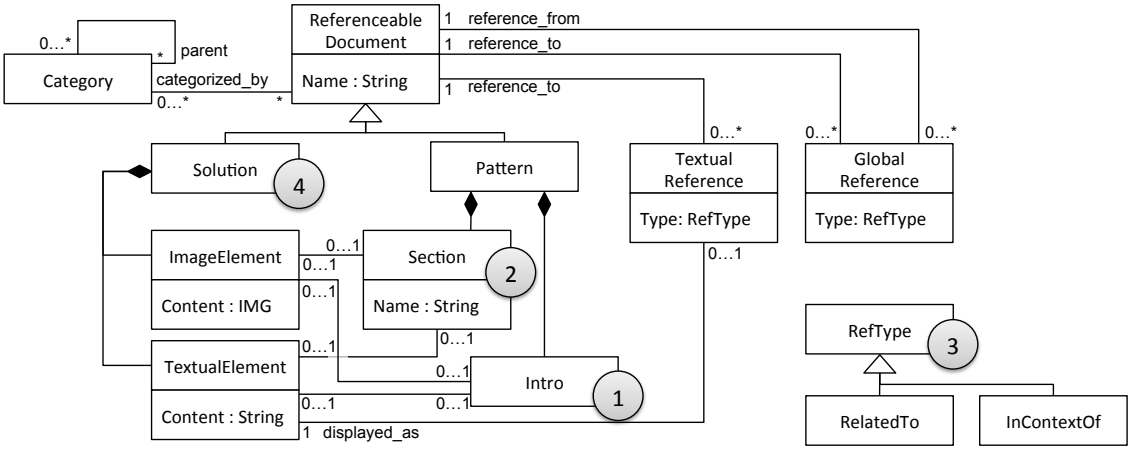


**Figure 4 - Overview of Extension Points for Cloud Computing Patterns**

### 4.1.1. Extension of the Pattern Format

The pattern format extension describes (i) the used pattern introduction at the beginning of each pattern document and (ii) the specific names of pattern sections to be used. Figure 5 displays the extension of the pattern metamodel with specific intro elements. These are used in the beginning of each document without individual section headings.

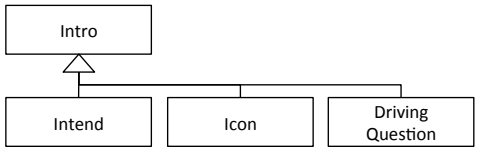


**Figure 5 - Extension for specific Introduction Content of Cloud Computing Patterns**

The *Intend* of each cloud computing pattern summarizes the complete pattern very briefly. It, therefore, gives readers a very quick overview. The *Icon* of each cloud computing pattern is its graphical representation. It, especially, is intended to be used in architectural diagrams of (i) other architectural patterns composing the pattern and of (ii) architectural diagrams using the cloud computing patterns. The *Driving Question* finally gives the questions to be answered by the pattern. Thus, it allows the reader to identify the applicability of the pattern to his or her own architectural questions in a given use case. Figure 6 shows an intro format refinement of the public cloud pattern from the website http://cloudcomputingpatterns.org.

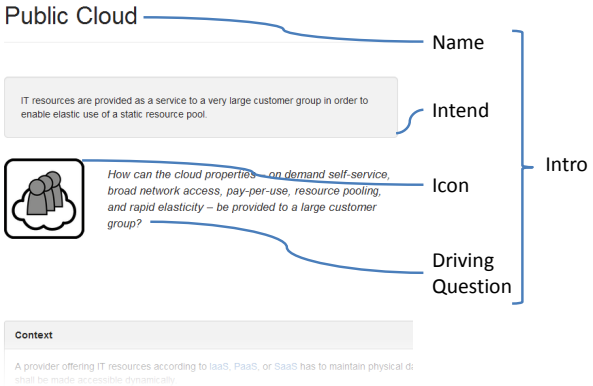


**Figure 6 - Refinement of the Intro in the Pattern Format for Cloud Computing Patterns**

After the introduction elements, the cloud computing patterns use a specific set of sections. The corresponding metamodel extension is shown in Figure 7.

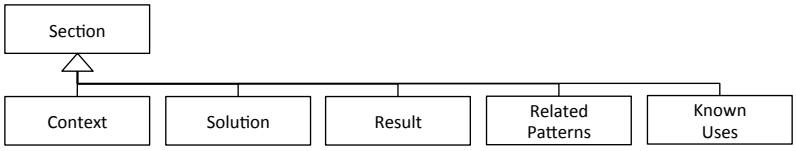


**Figure 7 - Extension for specific Sections of Cloud Computing Patterns**

The *Context* section describes the conditions under which the problem solved by a pattern arises. Especially, other patterns may be referenced here that form the environment in which the pattern is applied. Other influencing factors that lead to the problem - so called *forces* - are also listed here, so that the reader may compare them to the real life problem he or she is trying to solve. The *Solution* section then briefly states how the pattern solves the problem. This statement is given in an imperative form to quickly tell the reader what has to be done when applying the pattern. Each solution section contains one or more images – the *Sketch* of the pattern. This sketch is a graphical representation of the solution. It shows the reader abstract components to be implemented, other patterns to be composed, an abstract process that the implementation has to realize etc. Following the solution section, the *Result* section describes the outcome of the pattern application in greater detail. Especially, it may describe additional problems that may arise after the application of a pattern. The *Variations* section describes alternative applications of the pattern. If these variations are not significant enough

to justify their description as separate patterns, they are listed in this section. The *Related Patterns* section lists other patterns that may be relevant for consideration if the pattern is applied. These may be alternative patterns following a different approach or patterns that are often used together with the pattern etc. Finally, the *Known Uses* section closes the pattern. In this section, all information sources are summarized from which the pattern has been abstracted, especially, if they were existing applications. Remember that the pattern metamodel allows the capturing of known uses as separate documents – the solutions. Thus, they may be referenced by patterns just like other patterns, but they do not follow the same document structure.

### 4.1.2. Extension of the Pattern References

In scope of the cloud computing patterns, both reference types in the pattern metamodel may be used. Furthermore, additional types have been defined as shown in Figure 8.


**Figure 8 - Extension for specific Reference Types of Cloud Computing Patterns**

The *ConsiderAfter* reference type is used to describe an order of pattern consideration for the cloud computing patterns language. For example, these references are used to suggest that readers shall first consider cloud offerings, then application component patterns building applications on top of such offerings. More specifically, if a reader considers to use a blob storage offering[5] to store data, the ConsiderAfter reference may point to strict consistency[6] and eventual consistency[7] as patterns that describe the offering behavior in greater detail. Also, the data access component[8] pattern may be referenced, because after the reader has chosen a storage offering, he or she should consider how to build an application component that interacts with the storage offerings and provides data accessibility to other application components. The *Alternative* reference type connects patterns that cannot be combined but form different approaches to solve a similar problem. This way, if a reader finds a pattern relevant to his or her problem, but the solution is unsuitable, the alternative references may point to similar patterns that may be applicable instead. The *Composition* reference is used to point to other patterns that are used in the solution of the described pattern. More complex patterns may, thus, combine other patterns to form more complex solutions. The *KnownUse* reference type is the only one that does not connect two patterns. Instead, it connects patterns with solutions documented during the information collection phase. Readers may use these references to learn how the described pattern may be applied from existing solutions. While most of these reference types are special forms of the generic *RelatedTo* relationship type, modeling them explicitly enables a better interpretation by supporting tool chains to ensure a higher level of usability to users.

[5] http://www.cloudcomputingpatterns.org/Blob_Storage
[6] http://www.cloudcomputingpatterns.org/Strict_Consistency
[7] http://www.cloudcomputingpatterns.org/Eventual_Consistency
[8] http://www.cloudcomputingpatterns.org/Data_Access_Component

### 4.1.3. Extension of the Solution Format

The cloud computing patterns (Fehling et al., 2014) have been extracted from information sources, such as cloud provider documentation or documentation about existing applications. See Fehling et al. (2012) for a detailed list. These information sources are captured as solutions in scope of the pattern metamodel. For every information source, the following aspects have been collected. The metamodel has been extended respectively as shown in Figure 9.

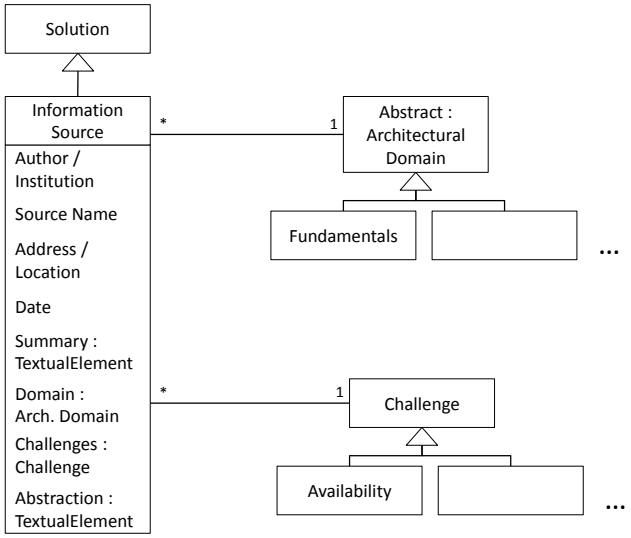


**Figure 9 - Extension for specific Solution Format of Cloud Computing Patterns**

Author, Source Name, Address, and Data capture general information about the information source. Summary is a Textual Element that briefly captures the essence of the solution. The domain is used to classify information sources. A large set of Architectural Domains has been identified that is relevant for cloud applications (Fehling et al., 2012). Similarly, challenges, such as scalability, performance etc. are used to group information sources. Finally, the summary is abstracted to provider-independent statements. These abstractions were identified in multiple information sources to find patterns. Exemplary information source for cloud computing patterns:

**Author:** Jinesh Varia
**Source Name:** Cloud Architectures
**Address:** http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf
**Date:** 01. Jun 08
**Summary:** Cloud resources should be started and stopped automatically.
**Domain:** Cloud Application Management
**Challenge:** Scalability
**Abstraction:** Cloud resource management has to be automated.

Please download the pattern authoring toolkit[9] (Fehling et al., 2012) for a complete list of information sources from which a pattern has been abstracted.

## 4.2. Extending the Pattern Metamodel for Costume Patterns

Patterns in the domain of costumes in films require analogous extensions of the pattern metamodel as explained for cloud computing patterns above. Although, the explained

[9] http://cloudcomputingpatterns.org/authoringtoolkit.zip

extensions are not considered to be final at the moment due to ongoing research in the field of costume patterns it is worth to introduce the current structure of costume patterns at this point. As with cloud computing patterns, there are also specific extensions in order to incorporate special needs for costume patterns, which are explained in the following. The general metamodel is extended at three locations depicted in Figure 10. The extension points one and two detail Intro and Sections according to costume patterns, while extension point three details the format solutions are captured into.

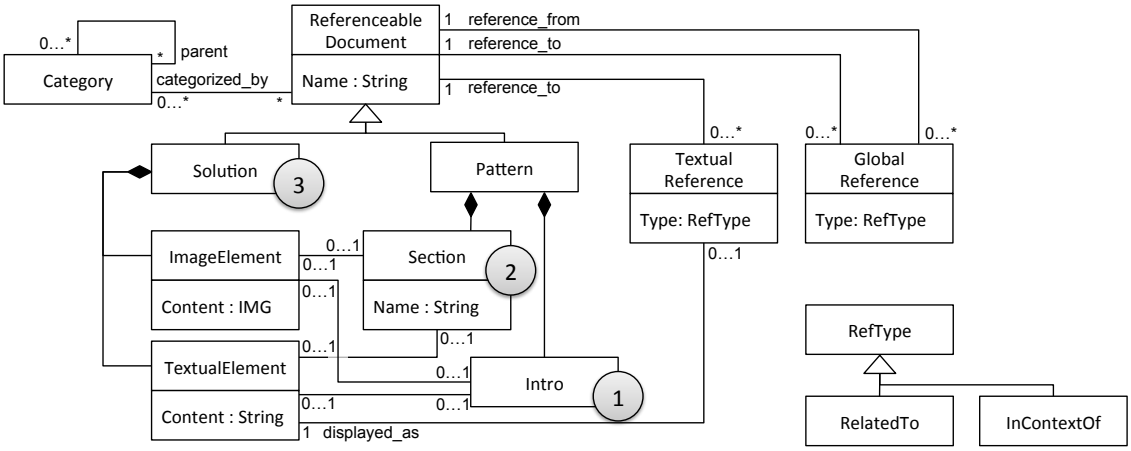


**Figure 10 - Overview of Extension Points for Costume Patterns**

### 4.2.1. Extension of the Pattern Format

The costume pattern format likewise extends the metamodel by costume specific intro elements as well as sections. Figure 11 lists the two intro elements *Purpose* and *Icon* used in costume patterns to specify the extension point one. A costume pattern contains an icon that illustrates the patterns quintessence similarly to cloud computing patterns. Also, it additionally contains a brief textual description of its purpose – similar to the intent of cloud computing patterns. Thus, a reader can realize the effect of a costume without studying the pattern as a whole.

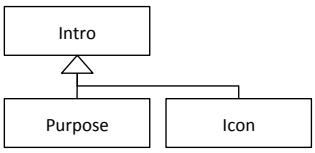


**Figure 11 – Extension for specific Intro Elements for Costume Patterns**

The second extension point deals with specific sections of a costume pattern as illustrated in Figure 12. While *Context* and *Related Patterns* are similar to the corresponding sections of the cloud computing patterns, there are differences in the presentation of the result and respectively the purpose. *Description* details the *Purpose* section of the intro by means of textual explanations.

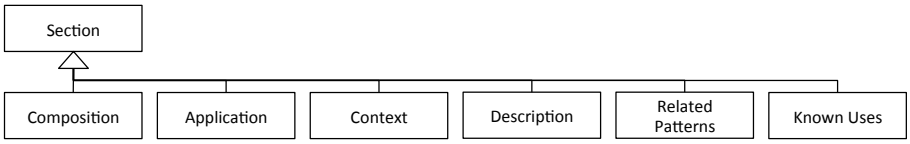


**Figure 12 - Extension for specific Sections of Costume Patterns**

The solution of a costume pattern is provided by a composition graph of all parts of a costume within the *Composition* section. The formal language of a costume's composition graph is described in Barzen & Leymann (2014). Thus, composition captures all parts a costume is made of, the so-called base elements and how they are worn layer by layer. The composition graph is represented graphically as exemplarily depicted in the excerpt of the "High-School-Queen" pattern shown on the left of Figure 13. Besides the graph structure, all base elements are textually listed as well, what is omitted in the figure for the sake of brevity.

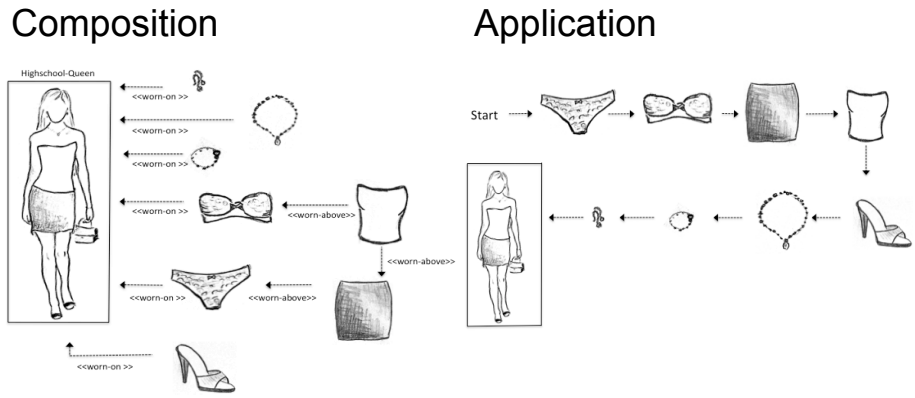


**Figure 13 - Composition and Application of the exemplarily High School-Queen Pattern**

Further, the section *Application* additionally provides graphical instructions about how to dress an actor step by step in the costume as depicted on the right of Figure 13. Finally, the *Known Uses* section covers a list of concrete solutions captured by the costume solution repository in order to establish a clear pattern provenance. Thus, costume patterns are linked to entries of the solution repository. This allows navigation from generic solution knowledge within a costume pattern to detailed concrete solution knowledge in the form of captured costumes of films.

### 4.2.2. Extension to the Solution Format

The extension of the solution element of the generic pattern metamodel shown at point three in Figure 10 is based on the technical report (Barzen, 2013) where a set of descriptive costume properties are introduced and discussed. Solutions in the domain of costumes in films are documented costumes. Thus, solutions are gathered by investigating clothes worn by roles in specific film scenes. Since costumes vary extremely in their appearance it is necessary to define a common structure to achieve comparable descriptions of costumes captured by different solution collectors. So, in contrast to text-based descriptions as presented in the solution format of cloud computing patterns, costumes are described by assigning categorical properties to costume entities. This assures that the same properties are considered and predefined categorical values are assigned for each costume. Hence, a fundus of solutions is gathered, while the solutions' properties enable to query and analyze solutions in order to identify patterns. Since a costume consists of all clothes worn by an actor in a scene it is also important to capture which items of clothing are part of the costume. The items a costume is made up are called *base elements*. Base elements are composed to a costume, so the composition structure has to be captured (Barzen & Leymann, 2014). Finally, base elements are comprised of primitives, which are the atomic elements of investigation in the context of costumes in films. In the following, those circumstances are put into an extension of the solution metamodel for costumes in films. The general metamodel adaption is depicted in Figure 14.

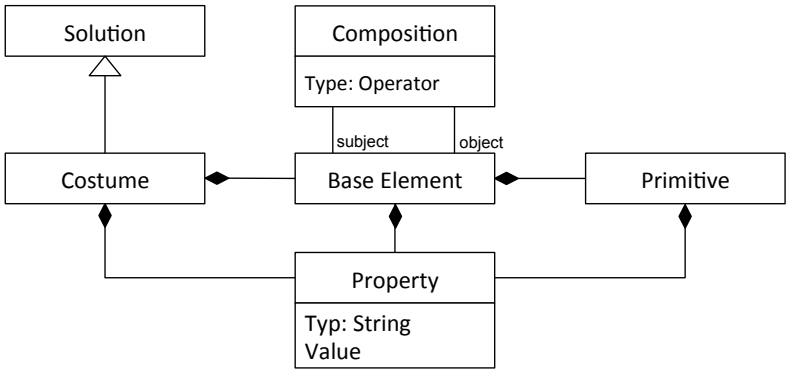


**Figure 14 - General extension of the solution metamodel for costumes**

The extended solution metamodel introduces the entities *Costume*, *Base Element* and *Primitive* as the conceptual parts a concrete costume solution consists of. While Costume captures all holistic properties valid for a costume in whole, Base Element captures properties specific to components (e.g., a jacket or a shirt etc.) of the costume. Since each base element in turn is made of several parts, properties to describe those parts are captured by Primitive. *Composition* indicates the composability of base elements in order to describe how parts of a costume are precisely worn together (e.g., shirt is worn on body, jacket is worn above shirt). For each relation between base elements of a costume, composition information is stored to finally represent a composition graph of all base elements of a costume. Consequently, a composition relates a *subject* base element via an *Operator* to an *object* base element. The semantics of a relation between two base elements are captured by the Operator as shown in Figure 15. For the sake of brevity the illustrated operators are just a subset of all available types, which can be studied particularly in Barzen & Leymann (2014) and Schumm et al. (2012).

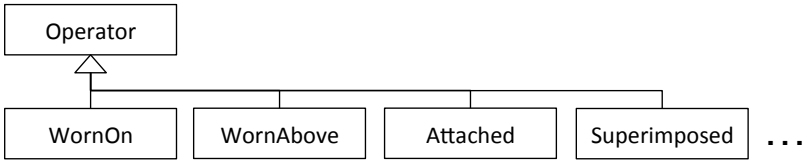


**Figure 15 - Composition Types**

Further, the class *Property* shows the generic and structured manner of the description of costumes in the way, that predefined properties are assigned to costumes, base elements as well as primitives in order to provide descriptive categorical information about them.

Before the refinement of properties is detailed for costumes the general principles valid for all properties is explained in detail. Properties are the means to define a set of characteristics, which are investigated in order to describe a costume, its base elements as well as their primitives. For instance, if we investigate how to describe the color of a base element in detail, we encounter that there is a multitude of different colors to just describe a plain yellow shirt, since there are so many nuances of the color yellow: yellow, translucent yellow, maize, bright yellow, dark yellow, brazen yellow etc. So, to handle the vast number of different possible values for properties it is inevitable to predefine a subset of values presenting the granularity needed to capture the details relevant for the specific domain. For this purpose, sets of categorical descriptive values for properties are defined by means of taxonomies to formulate a valid domain model in order to describe costumes in films (Barzen & Leymann, 2014). Figure 16 exemplarily shows branches from the taxonomy of valid values for the property color (Barzen, 2013). The depicted taxonomy limits the number of valid yellow tones

to exactly four – vanilla, lemon, postal-yellow and amber. Thus, it shows the principle to limit valid values through means of taxonomies by example. In total, the taxonomy contains 38 colors, although not all of them are depicted in the figure since the branches of achromatic, secondary, tertiary and glossy colors are not extended.

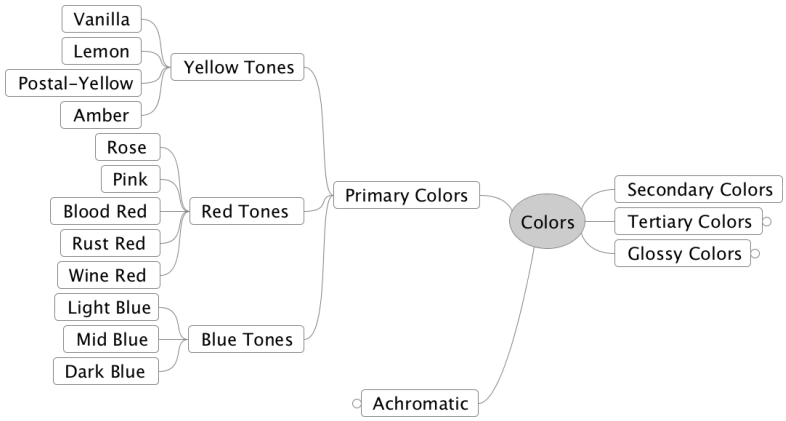


**Figure 16 - Taxonomy of valid values for the property color**

This way, a predefined set of properties limits the number of characteristics to describe a costume on the one hand and reduces the vast quantity of possible values for each property on the other hand. The set of values that are sufficient to detail each property is defined by domain experts.

### 4.2.3. Costume Properties

The detailed extension of the solution metamodel is exemplarily shown below in Figure 17 by the entity *Costume*. Properties assigned to a costume provide information about the characteristics of a costume as an complete outfit, i.e., details about the parts of a costume are not covered. Valid properties are depicted in Figure 17 as attributes of the entity Costume. *ShortText* and *SceneDescription* are used to provide a rough textual description of the costume and the scene, in which it occurs. In order to understand how often and how long a costume can be seen during a film, all occurrences are captured by means of *Timecodes*. Therefore, a *Timecode* captures the starting time and the end time of each occurrence of a costume. *DestinationOccurrence* determines if the scene is played indoors, outdoors or both, indoors and outdoors. *RelevantForStereotype* indicates if a costume has to be considered to describe the stereotype (see next section) of the role wearing the costume or not. *DominantColor* is used to define the main color of a costume that dominates, although baseelements and primitives can have additional colors to this. Same applies to *DominantFunction* and *DominantStatus*, while they define the main purpose, like if the costume contains the function of being business clothes, sport clothes or rain protection clothes etc., respectively if it is clean, damaged, wet or ironed etc. *CharacterTraits* lists all traits the role shows in the scenes while wearing the costume. This shall support to understand how costumes facilitate specific character traits of roles. A list of specific modifications of the role, such like if he or she is without make-up or if he or she is blond with curly hair etc. is provided by *ModificationsOfBody.* Since a costume often occurs several times during a film, and because the interaction of the scenery and a costume has an effect on what a costume “tells” us, also the places, where the scenes play are stored in a list of *Venues.* Each *Venue* captures the concrete venue like United States of America or Germany etc. as well as a textual description like “restaurant” or “office” etc. for that reason. Further, the impressions of age as well as optionally the known age of the character during the

scenes where the costume is worn are captured into *ImpressionsOfAge.* To understand a costume's communicative effect in relation to the period respectively to the concrete time the costume is played in a film, a list of *Seasons* is provided. Therefore each *Season* is detailed by a categorical placement in time, e.g, "Baroque" if the scene plays at that time as well as the concrete time period by use of year dates if the information is available. Finally, the daytimes a costume occurs during a film is captured in *TimesOfDay.*

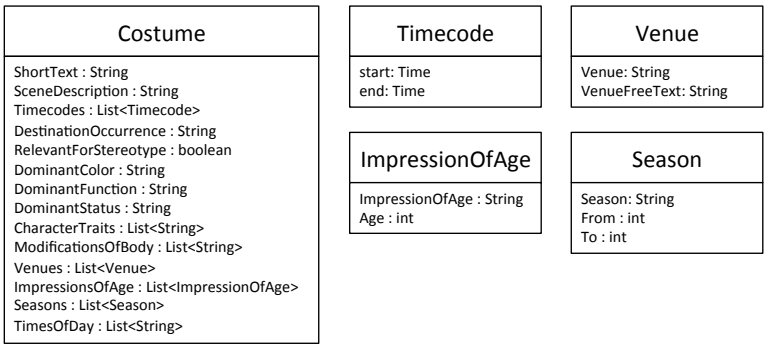


**Figure 17 - Valid properties to describe a costume**

All properties respectively the complex types of the properties are populated by categorical textual information predefined in (Barzen, 2013), numerals or boolean values that indicate either yes or no. The entities Base Element and Primitive likewise provide a predefined set of properties specific for base elements and primitives, in the manner of the explained entity Costume. For the sake of brevity, Base Element and Primitive are not described in detail at this point because the principle of predefined description categories in the form of properties is applied to all parts of a costume, thus, also to Base Element and Primitive. So, the method of extending the general metamodel to be suitable for all parts of a costume is applied accordingly to Costume.

### 4.2.4. Categorization of costume solutions by roles and films

In order to bring costume solutions in line with a corpus of films, which means a representative set of movies of a selected genre, they are related to characters as well as concrete films (Barzen & Leymann 2014). Therefore, the solution metamodel for costumes is extended by the entities *Role* and *Film* as depicted in Figure 18. They provide information to categorize costumes into genres or even the costume fundus of a specific costume designer by properties assigned to Film. Further, Role holds additional information about the generic character of a role, due to the assignment of the property *Stereotype* to each role. Stereotypes are manually assigned categories, which describe a role's behavior and thus provide a means to specify candidates for patterns. So, if several roles are annotated to be a "Sheriff" or a "Saloon-Lady" the related costume solutions provide properties that have to be queried and analyzed in order to find commonalities that make up corresponding patterns.

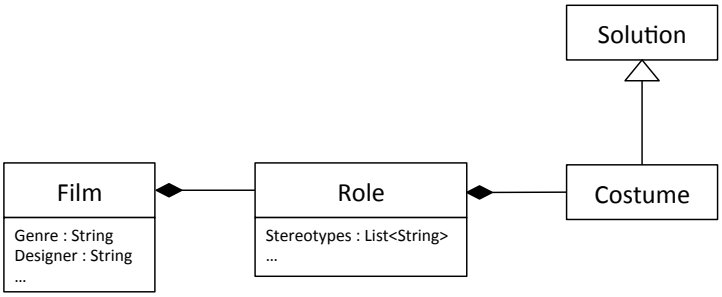


**Figure 18 - The classes Film and Role provide means to categorize costume solutions**

The introduced extension of the generic PatternPedia solution metamodel is due to the very domain specific character of solutions. In contrast to the solution format of cloud patterns that can be handled in the MediaWiki, the much more complex structure of solutions in the domain of costumes in films is supported by a custom solution repository for costumes. The extension of the generic metamodel for solutions to a very fine-grained structure of properties to enable detailed querying of the data by means of data mining and analytics methods is explained in the next section.

# 5. Implementation of the PatternPedia Tool Chain

The extensible pattern metamodel forms the basis for a web-based patterns and solutions management tool. This PatternPedia tool chain can be extended just like the metamodel to support different pattern formats, pattern reference types etc. The solution repository covered in this section is specific to the costume domain to respect the intensive metamodel adjustments covered in Section 4.2. Figure 19 displays the software stack of pattern repository on the left and the software stack of the solution repository for costumes on the right. Both stacks are based on Linux, Apache, MySQL and PHP (LAMP), while the solution analyzer component of the solution repository is additionally based on Microsoft Windows Server. As mentioned before, PatternPedia is then based on MediaWiki and its semantic extensions. Since capturing of costumes requires a very detailed data structure as mentioned in Section 4.2.2, a separate front- and backend was developed by means of Angular.js[10] and Node.js[11], to implement an optimized environment to document costumes. While Angular.js is used to provide a convenient web-based user interface the backend is realized by Representational State Transfer (REST) web services implemented in Node.js. Further, the solution analyzer is implemented upon the Microsoft business intelligence stack using Microsoft SQL Server and Microsoft Analysis Services. In the following the user interfaces shown on top of the software stacks in Figure 19 are covered in detail.

[10] https://angularjs.org/
[11] http://nodejs.org/

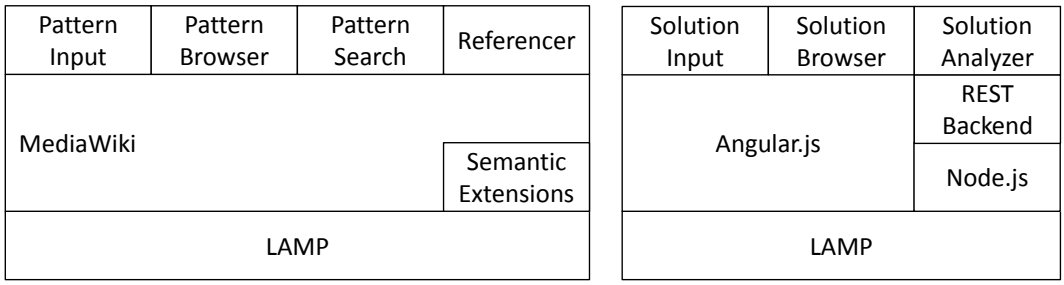


**Figure 19 - Software Stack of PatternPedia and the costume repository**

## 5.1. Pattern Browser and Pattern Search

PatternPedia visualizes pattern documents as wiki articles. The pattern name, intent, icon, and driving question are visualized similar to the book Fehling et al. (2014) to increase accessibility of the content. In the left of the screenshot shown in Figure 20, the categorization of the cloud computing patterns can be seen. The reference box seen on the upper right part of the pattern visualization is automatically generated from references contained in the textual elements of the pattern sections. Any typed references to other pattern documents are shown here grouped by their type to enable quick navigation among patterns. Search is currently based on the standard full text search functionality provided by MediaWiki. The respective search field can be seen in the upper right part of Figure 20. In Fürst (2013) additional functionality has been described to support the search of patterns based on questionnaires and pattern references. This functionality will be integrated into the PatternPedia tool chain in the future.

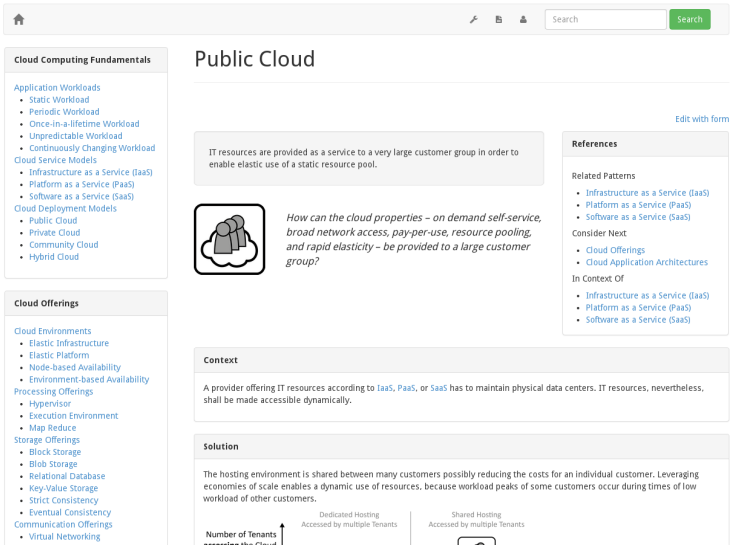


**Figure 20 - Screenshot of the Public Cloud Pattern in PatternPedia**

## 5.2. Pattern Input

Above the reference box seen in Figure 20, a link can be followed to edit a pattern document using a pattern format-specific input form. This form is shown in Figure 21. It is comprised of input fields according to the pattern format that has been specified using the pattern metamodel. References to other patterns can be added anywhere in the textual elements of these form fields.

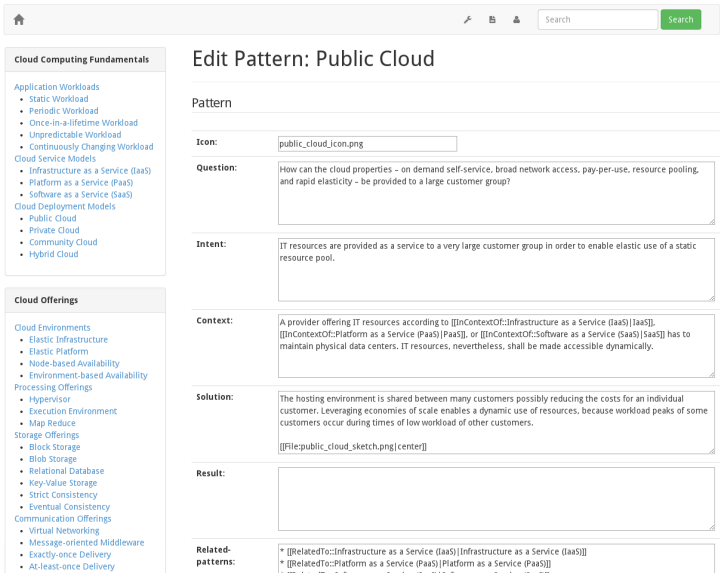


**Figure 21 - Editing the Public Cloud Pattern**

## 5.3. Pattern Referencer

The pattern references box of the pattern browser is generated based on the typed references among patterns, but other graphical visualizations may also be used. Figure 22 shows an interactive graph of the references among cloud computing patterns of the cloud fundamentals category. Pattern names are arranged in a circle with links among them as seen on the left of the figure. Whenever a user hovers one of these pattern names with the cursor all incoming and outgoing references to and from this pattern are highlighted for easier navigation.

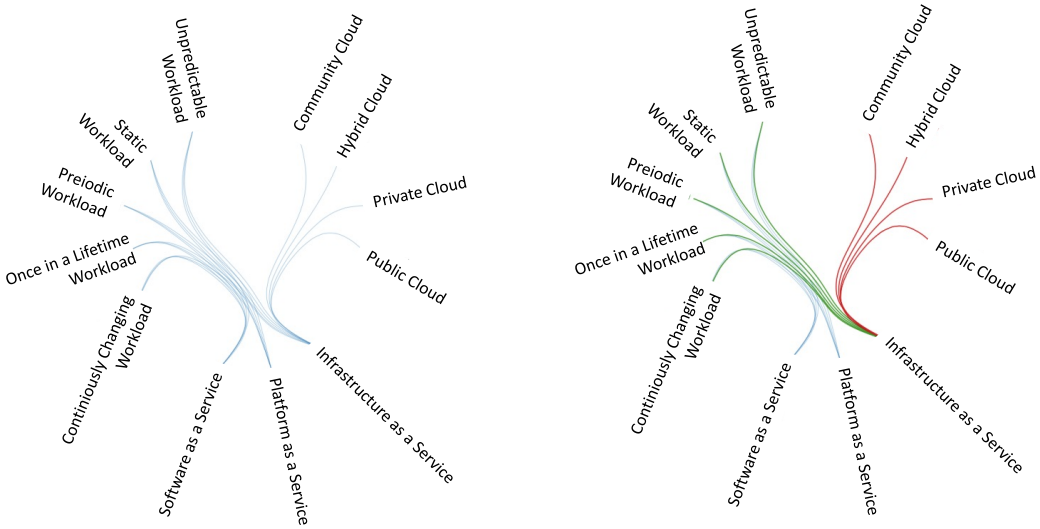


**Figure 22 - References of Patterns in the Cloud Fundamentals Category – without highlighting (left) and highlighted incoming and outgoing references of the "Infrastructure as a Service pattern" (right)**

### 5.4. Costume Browser and Costume Input

The solution repository for costumes displays costumes in a custom developed user interface. The user interface is implemented using Angular.js and, therefore, is provided by a web server as a single site web application. The website is delivered at the first request just one time from the web server to the browser and interacts afterwards only by asynchronous calls to the REST backend to load, manipulate or delete data. The display logic is handled by Angular.js in the local browser, thus, the web server is not responsible to render web sites dynamically.

The current entry point of the repository is a view containing a list of films of all captured costumes. So, a user can navigate from a film to the roles that fall under the film. Based on the roles of a film, he or she can navigate to the costumes of the roles. Therefore, in order to enter a costume to the solution repository the corresponding film and role has to be entered initially.

Angular.js is used in combination with the CSS framework Twitter Bootstrap[12] to support fast interactions with the data pool, while all control elements of the web site are especially tailored to the entry process of new costumes. This is due to the time consuming effort that has to be spend to capture costumes because of the input of many properties. Therefore, the web front end of the solution repository is optimized to allow user inputs as efficient as possible. This shows up in several specific implementations of control elements of the user interface, e.g., a special tree selector that allows browsing the taxonomies of valid values for properties. It is provided for cases when a user does not feel certain which of the values of a taxonomy to key in. So, in this case a user can expand the several branches of a taxonomy in order to navigate to the desired entry. The control element combines this with search functionality that enables to directly enter values, if the user already knows them. Further, it is often the case that multiple values have to be entered for a property. Therefore, special control elements enable to efficiently enter many values for properties. All that is exemplarily shown by an excerpt of the view to display, input and edit a costume in Figure 23 by the property character traits at the bottom.

Further, the formally described composition of base elements in Barzen & Leymann (2014) is supported by another control element depicted in Figure 24. A user can define the relation between two base elements due to the selection of a subject base element and an object base element and a specific operator that defines the semantic of the relation. By clicking the blue plus button in the upper right corner two base elements are composed with specific semantics. So, the shirt of the high school queen depicted in Figure 13 could be selected and connected with the bra using the worn above relationship.

[12] http://getbootstrap.com

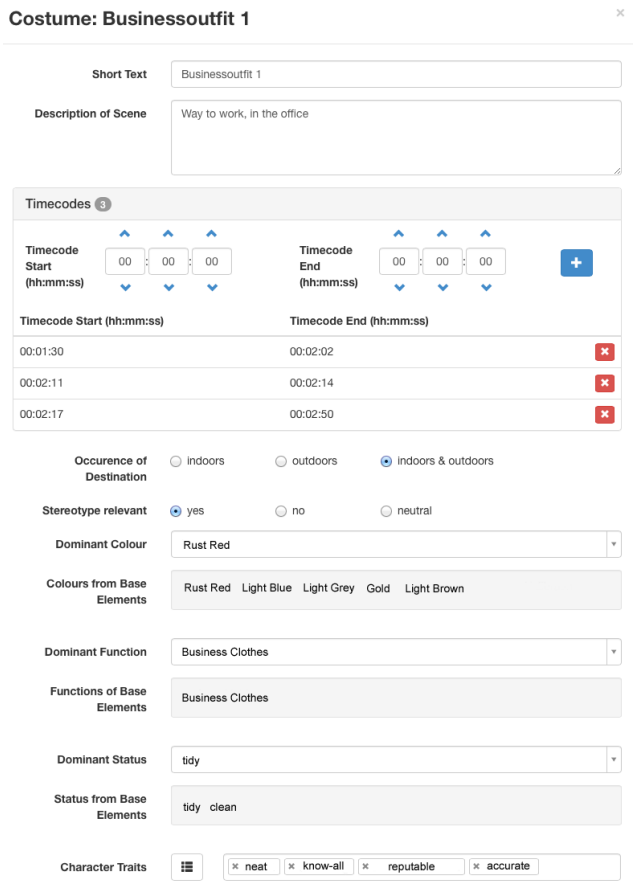


**Figure 23 - User Interface to Enter, View and Maintain a Costume**

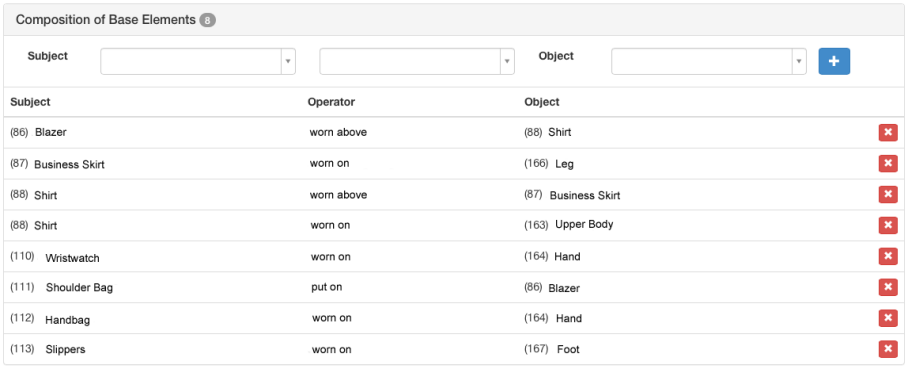


**Figure 24 - Control Element to Compose Base Elements**

## 5.5. Costume Query

In order to derive patterns from concrete solutions an equivalence function has to be applied on the fundus of solutions to indicate the similarity of several solutions (Barzen & Leymann, 2014). The extended metamodel for costumes enables to compare different solutions by means of assigned categorical properties. Thus, solutions can be investigated based on their properties to find similarities. In a current project, we investigate how to apply methods from data mining and analytics to implement the equivalence function for costumes. The current approach is to use data cubes, which are multidimensional data structures, to compare costumes. Each property of a costume is handled as a dimension of a data cube. Since the values provided by the taxonomies are of hierarchical manner, it is possible to implement

aggregation functions that allow analyzing the data pool from different aggregation levels within a data cube.

For example, to analyze the colors of costumes of a specific genre, the properties color and genre are the dimensions of a data cube. Selecting one specific genre removes all costumes from the analysis that do not belong to a role of a film of that genre. So, the complete fundus of costumes can be limited in order to analyze specific questions. Further, an aggregation function can be defined that counts all occurrences of a color in costumes. Each hierarchy level in the taxonomy is, therefore, an aggregation level within a data cube. For instance, since all specific variants of blue are hierarchically beneath blue tones, the taxonomy of colors enable to role up and drill down along its branches as illustrated in Figure 25. There, only results for the genre high school comedy are shown due to a filter applied to the genres property. Further, a list of films and costumes to those films is indicated on the left. The dimension of colors is drilled down to the most detailed level for the blue tones (rectangle one), while yellow and red tones are not expanded (rectangle two). Therefore, the number of occurrences of the color blue in costumes is shown for every fine-grained tone of blue, but is also summed up for all blue tones. Since yellow tones and red tones are not expanded, only the values of the aggregation level of the respective tones are shown. This demonstrates that the aggregation function calculates the number of occurrences for all hierarchical levels. The analysis is also supported by means of graphical representations of the data, as also depicted by the bar chart in Figure 25 at the bottom area. This enables to analyze how often several colors are used in costumes of a specific film genre and may lead to general statements about the colors of costumes for a genre.

If this approach is generalized to cover all properties of costumes, base elements, primitives, roles and films, it is possible to detect commonalities of costumes in order to abstract the essence into patterns. Thus, data cubes can be used to create implementations of the formally described equivalence function in Barzen & Leymann (2014).

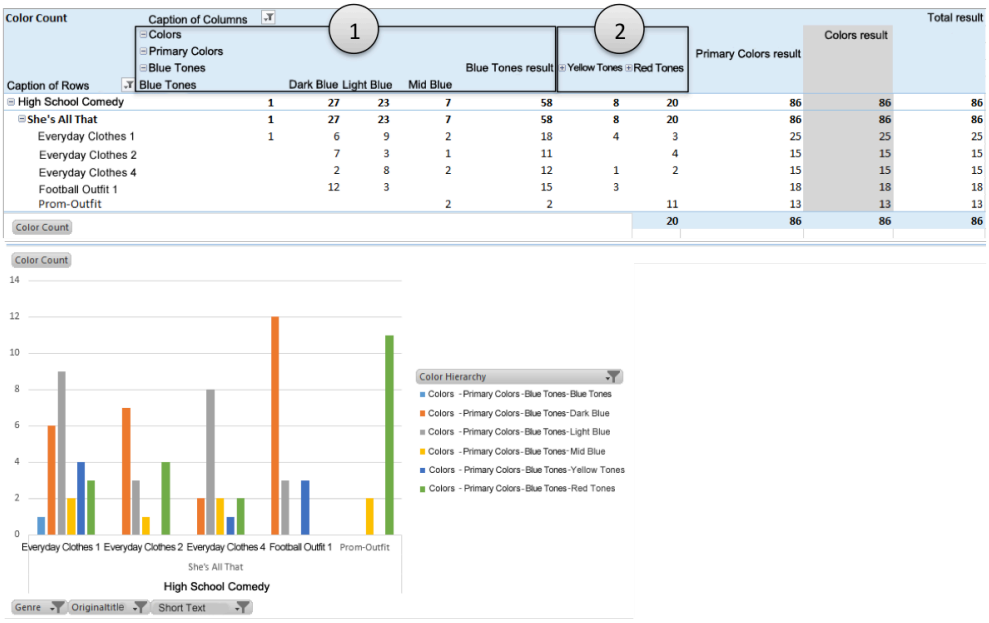


**Figure 25 - Analysis of Costume Properties by Means of a Data Cube**

## 6. Summary and Outlook

In this article, we introduced PatternPedia as a tool chain to support the capturing of existing solutions as well as the identification and authoring of patterns. An extensible pattern metamodel was used to describe the data structure supported by this tool chain to capture solutions, patterns and references among them. The tool chain was used in the domains of cloud computing and costumes with respective metamodel extensions. As the extensions for solution documents in the domain of costumes were very significant, a custom tool for the capturing of costumes has been presented.

In the future, the applicability of PatternPedia for the management of additional pattern catalogs is planned. The functional overlap with other existing pattern repositories (see Section 2) is currently being investigated aiming at the integration of complementary functionality. Especially, rating and recommendation of patterns will be considered. Another important functional aspect is the accessibility of managed patterns while the catalog grows in size. In Fehling et al. (2012), Davidkov (2014) and Strauch, Andrikopoulos, Thomas, Karastoyanova, Passow & Vukojevic-Haupt (2013) decision support methods and functionality are covered to find applicable patterns easily, for example, using questionnaires. Such functionality should be integrated with PatternPedia as well to help users find patterns applicable in their use case more quickly. Additionally, analysis functionality for the costume solution repository is going to be further developed in order to support abstraction and generalization of concrete solution knowledge into patterns.

## 7. Acknowledgements

The work published in this article was (partially) funded by the Co.M.B. project of the Deutsche Forschungsgemeinschaft (DFG) under the promotional references SP 448/27-1, LE 2275/5-1.

## 8. References

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language – Towns Buildings Construction*. Oxford: Oxford University Press.

Barzen, J. (2013). Taxonomien kostümrelevanter Parameter: Annäherung an eine Ontologisierung der Domäne des Filmkostüms. *Technischer Bericht Nr. 2013/04*, Universität Stuttgart. Retrieved from http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2013-04&mod=0&engl=0&inst=FAK.

Barzen, J., & Leymann, F. (2014). Costume Languages as Pattern Languages.

Buschmann, F., Meunier, R., Rohnert, H., & Stal, M. (1995). *Pattern-Oriented Software Architecture – A Pattern System.* New York: Wileys and Sons.

Davidkov, L. (2014). *Pattern-basierte Optimierung des Umwelteinflusses von Geschäftsprozessen* Diploma Thesis Nr. 3591, University of Stuttgart.

Fehling, C., Ewald, T., Leymann, F., Pauly, M., Rütschlin, J., & Schumm, D. (2012). Capturing Cloud Computing Knowledge and Experience in Patterns. *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD).*

Fehling, C., & Leymann, F. (2014). PatternPedia: a Wiki for Patterns. *Technical Report No. 2014/03*, University of Stuttgart.

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

Fowler, M. (2002). *Patterns of enterprise application architecture.* Boston: Addison-Wesley.

Fürst, N. (2013). *Semantisches Wiki zur Erfassung von Design-Patterns* Diploma Thesis No. 3527, University of Stuttgart.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Boston: Addison-Wesley.

Hanmer, R. (2012). *Pattern-oriented software architecture for dummies.* Chichester: John Wiley & Sons.

Hanmer, R. (2013). *Patterns for fault tolerant software*. Chichester: John Wiley & Sons.

Harrison, N. (1999). Language of Shepherding. *Proceedings of the Conference on Pattern Languages of Programs (PLoP).*

Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Boston: Addison-Wesley.

Iba, T., & Isaku, T. (2012). Holistic Pattern Mining Patterns. *Proceedings of the Conference on Pattern Languages of Programs (PLoP).*

Object Management Group (2011). *Unified Modeling Language (UML).* Available at: http://www.omg.org/spec/UML/.

Leuf, B., & Cunningham, W. (2001). *The Wiki way: quick collaboration on the Web.* Boston: Addison-Wesley.

Lucrédio, D., de Almeida, E. S., Alvaro, A., Garcia, V. C., Piveta, E. K. (2004). Student's PLoP Guide: A Pattern Family to Guide Computer Science Students during PLoP Conferences. *Proceedings of the Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP).*

Meszaros, G. (1997). Pattern Language for Pattern Writing. *Pattern languages of program design 3,* 529-574.

Raistrick, C., Francis, P., Wright, J., Carter, C., & Wilkie, I. (2004). *Model driven architecture with executable UML* (Vol 1). Cambridge : Cambridge University Press.

Reiners, R., Falkenthal, M., Jugel, D., & Zimmermann, A. (2013). Requirements for a Collaborative Formulation Process of Evolutionary Patterns to Support Knowledge Management. *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP).*

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).*

Strauch, S., Andrikopoulos, V., Thomas, B., Karastoyanova, D., Passow, S., & Vukojevic-Haupt, K. (2013). Decision Support for the Migration of the Application Database

Layer to the Cloud. *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom),* 639-646.

Wellhausen, T., & Fießer, A. (2011). How to write a pattern?. *Proceedings of the European Conference on Pattern Languages (EuroPLoP)*.

Willig, D. (2014). *Kollaborative Musteridentifikation basierend auf WIKI-Technologien* Diploma Thesis No. 3533, University of Stuttgart.

World Wide Web Consortium (2008). *Extensible Markup Language (XML) 1.0* (Fifth Edition). Available at: http://www.w3.org/TR/REC-xml/

Zdun, U. (2007). Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis. *Software: Practice & Experience*, 37 (9), 983-1016.

Zdun, U., & Avgeriou, P. (2005). Modeling architectural patterns using architectural primitives. *Proceedings of the 20th ACM Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, 133-146.

All links were last followed on 08th January 2015.

## 9. About the author/s:

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).

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 Stuttgart doing research on vestimentary communication in film.

Michael FALKENTHAL is 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 focussing on business process management, services computing and enterprise architecture management. Michael gained experience in several IT transformation and migration projects of small-, medium- and big-sized companies. His current research interests are fundamentals on pattern language theory as well as cloud computing.

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.