



**Universität Stuttgart**

# Cloud Computing Patterns

Identification, Design, and Application

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der  
Universität Stuttgart zur Erlangung der Würde eines Doktors der  
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

**Christoph Fehling**

aus Hannover

**Hauptberichter:** Prof. Dr. Dr. h. c. Frank Leymann

**Mitberichter:** Univ.Prof. Dr. Schahram Dustdar

**Tag der mündlichen Prüfung:** 07.10.2015

Institut für Architektur von Anwendungssystemen  
der Universität Stuttgart

2015



---

# Contents

---

<b>List of Acronyms</b>	<b>9</b>
<b>Zusammenfassung</b>	<b>13</b>
<b>Abstract</b>	<b>15</b>
<b>1 Introduction</b>	<b>17</b>
1.1 Terminology and Conventions . . . . .	19
1.1.1 Patterns and Pattern Languages . . . . .	19
1.1.2 Cloud Applications and Cloud Providers . . . . .	20
1.2 Problem Domain and Contributions . . . . .	21
1.2.1 Architectural Baseline of Cloud Computing . . . . .	22
1.2.2 Cloud Computing Patterns . . . . .	24
1.2.3 Cloud Computing Pattern Language . . . . .	29

1.2.4	Design Method for Cloud Applications . . . . .	31
1.3	Pattern Engineering Process . . . . .	33
1.4	Chapter Summary . . . . .	35
<b>2</b>	<b>Related Work</b>	<b>39</b>
2.1	Guidelines for Pattern Research . . . . .	40
2.1.1	Pattern Identification and Authoring . . . . .	41
2.1.2	Iterative Pattern Review . . . . .	42
2.1.3	Participating in Pattern Conferences . . . . .	42
2.2	Other Cloud Computing Patterns . . . . .	43
2.2.1	Cloud Computing Patterns of PLoP Conferences	44
2.2.2	Cloud Design Patterns (Amazon Web Services)	45
2.2.3	Cloud Design Patterns (Microsoft Azure) . . . . .	46
2.2.4	Cloud Architecture Patterns . . . . .	48
2.2.5	CloudPatterns.org . . . . .	49
2.3	Chapter Summary . . . . .	51
<b>3</b>	<b>Identification of Patterns for Cloud Computing</b>	<b>53</b>
3.1	Architectural Principles of Cloud Computing . . . . .	56
3.1.1	Cloud Computing Properties . . . . .	57
3.1.2	Impact of Cloud Computing Properties on Applications . . . . .	59
3.1.3	IDEAL Properties of Cloud Applications . . . . .	63
3.2	Cloud Application Properties in Other Architectural Styles . . . . .	66
3.2.1	Distributed Systems . . . . .	67
3.2.2	Messaging . . . . .	72
3.2.3	Representational State Transfer (REST) . . . . .	74
3.2.4	Service-Oriented Architectures . . . . .	78
3.3	Reference Cloud Application . . . . .	80
3.3.1	Cloud Runtime Environment . . . . .	81

3.3.2	Cloud Application . . . . .	82
3.3.3	Application Stack . . . . .	84
3.4	Architectural Topics for Cloud Applications . . . . .	85
3.4.1	Accounting and Controlling . . . . .	86
3.4.2	Application Migration . . . . .	87
3.4.3	Cloud Integration . . . . .	88
3.4.4	Compliance to Laws and Regulations . . . . .	89
3.4.5	Data Storage . . . . .	90
3.4.6	License Management . . . . .	91
3.4.7	Monitoring, Analysis, and Reporting . . . . .	91
3.4.8	Multi-Tenant Cloud Middleware . . . . .	92
3.4.9	Organizational Structures . . . . .	93
3.4.10	Security . . . . .	94
3.5	Information Sources for Cloud Computing Patterns . . . . .	95
3.5.1	Cloud Providers and Cloud Applications . . . . .	95
3.5.2	Existing Patterns . . . . .	96
3.6	Chapter Summary . . . . .	97

**4 Design of Cloud Computing Patterns 99**

4.1	Pattern Document Format . . . . .	102
4.2	Pattern Language Structure . . . . .	105
4.2.1	References among Patterns . . . . .	107
4.2.2	Constraints on the Pattern Language Metamodel . . . . .	108
4.3	Graphical Design . . . . .	111
4.3.1	Pattern Document Layout . . . . .	111
4.3.2	Graphical Elements Used in Patterns . . . . .	113
4.3.3	Composition of Graphical Elements . . . . .	118
4.4	Summary of Cloud Computing Patterns . . . . .	121
4.4.1	Cloud Computing Fundamentals . . . . .	123
4.4.2	Cloud Offerings . . . . .	129
4.4.3	Cloud Application Architectures . . . . .	136

4.4.4	Cloud Application Management . . . . .	142
4.4.5	Composite Cloud Applications . . . . .	145
4.5	Chapter Summary . . . . .	148
<b>5</b>	<b>Application of Cloud Computing Patterns</b>	<b>149</b>
5.1	Accessibility of Cloud Computing Patterns . . . . .	151
5.1.1	Categories of Cloud Computing Patterns . . . . .	152
5.1.2	Order of Pattern Consideration . . . . .	152
5.2	Pattern-Based Design Method for Cloud Applications . . . . .	153
5.2.1	Decomposition . . . . .	155
5.2.2	Workload . . . . .	158
5.2.3	State . . . . .	160
5.2.4	Component Refinement . . . . .	164
5.2.5	Elasticity and Resiliency . . . . .	167
5.3	Chapter Summary . . . . .	171
<b>6</b>	<b>Toolchain for Cloud Computing Patterns</b>	<b>173</b>
6.1	Pattern Authoring Toolkit . . . . .	175
6.1.1	Information Classification Template . . . . .	176
6.1.2	Pattern Document Template and Stencil Set . . . . .	178
6.2	Pattern Importer . . . . .	179
6.2.1	Pattern Import Format Editor . . . . .	179
6.2.2	Pattern Import Format Converter . . . . .	182
6.3	Pattern Repository . . . . .	183
6.3.1	Pattern Document Database . . . . .	184
6.3.2	Pattern Browser . . . . .	185
6.3.3	Pattern Recommender . . . . .	187
6.3.4	Pattern Editor . . . . .	188
6.4	Reference Implementation Repository . . . . .	189
6.5	Chapter Summary . . . . .	190

<b>7</b>	<b>Validation</b>	<b>193</b>
7.1	Use of Cloud Computing Patterns by Industry Partners	194
7.1.1	Daimler TSS GmbH . . . . .	194
7.1.2	Dr. Ing. h.c. F. Porsche AG . . . . .	200
7.2	Uninfluenced Use of Cloud Computing Patterns . . . . .	203
7.2.1	Use in Research . . . . .	204
7.2.2	Use in Industry . . . . .	206
7.3	Chapter Summary . . . . .	208
<b>8</b>	<b>Conclusions and Outlook</b>	<b>211</b>
8.1	Answers to Research Questions . . . . .	212
8.2	Limitations of Cloud Computing Patterns . . . . .	214
8.3	Research Opportunities . . . . .	215
8.3.1	Research-Driven Improvement of Patterns . . . . .	215
8.3.2	A Pattern Language for IT Applications . . . . .	216
8.3.3	Cloud Computing Patterns as Architectural Decisions . . . . .	218
8.3.4	Pattern-Based Architectural Modeling . . . . .	218
8.4	Chapter Summary . . . . .	219
<b>A</b>	<b>Detailed Pattern Engineering Process</b>	<b>221</b>
<b>B</b>	<b>Detailed Mapping of Related Patterns</b>	<b>223</b>
<b>C</b>	<b>Toolchain for Cloud Computing Patterns</b>	<b>247</b>
	<b>Bibliography</b>	<b>249</b>
	<b>Acknowledgments (Danksagungen)</b>	<b>269</b>





---

## List of Acronyms

---

<b>AADL</b>	Architecture Analysis & Design Language
<b>ACID</b>	Atomicity, Consistency, Isolation, Durability
<b>ADL</b>	Architecture Description Language
<b>AG</b>	Aktiengesellschaft
<b>API</b>	Application Programming Interface
<b>AWS</b>	Amazon Web Services
<b>BASE</b>	Basically Available, Soft state, Eventual consistency
<b>BPEL</b>	Business Process Execution Language
<b>BPMN</b>	Business Process Model and Notation

## List of Acronyms

---

<b>CAFE</b>	Composite Application Framework
<b>CAP</b>	Consistency, Availability, Partition tolerance
<b>CAPEX</b>	Capital Expenditures
<b>CDN</b>	Content Delivery Network
<b>CMS</b>	Content Management System
<b>CPU</b>	Central Processing Unit
<b>CQRS</b>	Command Query Responsibility Segregation
<b>CRM</b>	Customer Relationship Management
<b>CTO</b>	Chief Technology Officer
<b>DB</b>	Database
<b>DBMS</b>	Database Management System
<b>DDOS</b>	Distributed Denial of Service
<b>DMTF</b>	Distributed Management Task Force, Inc.
<b>DNS</b>	Domain Name System
<b>EAI</b>	Enterprise Application Integration
<b>EC2</b>	(Amazon) Elastic Compute Cloud
<b>ERP</b>	Enterprise Resource Planning
<b>ESB</b>	Enterprise Service Bus
<b>GB</b>	Gigabyte
<b>GPS</b>	Global Positioning System
<b>HATEOAS</b>	Hypermedia as the Engine of Application State

<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IDE</b>	Integrated Development Environment
<b>IDEAL</b>	Isolation, Distribution, Elasticity, Automated Management, Loose Coupling
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IP</b>	Internet Protocol
<b>IT</b>	Information Technology
<b>LLC</b>	Limited Liability Company
<b>LUN</b>	Logical Unit Number
<b>MVP</b>	Most Valuable Professional
<b>NAT</b>	Network Address Translation
<b>NFS</b>	Network File System
<b>NIC</b>	Network Interface Controller
<b>NIST</b>	National Institute of Standards and Technology
<b>OASIS</b>	Organization for the Advancement of Structured Information Standards
<b>OCL</b>	Object Constraint Language
<b>OPEX</b>	Operational Expenditure
<b>OVF</b>	Open Virtualization Format
<b>PHP</b>	PHP: Hypertext Preprocessor

## List of Acronyms

---

<b>PNG</b>	Portable Network Graphics
<b>RAM</b>	Random-Access Memory
<b>RDF</b>	Resource Description Framework
<b>REST</b>	Representational State Transfer
<b>RPC</b>	Remote Procedure Call
<b>SLA</b>	Service-Level Agreement
<b>SOA</b>	Service-Oriented Architecture
<b>SOC</b>	Service-Oriented Computing
<b>SQL</b>	Structured Query Language
<b>UML</b>	Unified Modeling Language
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>VPC</b>	(Amazon) Virtual Private Cloud
<b>VPN</b>	Virtual Private Network
<b>WAF</b>	Web Application Firewall
<b>XML</b>	Extensible Markup Language

---

## Zusammenfassung

---

Die breite Verfügbarkeit von schnellen Internetzugängen hat die Bedeutung der geographischen Position von IT Betriebsumgebungen reduziert. Fortschritte in der Verwaltungsfunktionalität von Hardware und Software erlauben es darüber hinaus IT Ressourcen sehr flexibel und automatisiert zur Verfügung zu stellen. Das Geschäftsmodell Cloud Computing entstand auf Basis dieser Entwicklungen. Cloud Anbieter stellen IT Ressourcen, wie Server, Laufzeitumgebungen oder komplette Anwendungen zur Selbstbedienung durch den Kunden über das Internet bereit. Da diese Cloud Dienste sehr große Kundengruppen erreichen, können Anbieter Skaleneffekte ausnutzen. Daher sind Cloud Dienste oft schneller betriebsbereit und günstiger als gleichartige selbstverwaltete IT Ressourcen. Insbesondere können Cloud Ressourcen oftmals flexibel reserviert und freigegeben werden, wodurch Kunden die verwendeten

Ressourcen innerhalb von Minuten dem aktuellen Bedarf ihrer Anwendungen anpassen können. Die Auswirkungen dieser Eigenschaften von Cloud Umgebungen auf die dort betriebenen Anwendungen und die resultierenden Herausforderungen waren bisher größtenteils unbekannt. Die von Anbietern bereitgestellte Dokumentation und Architekturrichtlinien waren stets auf die spezifischen Cloud Dienste zugeschnitten.

Die hier vorgestellten Cloud Computing Patterns sind untereinander verknüpfte strukturierte Dokumente, die bewährte Lösungen zu wiederkehrenden Architekturproblemen beschreiben, mit denen IT Architekten bei der Erstellung von Cloud Anwendungsarchitekturen und ihrer Verwaltung zur Betriebszeit konfrontiert werden. Diese Patterns (dt. Muster) abstrahieren von technologiespezifischen Anbieterdokumentationen und -richtlinien, sowie von existierenden Anwendungen. Daher beschreiben sie abstrakte Architekturrichtlinien, die technologieunabhängiges und langlebiges Erfahrungswissen darstellen.

Diese Arbeit beschreibt die strukturierte Identifikation der Cloud Computing Patterns, ihr textuelles und grafisches Design, um sie für Menschen leicht zugänglich zu machen, sowie eine Designmethode für Ihre Anwendung. Ein *Prozess zur Pattern-Entwicklung* beschreibt die Forschung die zu diesen Beiträgen geführt hat. *Architekturelle Eigenschaften von Cloud Umgebungen und Cloud Anwendungen* strukturieren die Cloud Computing Domäne für die Identifikation von Patterns. Weiterhin wird eine *Cloud Referenzanwendung* eingeführt. Ein *Metamodel* für die Dokumentstruktur und die Organisation von Patterns beschreibt das Design der Cloud Computing Patterns. Graphische Elemente und Ihre Komposition sichern die einheitliche Darstellung der Patterns, um Ihre Benutzerfreundlichkeit zu erhöhen. Eine *Designmethode für Cloud Anwendungen* beschreibt wie IT Architekten die Cloud Computing Patterns nutzen können, um neue Cloud Anwendungen zu erstellen.

---

## Abstract

---

The broad availability of high-speed Internet has reduced the impact of geographic location of hosting environments on the user experience provided by hosted applications. Advancements in hardware and software management have enabled hosting providers to offer such IT resources very flexibly and in an automated fashion. Cloud computing is the business model exploiting this IT evolution. Cloud providers offer numerous IT resources, such as servers, application runtime environments, or complete applications via self-service interfaces to be used by customers over the Internet. As these cloud offerings target a large number of customers, providers may leverage economies of scale. Therefore, cloud offerings are often set up more quickly and with less expense than respective IT resources managed by customers. Especially, cloud resources can commonly be provisioned and decommissioned flexibly, enabling customers to adjust resources to the current demand

of their applications within minutes. The impact of these properties of the cloud environment on hosted applications and the resulting challenges have been largely unknown. Provider-supplied documentation and architecture guidelines have been tailored to the specific cloud offerings supported by each provider.

The cloud computing patterns covered in this work are interrelated, well-structured documents, capturing proven solutions to recurring problems experienced by IT architects during cloud application architecture design and the associated runtime management. They have been abstracted from technology-specific provider documentation and guidelines, as well as from existing cloud applications. These patterns, therefore, capture abstract architecture concepts to provide technology-independent and persistent knowledge gained from experience.

This work presents the structured identification of the cloud computing patterns, their textual and graphical design to be easily accessible by humans, and a design method for their application. A *pattern engineering process* governs the research undertaken to obtain these contributions. *Architectural properties of clouds and cloud applications* structure the domain of cloud computing for pattern identification. A *cloud reference application* is also introduced. A *metamodel* of the document structure and organization of patterns describes the design of the cloud computing patterns. Reusable graphical elements and their composition ensure a consistent look and feel of patterns to increase user-friendliness. A *design method for cloud applications* describes how the cloud computing patterns can be used by IT architects in order to create new cloud applications.



# CHAPTER 1

---

## Introduction

---

The need for abstract architectural guidelines for cloud applications arose when cloud computing began to gain momentum in the industry [RJKG11]. Due to the industry-driven evolution of cloud computing, existing guidelines are often tailored to a specific cloud provider. Provider-specific terminology and products obfuscated common underlying concepts and techniques. Consequently, in large companies the need arose to abstract from provider-specific idiosyncrasies to obtain provider-independent knowledge. This knowledge should be applicable by IT architects and development teams regardless of the cloud provider used in a particular scenario: an abstract description of cloud hosting environments and cloud offerings should be provided in order

to classify and compare cloud providers and their offerings regarding the functional and nonfunctional requirements. These abstractions should also guide IT architects to select applicable cloud offerings or on-premise IT infrastructure in a given use case. Furthermore, architectural best practices should be deduced from existing cloud applications and provider-specific guidelines to describe abstractly how cloud applications should be designed, built, and managed during runtime. The resulting cloud computing patterns are the main focus of this work. They have previously been published as a book [FLR+14] to make them accessible to practitioners. The current work presents the systematic identification of these patterns and their design, and it evaluates how they are used in practice. It summarizes the cloud computing patterns and covers the fundamental cloud computing architectural principles and the design of cloud applications using these patterns. By convention, names of patterns are in italics, when used in running text. Each pattern name is followed by a number in parentheses that provides the page number of the pattern in [FLR+14] for easier reference.

In the following section, common terminologies and conventions of cloud computing (Section 1.1) that are used in this work are introduced. Then, the problem domain and contributions are covered (Section 1.2). These contributions consist of the following:

- The architectural baseline of cloud computing
- The cloud computing patterns themselves
- The organization of the cloud computing patterns necessary to make them accessible by readers
- The design method for cloud applications based on the presented cloud computing patterns

The overall process followed to engineer the cloud computing patterns is introduced in Section 1.3. Its phases of *pattern identification*, *pattern authoring*, and *pattern application* are detailed in the remaining sections of this work. Section 1.4 presents an overview of the following chapters with respect to the phases of the pattern engineering process.

### 1.1 Terminology and Conventions

This section provides background information about patterns and pattern languages, as well as cloud applications and cloud providers. Patterns are defined as human-readable documents covering proven knowledge captured from experience. Pattern languages are the interconnections of patterns using typed links; for example, to express general relations, alternatives or compositions.

#### 1.1.1 Patterns and Pattern Languages

As patterns are used in many domains, such as building architecture, object-oriented software, and education, the concept of patterns is perceived differently in different contexts [Cop14]. In this work, *patterns* are defined as structured textual documents that describe abstract problem-solution pairs to design problems recurring in a specific context. The cloud computing patterns, therefore, cover problems faced when designing, building, and managing cloud applications. Each pattern is self-contained; thus, the pattern user may access the documents in an arbitrary order. Pattern documents reference each other to guide the order of consideration by pointing to alternative patterns, commonly-used combinations of patterns, etc. In contrast to technical documentation or development guidelines, which consider specific

technologies, a pattern captures the core of a solution to a reoccurring problem [Ale78; Fow02; GHJ94; BMR+96]. Patterns are applicable to multiple cloud providers and can be implemented using various technologies; thus, patterns present timeless knowledge gained from praxis. The cloud computing patterns have been abstracted from existing applications, provider services, and other sources to describe common cloud architectural styles, components of cloud runtime environments, and components of cloud applications. The format of these pattern documents has been homogenized to present a common look and feel.

A *pattern language* of a certain domain is the set of existing patterns in this domain, their interrelations, and rules to combine them [Cop14; Zdu07]. Therefore, the pattern language addresses the common problems in the domain in order to guide the design process. In the domain of cloud computing, the cloud computing patterns must accordingly provide sufficient advice to guide the user to achieve the overall design goal of creating a cloud application or restructuring an existing one. Furthermore, these patterns need to document enough abstract concepts to ensure that pattern names may be used for communication among humans, as suggested by Hanmer and Di Martino [Han13; DiM14]. Similarly to the concept of patterns, other definitions of pattern languages exist in the various domains in which patterns are used, for example [Ale78; Bor01; MD98; BMR+96; GHJ94]. In this work, a pattern language is considered to consistently evolve over time and to comprise interrelated patterns addressing the design problems of the domain.

### 1.1.2 Cloud Applications and Cloud Providers

When discussing cloud environments, the following participant roles and entities are often referred to. A *cloud provider* makes different *cloud*

*offerings* accessible to customers. Customers are sometimes referred to as *tenants*. A tenant is a company or individual using the cloud offering. Each tenant may have a number of associate users accessing the cloud offering on its behalf. A cloud offering constitutes functionality for processing, communication, or storage to be used in cloud applications. The granularity of provided functionality is referred to as an IT resource, i.e., a server, storage space, etc. A special type of IT resource is a *Web service*: this is an application functionality that can be accessed via a well-defined interface over a network and that displays an “always on” semantic [WCL+05].

A *cloud application* is hosted in such a cloud environment and relies on different cloud offerings to provide its custom functionality. An application may experience *workload*, which is the utilization of IT resources originating from users accessing the application or automated tasks handled by the application. Workload can be measured in different forms depending on the IT resource: servers may handle more or fewer processing tasks, storage functionality may have to handle varying volumes of data, communication components may have to transfer varying quantities of data, etc.

## 1.2 Problem Domain and Contributions

This section summarizes the research questions arising in the domain of cloud computing that are addressed by the contributions of this work. First, the characterizing properties of a cloud application and how to enable them are discussed in order to develop the first contribution: an architectural baseline for cloud computing. Then, the cloud computing patterns presented in [FLR+14], as well as their textual and graphical design, are covered as second and third contributions. The fourth

contribution is the organization of these patterns into a pattern language enabling tool support for accessing the pattern language and applying the patterns. The architecture of the resulting toolchain supporting the cloud computing patterns is the fifth contribution. Finally, the design of a cloud application based on the presented patterns is covered as the sixth contribution of this work.

### 1.2.1 Architectural Baseline of Cloud Computing

According to the National Institute of Standards and Technology (NIST) [MG11], cloud computing is a business model that subsumes methods and technologies to provide IT resources flexibly over a network via a self-service interface. The NIST definition of cloud computing furthermore describes three service models leading to a certain set of properties displayed by cloud offerings (see Section 3.1.1 for a detailed discussion). The impact of these cloud offering properties on the architectural properties of applications hosted in these environments has, however, not been addressed by the NIST definition of cloud computing.

Therefore, an *architectural baseline for cloud computing* is needed. This describes the architectural properties of cloud applications that make them suitable for a cloud environment. These cloud application properties can guide the design of new applications and enable the evaluation of existing applications during a migration of an existing application to a cloud offering.

#### *Research Question 1 (Q-1): Architectural Baseline*

How can the architectural properties of cloud applications be identified?

A method for the identification and abstraction of architectural properties is required that can be used to identify these properties in the domain of cloud computing. The method will ensure the following characteristics with respect to the identified properties:

- (i) **Relevance:** the architectural properties are required by applications to benefit from the domain-specific environment properties.
- (ii) **Distinction:** the architectural properties are specific to the domain and do not originate from a broader setting.

### *Contribution 1 (C-1): IDEAL Cloud Application Properties*

The method for the identification and abstraction of architectural principles has been applied to the domain of cloud computing. A set of properties has been identified in applications, which respect the architectural principles of cloud computing. These are the IDEAL cloud application properties [FLR+14; FLR14]:

- (i) **I**solated State: State information (state of the interaction with an application and the data handled by the application) is handled by a minimal number of application components and preferably in data storage functionality offered by cloud providers. This isolation eases the scaling and resiliency with respect to failures of the application.
- (ii) **D**istribution: The application functionality is spread out among multiple components to be deployed on multiple cloud resources. As the cloud environment is by nature a large distributed system, application functionality is distributed in a similar manner.
- (iii) **E**lasticity: The application is enabled to add and remove required resources during runtime. This addition and removal occurs without affecting the application user.

- (iv) Automated Management: To react promptly to failures and changed resource demand, the corresponding management operations are automated and do not involve any human interaction.
- (v) Loose Coupling: The dependencies among application components are reduced to ease addition and removal in the scope of elasticity management and resiliency.

### 1.2.2 Cloud Computing Patterns

The cloud provider market has been and continues to be extremely diversified, as little standardization among providers exists. Each provider uses individualized terminology, functionality, access interfaces, service levels, and pricing. To benefit from economies of scale, cloud providers often target a very large number of customers, thus making the offered cloud environment a large distributed system. This distribution of IT resources in a cloud environment forces providers to consider challenges arising from this distribution, such as connectivity loss, resource failures, and communication latency, when designing their offerings. The strategies employed by providers to handle these challenges affects the complexity of the offerings' internal structure and, thus, impacts the price. Instead of handling these challenges themselves, cloud providers can relay this complexity to customers. Hosted cloud applications then have to implement the means to address these challenges; for example, to react to failing cloud resources. As a consequence, different nonfunctional behavior of functionally similar offerings can lead to varying implications on customer-built applications, which hinders the comparison of offerings.

Development guidelines for cloud applications are often provider-specific to respect the provider's individual terminology and behavioral



characteristics of provided offerings. Standardization of the cloud offerings with regard to their interfaces and supported specification formats is currently evolving [OAS12; Gro12; DMTF14; IEE15a; IEE15b], but has not been established for the entire cloud computing market. The aim of providers to ensure distinctive characteristics of their offerings in order to position their products in the market especially hinders standardization.

However, not only the comparison of cloud providers remains difficult. The comparison of established IT infrastructures used in companies with new cloud offerings is just as challenging. Often, the price to provision and maintain an IT resource is the only basis for comparison, which can make a migration to the cloud problematic if the impact of differences in behavior on hosted applications is neglected.

Therefore, persistent and generic architectural knowledge about cloud applications should be captured. As patterns have proven suitable for this purpose, *identification and abstraction of cloud computing patterns* is needed to enable the following: workforce and student education can be conducted on an abstract level that remains applicable to different providers. Such an education will be valid even as technologies and provider offerings evolve over time. Companies can establish strategic future-proof designs to be followed by developers. These designs are independent of the employed technologies. Providers could become comparable regarding the patterns they implement in their offerings and the patterns they support in implementing applications. Functionally similar cloud offerings can be compared on a conceptual architectural level. To achieve this, patterns need to address the challenges of the domain, i.e., in the domain of cloud computing, they should consider aspects required to create a cloud application.

### *Research Question 2 (Q-2): Pattern Coverage*

How can it be ensured that a set of patterns enables building a cloud application?

The structure of pattern documents is commonly followed implicitly by pattern authors and varies in different pattern domains or even between authors of the same domain. Such a varying presentation of knowledge, however, hinders its perception and accessibility, as humans understand content presented in a homogeneous fashion more easily [Pet95].

### *Research Question 3 (Q-3): Homogeneous Representation*

How can a common textual and graphical representation of patterns increase perceptibility?

A *method for pattern engineering* is needed that can be used to identify and create patterns in a domain. It considers the collection of relevant information, the identification of patterns, and their continuous evolution. This method will involve an iterative process followed by domain experts to ensure the following characteristics with respect to the created patterns:

- (i) Coverage: A set of patterns is found that enables building a cloud application. Most common challenges of the domain have, therefore, been considered in the extracted patterns answering challenges of the domain to be of use to practitioners.
- (ii) Documentation of Solutions: Rather than writing patterns from memory, a structured documentation of existing solutions will be used. This enables pattern users not only to rely on patterns for their implementations, but also to access knowledge about existing solutions.

- (iii) Unified Domain Language: The terms by which patterns will be described are defined to ensure that multiple authors use consistent terminology to refer to the same domain elements.
- (iv) Format Homogenization: Patterns are written in the same document format, comprised of consistently-formatted sections and layout. In addition to the unified domain language, similar graphical elements that follow an explicit or implicit composition language are used.

This list of desired characteristics is extended throughout this chapter to address additional research questions.

### *Contribution 2 (C-2): Cloud Computing Patterns*

Through the application of the method for pattern engineering, a catalog of 74 cloud computing patterns has been created and published as a separate book [FLR+14]. As these patterns cover abstract concepts, architectural knowledge has become usable to build applications for different providers, thus becoming persistent over time as the cloud offering market evolves. IT architects have been given an alphabet of architectural concepts in the form of patterns to design applications without considering provider-specific offerings too early in the architectural design process. Instead, IT architectures can now be described in an abstract form to be transferred to the specific offerings of different cloud providers.

### *Contribution 3 (C-3): Format of Pattern Documents and Graphical Elements*

To ensure the accessibility of the cloud computing patterns by human readers, the pattern document format has been homogenized. Based on an analysis of existing pattern languages, the pattern document format contains mandatory and optional sections and is used consistently by

all cloud computing patterns. Additionally, the graphical elements used by pattern documents have been homogenized to ensure that common concepts, such as servers or application components, are depicted consistently.

The cloud computing patterns can be divided into patterns that characterize the cloud provider and those that describe how cloud applications are built. Patterns characterizing the cloud provider can be used to compare cloud providers as well as existing IT infrastructure regarding the supported patterns. These can guide the decision to employ a particular offering. They are divided into three subcategories:

- **Cloud environments:** These patterns abstract hosting solutions used by providers and answer questions related to selecting a hosting type for an application.
- **Cloud service models:** These patterns abstract the business models according to which providers offer IT resources and also answer questions related to selecting a particular business model for an application.
- **Cloud offerings:** These patterns abstract the functional and non-functional behavior of concrete provider-supplied functionality for processing, communication, and storage.

Cloud computing patterns that describe how a cloud application is built are divided into three subcategories:

- **Cloud application architectures:** These patterns describe how applications relying on cloud offerings should be designed and built. They cover fundamental aspects, such as the decomposition of application functionality into separate components. They also reduce the dependencies between these components. Realizations

of application functionality for user interfaces, processing, data handling and integration are also addressed.

- **Cloud application management:** These patterns describe specific application components that do not provide application functionality but execute management plans during the runtime of the application to enable scalability or resiliency, for example. These management components are especially important if a cloud provider does not handle these management tasks for a customer.
- **Composite applications:** These patterns describe common combinations of the other patterns in certain use cases; for example, to handle periodic processing tasks in the cloud or to use cloud resources for backup purposes.

Therefore, unlike many other patterns, not all cloud computing patterns are implemented by the person creating a cloud computing application. The patterns characterizing cloud providers describe offered functionality and provide advice regarding the selection of a particular offering. This makes the cloud computing patterns different from patterns in many other domains, where patterns are always considered to be applied by the human user. In the domain of cloud computing, where applications largely rely on provider functionality, the concept of patterns has also been used to address the selection of an appropriate cloud provider.

### 1.2.3 Cloud Computing Pattern Language

Once cloud computing patterns have been created, they need to be made accessible to users. As patterns are written documents, they are

commonly presented as research papers or as larger collections in books. Patterns reference each other textually in the respective documents to express the notion of related patterns, alternatives, compositions, etc. The set of patterns in a domain and their interrelations form a so-called pattern language (see Section 1.1.1). As patterns are mainly managed as documents, browsing them, following interrelations, and selecting patterns for an application is currently a manual task. Following textual references between pattern documents that are spread out across different research papers, books, and online resources is, thus, very time-consuming and unintuitive.

Therefore, the *organization of cloud computing patterns* should be optimized to increase the accessibility of architectural knowledge provided by these patterns. It has to be verified that the pattern language for cloud computing contains the relevant references between patterns and that these references are easy to follow. Tool support should be established to organize patterns, and to enable references and queries. This enables pattern users to find applicable patterns and navigate the pattern language. In addition, a design process should be described that captures an order of pattern consideration based on the use case at hand; for example, the creation of a new cloud application.

### *Research Question 4 (Q-4): Pattern Organization*

How can patterns be organized and presented so that users find relevant patterns quickly?

The *method for pattern engineering* will be extended to ensure the following additional properties:

- (i) Pattern Language Structure: Patterns and their interrelations form a directed graph that can guide readers during the selection of patterns.

- (ii) **Pattern Organization Tool:** The patterns are stored in a pattern management system that allows queries to support search and recommendation of patterns.

### *Contribution 4 (C-4): Pattern Language Metamodel*

In addition to the pattern document and used graphical elements, the homogenization of pattern interrelations is described by a pattern language metamodel. It describes the semantics of references among patterns; for example, to express refinements, alternatives, or compositions. While doing so, this metamodel remains configurable to support various pattern domains. It has been used to describe the cloud computing pattern language with the specific pattern document structure, relevant link types between patterns, and design paths to enable navigation between patterns.

### *Contribution 5 (C-5): Pattern Organization Tool*

The presented pattern language metamodel forms the basis of the tool support for pattern organization, recommendation, and application. The tool architecture describes a platform to create and organize patterns. Users can access this platform to find patterns, follow pattern interrelations, and select patterns for use.

## **1.2.4 Design Method for Cloud Applications**

The pattern language contains references among patterns of different types to enable navigation. The order of patterns to be considered is then described implicitly in the pattern language. Commonly, a refinement process is followed by first considering general patterns for the overall solution structure and then refining individual aspects iteratively. The cloud computing patterns also have such an implicit ordering in which

patterns should be considered. First, deployment models and offerings are selected, then application components are built on these offerings and integrated. Finally, these components are managed during runtime. The resulting composition of patterns is specific to the considered use case. To introduce additional guidance to pattern users, the common task to create a new cloud application can be made more explicit. The implicit ordering of patterns and the refinement phases are described as an explicit process to be followed during IT architecture design. For each step of the process, a set of patterns is given that is commonly used. These patterns may be used as entry points to the pattern language in order to identify the use case-specific pattern compositions by following the references among patterns.

### *Research Question 5 (Q-5): Pattern-Based Design*

How can the selection of patterns be guided to create the architecture of a cloud application?

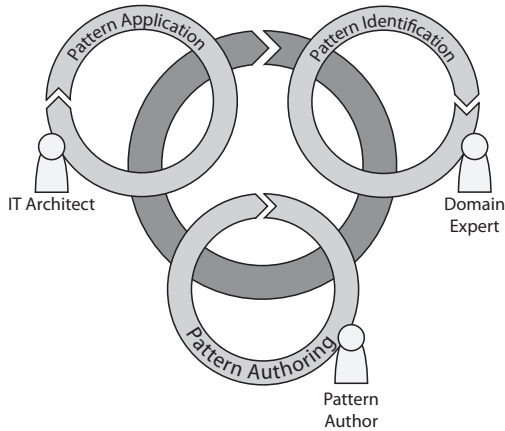
A *pattern-based design method for cloud applications* is needed to describe an overall order in which the cloud computing pattern language should be considered during the design of a new application.

### *Contribution 6 (C-6): Pattern-Based Design Method for Cloud Applications*

This method describes the order of considerations for the cloud computing patterns in different use cases:

- (i) Development of new cloud applications: A new application will be developed using the most suitable cloud provider.
- (ii) Restructuring of an existing application: The concepts of cloud computing can also be used in non-cloud applications to increase performance, availability, scalability, or other desirable application properties.





**Figure 1.1** – Pattern engineering process and participant roles (adapted from [FBBL14])

### 1.3 Pattern Engineering Process

The identification, authoring, and application of the cloud computing patterns follows the pattern engineering process depicted in Fig. 1.1. A detailed view of the process and the steps comprising each phase are given in Appendix A. This process was designed when the investigation of cloud computing patterns was initiated and presents the *method for the identification and abstraction of architectural principles* required to answer Q-1: “Architectural Baseline”. The steps comprising each phase of the process have been followed to identify, author, and apply the cloud computing patterns. The process evolved into the version presented in this work and has also been generalized to be applicable to pattern engineering efforts in various domains [FBBL14].

The set of patterns identified in a domain evolves over time. Thus, the overall pattern engineering process iterates indefinitely to identify new patterns and refine existing ones continuously. Each of the phases for pattern identification, pattern authoring, and pattern application comprising the pattern engineering process is also iterative. This is due to the fact that assumptions and decisions made during earlier iterations of a phase must subsequently be revised and adjusted. Each phase of the pattern engineering process is handled by a user who performs the steps prescribed by each phase. Each user role can be fulfilled by an individual or a group of persons.

A *domain expert* handles the pattern identification phase. During this phase, he<sup>1</sup> defines the domain and its fundamental characteristics. Also, the information to be collected about existing solutions, provider documentation, and other details is structured, and collection formats are negotiated. This is especially important if multiple persons are involved in the information collection phase to homogenize their work and results. In the scope of the cloud computing patterns, characteristics of cloud environments and cloud applications have been identified. A list of existing applications and providers to be analyzed has been compiled, followed by a collection and classification of these sources.

The *pattern author* analyzes the collected information to identify recurring patterns. These are then drafted and revised in several iterations with an experienced pattern author [Har99] and in a larger author group – so-called writers’ workshops [LAA+04]. Pattern documents follow a certain structure and reference other pattern documents using well-defined interrelations. This pattern format and reference types are also designed during this phase. In the scope of the cloud computing

---

<sup>1</sup>For reasons of readability, the male form is used throughout this work; however, the female form is also always implied.

patterns, a UML<sup>2</sup>-based metamodel and OCL<sup>3</sup> constraints have been used to define the pattern document structure and references among pattern documents. Furthermore, the graphical elements used in pattern documents have been homogenized to create a common look and feel.

An *IT architect* or the more general pattern user is the main beneficiary of the created patterns and employs them during the pattern application phase. This phase covers different means to make the cloud computing pattern language accessible by guiding pattern users during the consideration of patterns for their use case. In the scope of the cloud computing patterns, the abovementioned metamodel ensures their organization to be made accessible to IT architects. Furthermore, the pattern-based design method describes how to create a new cloud application and the patterns that should be considered.

### 1.4 Chapter Summary

This chapter has introduced a common terminology and has provided an overview of the research questions addressed by this work. The contributions answering these research questions are supported by the peer-reviewed publications shown in Table 1.1. The remainder of this work is structured with respect to the pattern engineering process introduced in the previous section:

---

<sup>2</sup><http://www.uml.org/>

<sup>3</sup><http://www.omg.org/spec/OCL/>

### **Chapter 2: Related Work**

This chapter covers related work on the identification and authoring of patterns in general and on other patterns for cloud computing. Existing guidelines on the identification, authoring, and improvement of patterns are integrated with the pattern engineering process. Other cloud computing patterns are compared with the patterns presented in this work.

### **Chapter 3: Identification of Patterns for Cloud Computing**

This chapter addresses the research questions Q-1: “Architectural Baseline” and Q-2: “Pattern Coverage”. The fundamental architectural properties enabling cloud applications to benefit from a cloud environment are derived from the properties of the cloud environment leading to the contribution C-1: “IDEAL Cloud Application Properties”. These architectural properties of cloud applications are compared with other architectural styles: distributed systems in general, messaging architectures, REST architectures, and service-oriented architectures. A reference cloud application and a list of architectural challenges are used to identify relevant provider documentation and guidelines, as well as existing applications and patterns from other domains. Based on this structuring, the cloud computing patterns have been identified in information sources.

### **Chapter 4: Design of Cloud Computing Patterns**

This chapter addresses the research questions Q-3: “Homogeneous Representation” and Q-4: “Pattern Organization”. It describes the cloud computing patterns with respect to their document structure (C-3: “Format of Pattern Documents and Graphical Elements”), the types of interrelations among patterns (C-4: “Pattern Language Metamodel”), and the graphical elements that are employed. Therefore, this chapter covers the pattern metamodel (format) and the pattern language metamodel (references). These models have been summarized into a single model

for better accessibility. A summary of all cloud computing patterns is also given (C-2: “Cloud Computing Patterns”).

### **Chapter 5: Application of Cloud Computing Patterns**

This chapter addresses the research questions Q-4: “Pattern Organization” and Q-5: “Pattern-Based Design”. It extends the organization of pattern documents by considering how they can be made accessible to pattern users. Then, it covers a pattern-based development method (C-6: “Pattern-Based Design Method for Cloud Applications”) for cloud applications using the cloud computing patterns to create new applications.

### **Chapter 6: Toolchain for Cloud Computing Patterns**

This chapter covers the tools supporting the creation, organization, and recommendation of cloud computing patterns (C-5: “Pattern Organization Tool”). It describes a pattern authoring toolkit to be used by pattern authors in order to identify and create patterns. These patterns, as well as existing ones, may then be imported into a wiki-based pattern management system, where they can be further refined, searched, and recommended to pattern users.

### **Chapter 7: Validation**

This chapter investigates the applicability of the cloud computing patterns and the pattern-based design method, as well as supporting tools in two use cases of industry partners. Also, uses of the cloud computing patterns in the scientific research community and in industry settings are investigated.

### **Chapter 8: Summary and Outlook**

This chapter reflects on the presented research contributions. It discusses related work that may be combined with the cloud computing patterns considered in this work, which may open new scientific research areas.

**Table 1.1** – Peer-reviewed publications in support of the contributions

Q-1: “Architectural Baseline”: How can the architectural properties of cloud applications be identified?	C-1: “IDEAL Cloud Application Properties”
C. Fehling et al. <i>Cloud Computing Patterns</i> . Springer, 2014 [FLR+14]*	
C. Fehling and R. Mietzner. “Composite as a Service: Cloud Application Structures, Provisioning, and Management.” In: <i>it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik</i> 53.4 (2011), pp. 188–194 [FM11]	
Q-2: “Pattern Coverage”: How can it be ensured that a set of patterns enables building a cloud application? Q-3: “Homogeneous Representation”: How can a common textual and graphical representation of patterns increase perceptibility?	C-2: “Cloud Computing Patterns” C-3: “Format of Pattern Documents and Graphical Elements”
C. Fehling et al. <i>Cloud Computing Patterns</i> . Springer, 2014 [FLR+14]*	
C. Fehling et al. “A Process of Pattern Identification, Extraction, and Application.” In: <i>Proceedings of the European Conference on Pattern Languages of Programs (EuroPLOP)</i> . 2014 [FBBL14]	
C. Fehling et al. “An Architectural Pattern Language of Cloud-based Applications.” In: <i>Proceedings of the Conference on Pattern Languages of Programs (PLOP)</i> . 2011 [FLR+11]	
C. Fehling et al. “Service Migration Patterns.” In: <i>IEEE International Conference on Service Oriented Computing and Application (SOCA)</i> . 2013 [FLR+13]	
C. Fehling et al. “Flexible Process-based Applications in Hybrid Clouds.” In: <i>Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)</i> . 2011 [FLS+11]	
Q-4: “Pattern Organization”: How can patterns be organized and presented so that users find relevant patterns quickly?	C-4: “Pattern Language Meta-model” C-5: “Pattern Organization Tool”
C. Fehling et al. “PatternPedia – Collaborative Pattern Identification and Authoring.” In: <i>Proceedings of Pursuit of Pattern Languages for Societal Change (PURPLSOC) – Preparatory Workshop</i> . 2014 [FBFL14]	
C. Fehling et al. “Capturing Cloud Computing Knowledge and Experience in Patterns.” In: <i>Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)</i> . 2012 [FEL+12]	
Q-5: “Pattern-Based Design”: How can the selection of patterns be guided to create the architecture of a cloud application?	C-6: “Pattern-Based Design Method for Cloud Applications”
C. Fehling et al. “Pattern-Based Development and Management of Cloud Applications.” In: <i>Future Internet</i> 4 (2012), pp. 110–141 [FLRS12]	
C. Fehling, F. Leymann, and R. Mietzner. “A Framework for Optimized Distribution of Tenants in Cloud Applications.” In: <i>Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)</i> . 2010 [FLM10]	
C. Fehling, F. Leymann, and R. Retter. “Your Coffee Shop Uses Cloud Computing.” In: <i>Internet Computing, IEEE</i> 18.5 (2014), pp. 52–59 [FLR14]	

\* The book containing the cloud computing patterns supports two contributions.

## CHAPTER 2

---

### Related Work

---

This chapter focuses on two groups of related work: First, existing identification and authoring techniques for patterns<sup>4</sup> that have been integrated with the pattern engineering process (Section 2.1) are summarized. Second, other patterns in the domain of cloud computing are compared with the cloud computing patterns presented in this work (Section 2.2).

Other related work is covered in other chapters, where relevant. Architectural properties of clouds and architectural styles that are related to

---

<sup>4</sup>Refer to Section 1.1.1 for a definition of the pattern concept used in this work.

cloud computing have been reviewed as part of the pattern engineering process and are covered Section 3.2. In these sections, the impact of cloud application properties and the relation of other architectural styles to the architectural principles of cloud applications are discussed in detail. Similar, existing patterns of other architectural styles related to cloud computing, for example, patterns for software architecture [BMR+96], object-oriented applications [GHJ94], and messaging applications [HW03], are summarized in Section 3.5 as they have been considered during the information collection phase of the pattern engineering process. Design methods related to the creation of cloud application architectures are mapped to the phases of the pattern-based design method for cloud applications covered in Section 5.2. An introduction to the topic of cloud computing and its history is not given here, but can be obtained from [FLR+14]. A discussion of related work that uses the cloud computing patterns is covered as part of the evaluation in Section 7.2.

### **2.1 Guidelines for Pattern Research**

The pattern engineering process introduced in Section 1.3 integrates existing guidelines on the identification and authoring of patterns. These are used without modification and considered during the respective activities. The existing works help pattern authors to write pattern documents and review them in consultation with experienced pattern authors.

All of these guidelines covered in this section have been integrated into the pattern authoring phase of the pattern engineering process detailed in Chapter 4.



### 2.1.1 Pattern Identification and Authoring

Hanmer [Han12] introduces patterns that involve mining other patterns. These describe considerations of how to structure an overall set of patterns, connect them, and ensure that the problems covered by the patterns address relevant problems of the considered domain. Wellhausen and Fießler [WF11] describe how to make pattern documents easily accessible. The target audience is mainly authors who are new to pattern writing. Considered topics include the structure of pattern documents and the semantics of the comprising sections. This structure has also been respected in the pattern language metamodel covered in Section 4.2. Furthermore, these authors provide motivation to consider how a pattern reader will access the pattern document: they describe how the relevant content and applicability of the pattern to a given use case can be identified quickly.

Meszaros and Doble [MD98] also describe best practices for pattern authors on the writing of pattern documents. These best practices are themselves captured as patterns, forming a “pattern language for pattern writing”. The authors have also covered the topic of the pattern document structure, which has influenced the pattern language metamodel (see Section 4.2). Furthermore, they focus on the naming of patterns and the naming of references among these patterns to ensure that the used terms are comprehensible and that these terms structure the pattern language adequately to the design process used in the domain. These naming conventions have also significantly influenced all cloud computing patterns and have been intensively discussed during shepherding and writers’ workshops: pattern names should capture the essence of the patterns described and should be easily usable within sentences to contribute to the natural language of IT architects. In

particular, nouns are preferred over verbs in pattern names so that patterns can be easily referred to as entities in sentences.

### 2.1.2 Iterative Pattern Review

Once the initial version of a pattern document has been drafted, it should be reviewed by a larger community. This is especially important to ensure an adequate level of abstraction: the pattern document has to describe concepts generically enough to be applicable in different use cases, while remaining understandable by readers of different skill levels. Harrison [Har99] reviews the best practices to be applied during this iterative review - called *shepherding* - in a pattern format. During this process, an experienced pattern author (*shepherd*) gives feedback to the pattern author (*sheep*) on different aspects of the pattern documents; for example, structure, style, and content during multiple review iterations. The “language of shepherding” describes how the shepherd and the sheep interact and which topics they consider in each iteration of the shepherding process.

### 2.1.3 Participating in Pattern Conferences

Conferences following the Pattern Languages of Programs (PLoP) format are very interactive due to the writers’ workshops that are scheduled during these events and result in a significant amount of feedback from the community. In order to prepare new participants, Lucrédio et al. [LAA+04] provide patterns related to the attendance of such conferences. The authors describe situations that may occur, problems that may arise, and how to resolve them. Another aspect is the preparation

for conferences that help attendees to prepare the feedback they will provide to other authors during writers' workshops.

## 2.2 Other Cloud Computing Patterns

The PLoP pattern community has developed other successful patterns for the IT domain, such as for object-oriented programming [GHJ94; BMR+96], message-based enterprise application integration [HW03], fault-tolerant systems [Han07], or distributed control systems [EKLR14]. This pattern community has been surveyed to identify existing cloud computing patterns.

As only a few patterns for cloud computing could be identified at PLoP conferences, books and online resources have also been searched for larger pattern catalogs related to cloud computing. Four pattern catalogs on cloud computing have been identified and compared with the cloud computing patterns presented in this work. Most of these patterns do not cover known uses for more than one cloud provider, or provide only examples and scenarios rather than known uses. In some cases, the presented patterns are still in development; thus, known uses and examples are necessarily omitted completely due to the early version status of the respective patterns. All four catalogs were published only a few weeks prior to submission of the cloud computing patterns book [FLR+14] for copyediting or during the copyediting process. The first version of the cloud computing patterns [FLMS11] and their discussion at research conferences [FLR+11; FEL+12] were all published one to two years prior to these other catalogs.

A possible extension of the cloud computing patterns by integration with these catalogs was considered during their evaluation. For all

patterns contained in the other catalogs, only limited redundancy with the cloud computing patterns [FLR+14] has been found to enable this identification of relations and extensions. An integration of these pattern catalogs with the cloud computing language could be promising, based on the presented mapping. Selected pattern catalogs are reviewed in the following section.

### 2.2.1 Cloud Computing Patterns of PLoP Conferences

Hashizume, Yoshioka, and Fernandez [HYF11] cover one misuse pattern for *public clouds* (62): an attacker uploads a virtual server image that other customers may use. Such “marketplaces” where customers may share virtual machine images are quite common. In this case, the attacker, however, has added malicious functionality to the image. This could be, for example, a monitoring tool that sends data handled by the virtual server to the attacker. Hashizume, Fernandez, and Larrondo-Petrie [HFL12] cover similar patterns for infrastructure as a service, platform as a service, and software as a service. In contrast to the respective cloud computing patterns, *IaaS* (45), *PaaS* (49), and *SaaS* (55), they detail the structure and behavior of these offerings using UML<sup>5</sup> models. Encina, Fernandez, and Monge [EFM14] later extend the misuse pattern for cloud computing by describing how cloud resources may be used for denial-of-service attacks. Fernandez, Yoshioka, and Washizaki [FYW14] describe two patterns for the building of firewalls in order to control network access to cloud resources. They seem to extend more general security patterns [SFH+06], which have also been used as information sources for the cloud computing patterns considered in the present work (see Section 3.5.2). Dara [Dar14] describes two patterns addressing how to enable symmetric and asymmetric encryption for access to

---

<sup>5</sup><http://www.uml.org/>

a cloud provider. Finally, Hanmer [Han14] provides for fault-tolerant cloud software – a revision of the more generic patterns of fault-tolerant software [Han07]. These revisions for the cloud, especially, reference the cloud computing patterns [FLR+14] presented here.

### 2.2.2 Cloud Design Patterns (Amazon Web Services)

Authors	Ken Tamagawa (Copyright of the site is held by Amazon Web Services LLC or its affiliates.)
URL	<a href="http://en.clouddesignpattern.org">http://en.clouddesignpattern.org</a>
Publication Date	Most of the patterns have been last modified on or around November 28, 2012.
Number of Patterns	47

The cloud design patterns for Amazon Web Services describe common designs for applications running on the Amazon cloud and are claimed to be obtained from existing applications running at this cloud provider.

A detailed mapping of the presented patterns can be found in Table B.1 in Appendix B. Of the presented patterns, 25 refine a complete cloud computing pattern or an aspect of a cloud computing pattern for the Amazon Web Services (AWS) cloud. Therefore, they cover the technical details related to how the abstract solution given by a cloud computing pattern can be implemented using the Amazon cloud. The remaining 22 patterns provide an extension of the cloud computing patterns: they cover technical best practices related to how to use the Amazon cloud offerings. Most (15) of these extensions can be mapped to an existing cloud computing pattern; thus, they describe technical practices that seem relevant in the scope of the respective pattern. It is, however,

unclear whether these best practices are also employed by any other cloud provider.

Therefore, the provided mapping of the cloud design patterns (AWS) to the cloud computing patterns can be employed by a pattern user who decides to base his application on Amazon's platform. An abstract architecture described using the cloud computing patterns can be refined for Amazon using the provided mapping. The 22 patterns that extend the cloud computing patterns should be investigated in the future for use at other cloud providers and for use in a variety of cloud applications. If such uses can be identified, a provider-independent abstraction of these patterns could be feasible in order to add them to the cloud computing pattern language.

### 2.2.3 Cloud Design Patterns (Microsoft Azure)

Authors	Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, Trent Swanson
URL	<a href="http://msdn.microsoft.com/en-us/library/dn568099.aspx">http://msdn.microsoft.com/en-us/library/dn568099.aspx</a>
Publication Date	2014
Number of Patterns	34

The cloud design patterns for Microsoft Azure provide a set of patterns for cloud applications considering distributed systems in general, cloud applications, and Microsoft Azure. While these patterns clearly focus on Azure, the authors claim that presented concepts are generally applicable. Known uses in existing applications or usability of the presented patterns in other cloud environments than Microsoft Azure are not covered in the current version.

A detailed mapping of the cloud design patterns for Windows Azure to the cloud computing patterns presented in Section 4.4 can be found in Table B.2 in Appendix B. Of the design patterns for Windows Azure, 11 provide a refinement of a cloud computing pattern or an aspect thereof to the Windows Azure platform. Therefore, these 11 patterns describe how a cloud computing pattern can be implemented using Windows Azure technologies. One cloud design pattern of data replication and synchronization generalizes several of the cloud computing patterns by discussing the need for data replication and data synchronization in general. The generalized cloud computing patterns only discuss these topics for use in a specific context where data replication or synchronization is needed. The remaining 22 cloud design patterns for Windows Azure extend the cloud computing patterns. Mostly, these patterns cover concepts of database management systems and messaging applications. These concepts are not explicitly covered by the cloud computing patterns. Instead, individual cloud computing patterns reference the respective existing related work: Hohpe and Woolf [HW03] for messaging applications, and Codd; Silberschatz, Korth, and Sudarshan; and Elmasri and Navathe [Cod70; SKS10; EN10] for databases.

Similar to the cloud design patterns for AWS, the mapping presented in this work can be considered by a pattern user who needs to refine an abstract architecture of the cloud computing patterns to the Microsoft Azure platform. The extending cloud design pattern for Windows Azure should be investigated in the future for use with different cloud providers and in multiple cloud applications to identify new abstract patterns. These extending patterns cover the following topics: (i) controlling the maximum resources that are made available to an application instance, (ii) allowing direct access to storage offerings by giving users the opportunity to bypass the application in order to in-

crease performance, and (iii) legal requirements and regulations when hosting an application in multiple data centers.

### 2.2.4 Cloud Architecture Patterns

Authors	Bill Wilder
URL	<a href="http://oreil.ly/cloud_architecture_patterns">http://oreil.ly/cloud_architecture_patterns</a>
Publication Date	September 2012
Number of Patterns	11

The cloud architecture patterns are considered to be generally applicable to cloud computing applications. Their exemplary use in a photo-sharing application throughout the book considers Windows Azure as the development platform. Other cloud providers are not considered.

A detailed mapping of the cloud architecture patterns to the cloud computing patterns can be found in Table B.3 in Appendix B. Three of the cloud architecture patterns are composed of multiple cloud computing patterns, and therefore cover the concepts of all of these patterns in a single pattern. Two of these compositions provide extensions by detailing different pricing models of cloud providers and heuristics addressing how many auxiliary resources should be provisioned to cope with sudden workload increases. Five other patterns provide more extension points by discussing (i) the graceful shutdown of resources in detail and (ii) guidelines for synchronous interactions. The cloud computing patterns currently assume that resources could potentially fail at any time, and accordingly focus on asynchronous interactions instead. Finally, three of the cloud architecture patterns are similar to or variants of the mapped cloud computing patterns.



### 2.2.5 CloudPatterns.org

Authors	Thomas Erl, Zaigham Mahmood, Amin Naserpour, Ricardo Puttini
URL	<a href="http://cloudpatterns.org/">http://cloudpatterns.org/</a>
Publication Date	May 2013 (first book), Announced for 2015 (second book), and continuously online.
Number of Patterns	88

The patterns provided on the website <http://cloudpatterns.org/> are partially sourced from the book *Cloud Computing Concepts, Technology and Architecture* by Thomas Erl, Zaigham Mahmood and Ricardo Puttini [EPM13]. The other patterns available online are authored by Thomas Erl and Amin Naserpour. The online version is often quite brief (one sentence per section of the pattern sections: problem, solution, application, etc.) and is likely to be detailed in the book *Cloud Computing Design Patterns* by Thomas Erl and Amin Naserpour, which had been announced for publication in 2015 at the time of this writing. The online catalog is divided into 20 *mechanisms*, 55 *design patterns*, and 13 *compound patterns*. The mechanisms do not follow a pattern format but cover basic concepts and techniques relevant to cloud computing. These are then referenced from the design patterns. In the current version, the compound patterns give a list of design patterns relevant to their implementation without any additional explanatory text. The covered concepts are independent of any particular cloud provider. Examples and applications are covered in generic case studies that should be applicable to different cloud providers and technologies. A concrete implementation of patterns using a specific technology is not described.

A detailed mapping of these patterns to the cloud computing patterns can be found in Table B.4 in Appendix B. Six of the patterns of cloudpatterns.org generalize concepts of cloud computing patterns and describe them in a more abstract setting. Eight of the mechanisms and design patterns available at cloudpatterns.org refine an aspect of a mapped cloud computing pattern. The concepts covered in three patterns seem similar to the contents of the cloud computing patterns. With respect to the patterns *elastic infrastructure* (87) and *elastic infrastructure* (87), the catalog of cloudpatterns.org uses more patterns to detail the components of these cloud offerings. This is reasonable, as most of the design patterns of cloudpatterns.org address the concepts of how a cloud itself can be created, and thus, how the components of cloud offerings function internally.

Thirty-five patterns and mechanisms describe the inner workings of clouds in greater detail and, thus, are categorized as being outside of the scope of the cloud computing patterns. Covered topics of virtualization technology, storage area networks, and provisioning of physical hardware are used by the cloud providers to establish a cloud environment. The cloud computing patterns, on the other hand, focus on the architecture of cloud applications, and describe how cloud offerings behave and when they should be used. The patterns of cloudpatterns.org are still evolving. Therefore, the mapping to the cloud computing patterns was not done for the 12 compound patterns, as only dependencies to other patterns are given at the time of this evaluation. The same was the case for some design patterns that are not contained in the available book [EPM13].

The current focus of the patterns of cloudpatterns.org on cloud-building technologies, concepts, and architectures motivates their mapping to the cloud computing patterns, which focus on building applications for the cloud – not the cloud itself. Through the mapping presented in

this work, the joint set of patterns can be used both by those building clouds and those that build applications hosted in these clouds. In particular, in a corporate setting, where a cloud environment is created by one department and development is handled by another department, this integration may govern the communication between IT architects, developers, and operation technicians.

### **2.3 Chapter Summary**

The presented methodologies, patterns, and guidelines of the pattern research community have been followed to identify the cloud computing patterns presented in Section 4.4. In particular, these concepts have been integrated into the pattern engineering process (see Section 1.3) organizing the overall identification, authoring and application of the cloud computing patterns presented in this work. A literature survey of past conferences of Pattern Languages of Programs (PLoP) and their European, Asian, and Indian versions has shown that cloud computing has not been investigated intensively by this community. Also, the patterns of this community have been identified that reference the cloud computing patterns introduced by this work.

Patterns in the domain of cloud computing, which were created independently from PLoP conferences, have been shown to be mostly provider-centric by focusing on a single cloud provider in the given known uses and examples. All of the investigated catalogs have shown redundancies and extensions with respect to the topics addressed by the cloud computing patterns. Therefore, a mapping of these catalogs to the cloud computing patterns has been beneficial to point to provider-specific patterns or to patterns describing how the clouds themselves are built.



## CHAPTER 3

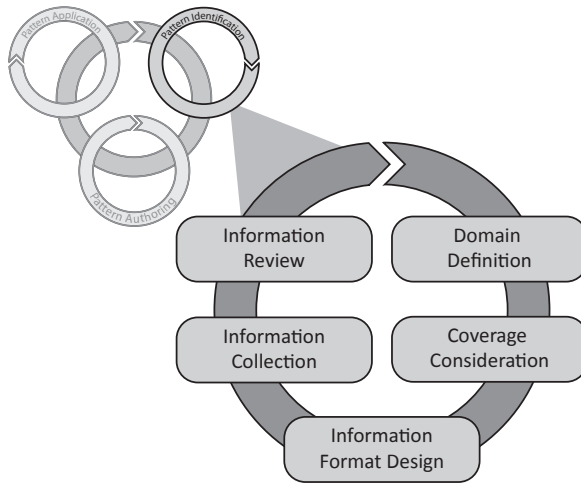
---

### Identification of Patterns for Cloud Computing

---

This chapter describes how the domain of cloud computing is structured in this work to ensure the coverage of the important challenges of this domain in the presented cloud computing patterns. This is done by addressing research questions Q-1: “Architectural Baseline” and Q-2: “Pattern Coverage”.

To answer Q-1: “Architectural Baseline”, the architectural properties of clouds are investigated to derive the requirements for cloud applications, enabling them to benefit from cloud environments. Architectural principles fulfilling these requirements are presented and compared with other architectural styles that are related to cloud computing.



**Figure 3.1** - Information collection phase for cloud computing patterns [FBBL14]

To answer Q-2: “Pattern Coverage”, a reference cloud application is introduced to abstractly describe typical components of a cloud application and a cloud environment. The cloud computing patterns describe these components comprising cloud applications and their runtime environment. Furthermore, a list of architectural topics affected by the properties of cloud computing is given to structure the identification of patterns.

The structure of this chapter is outlined according to the pattern identification phase of the pattern engineering process detailed in Fig. 3.1. Each step of this phase is briefly summarized here to illustrate how the sections of this chapter relate to the performed steps. Refer to [FBBL14] for a detailed discussion of these steps.

---

*Domain Definition:* This step lays the foundation for pattern identification by characterizing the investigated domain. Terminologies and concepts are collected and defined to establish a common knowledge base for all participants working on pattern identification. In Section 3.1, the architectural principles of cloud computing have been reviewed to derive the properties of cloud applications, enabling them to benefit from these principles. Section 3.2 compares these architectural properties of cloud applications with other related architectural styles: distributed systems, messaging, REST, and service-oriented architectures.

*Coverage Consideration:* This step ensures that relevant architectural challenges and topics are considered during pattern identification. It is investigated how information sources, from which patterns are to be identified, address these challenges and topics in their concrete environment; for example, within the scope of a particular cloud provider. To ensure coverage of the cloud computing patterns, Section 3.3 introduces a reference cloud application that characterizes the components of a cloud environment and a cloud application hosted in this environment. Section 3.4 covers the impact of cloud computing on architectural topics, thus making these topics candidates for pattern identification.

*Information Format Design:* During this step, the pattern authors agree upon a data format in which existing information sources are to be gathered. For the cloud computing patterns, the information sources were documents: provider guidelines, documentation on existing applications, books containing existing patterns of similar domains, research papers, etc. These sources were organized in a table format to be classified and summarized during later steps of the identification phase. The pattern authoring toolkit supporting the collection of information sources in this form is discussed in greater detail in Section 6.1.

*Information Collection:* During this step, information sources are collected, in the scope of the cloud computing patterns, by conducting a literature survey. The reference cloud application and the cloud architectural principles have guided this step for the cloud computing patterns: each information source, such as provider documentation, has been reviewed to identify whether it addressed a component of the reference cloud application or described how to enable an architectural property of cloud applications. Section 3.5 presents an overview of the considered information sources. All information sources that were used to describe the cloud computing patterns have been referenced as a known use of the respective pattern abstracted from them. Therefore, the complete list of information sources comprises the bibliography of [FLR+14].

*Information Review:* During this step, the information sources are classified and partially abstracted as a first step toward identification of patterns. The classification of information sources for cloud computing patterns has again been governed by the cloud reference application and the architectural properties of cloud applications. This has enabled, for example, a first grouping of cloud provider offerings with respect to the functionality they offer to the cloud reference application. Hammer [Han12] generally describes how to mine patterns from information sources and structure the resulting set of patterns. This approach has also been followed to identify the cloud computing patterns during this step of the pattern engineering process.

## **3.1 Architectural Principles of Cloud Computing**

Properties of clouds and architectural styles commonly used in cloud applications are evaluated in this section to deduce the architectural prin-



principles of cloud applications. The definition of cloud computing released by the National Institutes of Standards in Technology (NIST) [MG11] is used to establish the proposed architectural properties specific to cloud applications: isolated state, distribution, elasticity, automated management, and loose coupling (IDEAL properties). Furthermore, the properties of data handled by cloud applications are shown to have a major impact on the architecture of cloud applications.

### 3.1.1 Cloud Computing Properties

The NIST defines cloud environments to have five essential characteristics [MG11]. Furthermore, providers are defined to use one of three service models, and one of four cloud deployment types. The five *essential characteristics* of cloud environments are:

*On-Demand Self-Service:* Customers access cloud offerings independently of any human employed by the provider. Self-service is commonly realized using human-accessible Web interfaces or application programming interfaces (API). Customers can access these interfaces at their own pace and independently of the provider's business hours.

*Broad Network Access:* The bandwidth of the network connection between customer and provider is large enough to ensure an acceptable latency and throughput. This makes the performance experienced by customers independent of the geographic location of the provider's data center.

*Resource Pooling:* The cloud provider exploits economies of scale by sharing IT resources among customers. Sharing of resources also enables flexible resource use by customers, as resources no longer needed by one customer can serve others.

*Rapid Elasticity:* Customers can provision and decommission offered IT resources quickly and at any time. Self-service interfaces and resource pooling forms the basis for this flexible resource management.

*Measured Service:* Monitoring of resource use enables transparent billing and a broad range of pricing options. Many providers charge customers only for the resources that they actually use and do not expect fixed monthly or upfront investments. Still, monthly or yearly pricing options are sometimes also available. Due to this property, clouds enable customers to reduce large upfront investments – capital expenditures (CAPEX) – and move to operational costs – operational expenditures (OPEX).

In an environment displaying these five characteristics, cloud providers follow one or a combination of the following *service models*:

*Infrastructure as a Service (IaaS):* The provider offers processing, storage, and network connectivity, commonly in the form of servers or virtualized servers. Using these IT resources, customers may host their own operating systems and applications.

*Platform as a Service (PaaS):* The provider offers a runtime environment for applications. Servers, storage, network connectivity, and other resources are hidden from the customer, who creates a provider-compatible application. Providers can support custom or well-established programming languages and offer libraries to access environment functionality.

*Software as a Service (SaaS):* The provider offers a complete application. The cloud infrastructure that is employed is hidden from the customer. Customers access this application via user interfaces and can only configure the application itself.

Finally, the cloud provider has to maintain IT resources constituting the cloud environment. This is done according to four *cloud deployment models*:

*Private cloud*: One organization can access the cloud environment. The cloud may be hosted on or off the premises of the organization.

*Community cloud*: A group of organizations can access the cloud environment. Often, this group wishes to cooperate or shares common requirements regarding security, compliance, and other considerations. The cloud may be hosted on the premises of a member of this group or by a third party.

*Public cloud*: The general public can access the cloud environment. The cloud is hosted by the cloud provider.

*Hybrid cloud*: The cloud environment is composed of a combination of clouds and data centers following an arbitrary deployment type. This combination requires a level of integration functionality that is commonly managed by the customer.

### **3.1.2 Impact of Cloud Computing Properties on Applications**

The NIST characterizes the cloud environment, but does not address the properties of the applications using this environment. The environment characteristics presented by NIST originate from the enabling concepts or technologies: virtualization, elasticity, and utility computing [Ley09]:

*Virtualization*: Multiple IT resources on which a cloud environment is hosted are summarized to one abstract resource to enable the NIST

property of *resource pooling* among customers. For example, multiple physical servers are summarized to provide a hosting environment for virtual servers. The concept of hardware virtualization and its implementation in hypervisors has been well established for several decades [Gol73; Gol72; Gol71].

*Elasticity*: The size or number of IT resources used by a customer can grow or shrink as the demand of that customer changes. This concept corresponds to the NIST property of *rapid elasticity*.

*Utility Computing*: The use of IT resources is similar to the use of electricity, gas, or water (utilities). Therefore, the customer can use resources on-demand and pays only for the consumed amount. This concept is visible in the NIST properties of *on-demand self-service* and *measured service*.

To evaluate the impact of cloud properties from the NIST definition, the NIST properties relevant to the abovementioned enabling concepts and technologies have been considered to determine the requirements of the cloud application. The NIST property of *broad network access* has been omitted, as it is an enabling property of the connection network and, thus, does not describe a property of the cloud environment itself. The requirements deduced from the remaining NIST properties for cloud applications are as follows:

*On-Demand Self-Service*: The cloud provider enables the customer to provision and configure IT resources at all times and, especially, via application programming interfaces (API).

*Requirement 1 (R-1): Cloud Interface Integration*: A cloud application should integrate provider-supplied self-service interfaces to enable dynamic and automatic reconfiguration if environmental conditions change; for example, dynamic workload or resource prices.

*Resource Pooling:* Shared IT resources are dynamically assigned to customers. This implies that a cloud provider manages a large number of IT resources. Resource assignments are made with respect to the granularity of this pool; for example, virtual servers or amount of storage in gigabytes (GB).

*Requirement 2 (R-2): Redundant Resource Use:* A cloud application should rely on multiple IT resources, as the granularity offered by a provider may not meet the application's demand.

*Rapid Elasticity:* The cloud provider enables the customer to provision and decommission IT resources flexibly. This implies that the number of resources on which an application relies can be adjusted on demand.

*Requirement 3 (R-3): Resource Number Adjustment:* A cloud application should be able to flexibly change the number of resources upon which it relies to enable their addition and removal as the workload changes.

*Measured Service:* The cloud provider offers monitoring information based on which the use of the IT resources is billed. Often, this is used to enable a pay-per-use pricing model.

*Requirement 4 (R-4): Cost Management:* A cloud application should be aware of the running costs and the workload encountered in order to adjust the number of IT resources automatically.

These requirements are identical regardless of the NIST *cloud service model* or *cloud deployment model* that the cloud provider offers. These aspects of the NIST definition of cloud computing are, therefore, not considered to deduce the architectural properties of cloud applications. They have, however, been used to structure the domain of cloud computing for pattern identification.

In addition to these cloud properties, cloud computing introduced new concepts of data consistency, which has a major impact on cloud applications. Traditionally, storage offerings could be accessed in transactions comprised of multiple data manipulations that guarantee ACID behavior [BN09; GR93]: atomicity – operations of a transaction succeed as a whole or fail as a whole; consistency – the state of the database is valid before and after the transaction; isolation – concurrent transactions do not interfere with each other; and durability – after completion of a transaction, the resulting state of the database is permanent. ACID behavior of data storage may simplify the implementation of application functionality, as fewer situations regarding the propagation of data changes have to be considered. However, more coordination is required if data is spread out among multiple cloud resources, to assure a consistent state. Distribution may cause performance reduction in extremely large storage offerings that are globally distributed, which is often the case for cloud computing. That is why, according to the CAP theorem [GL02], only two out of the three properties of a storage offering can be maximized: consistency, availability, or performance. This leads to the BASE behavior [Pri08; Bre12; Vog09] of many cloud offerings, which are basically available, soft state, and eventually consistent. Therefore, availability is preferred over a precisely-known state at all times, and data consistency is only assured over time.

*Requirement 5 (R-5): Data Consistency:* A cloud application has to consider the physical locations where data is handled and the respective consistency behavior.

### 3.1 Architectural Principles of Cloud Computing

**Table 3.1** – Mapping of cloud environment requirements to cloud application properties

	Isolated State	Distribution	Elasticity	Automated Management	Loose Coupling
R-1: Cloud Interface Integration				✓	
R-2: Redundant Resource Use		✓			✓
R-3: Resource Number Adjustment			✓		✓
R-4: Cost Management				✓	
R-5: Data Consistency	✓				

#### 3.1.3 IDEAL Properties of Cloud Applications

Based on the requirements originating from the properties of clouds, the following properties of cloud applications have been deduced [FLR+14]. An overview of these properties and the requirements they address is shown in Table 3.1.

*Isolated State*: This property addresses R-5: “Data Consistency”: The ACID and BASE consistency behavior of data storage is reflected by the cloud application architecture. For every data element handled by the application, the consistency behavior acceptable by the use case of the application is considered. Furthermore, the cloud application manages *session state* – the state of interaction with users and other applications – and *application state* – the data elements handled by the application – only in a limited number of application components. Ideally, state information is provided to application components with

each request or is kept in provider-supplied storage offerings, because the management of state information can significantly hinder scaling application components and handling component failures.

*Distribution*: This property addresses R-2: “Redundant Resource Use”: Cloud resources are pooled among customers. The cloud application’s functionality is, therefore, decomposed to obtain multiple application components. These components can rely on multiple IT resources to support the distributed nature of the cloud environment.

*Elasticity*: This cloud application property addresses R-3: “Resource Number Adjustment”: Cloud resources can be added and removed flexibly. The cloud application, therefore, has to support IT resources being provisioned and decommissioned quickly to meet different workload demands. In combination with the distribution property, this means that a cloud application is *scaled out* instead of *scaled up*: resource numbers are increased rather than increasing the capabilities of individual resources. For a detailed comparison of both scaling approaches, refer to [FLR+14]. The required continuous addition and removal of resources from the application is highly dependent on the property of isolation of state: if IT resources handle the state, this state has to be synchronized or extracted upon provisioning or decommissioning, respectively. Elasticity is, therefore, more complex to realize with more application components handling the state information

*Automated Management*: This cloud application property addresses R-1: “Cloud Interface Integration”: Providers’ self-service interfaces are accessed on demand. Cloud applications, therefore, should independently handle runtime management, such as elastic scaling or failure resiliency. Human decisions and interaction are often too time-consuming to respect the flexibility of a cloud environment. In particular, if the provider offers a pay-per-use pricing model, scaling decisions should be made as



quickly as possible. The measurement of resource use described by R-4: “Cost Management”, thus, also impacts this cloud application property. Service level agreements of cloud providers often provide no assurances for the availability of individual resources — only for the possibility to initiate replacements. Under such conditions, fault tolerance has to be automated.

*Loose Coupling:* This cloud application property is an enabling factor to fulfill R-2: “Redundant Resource Use” and R-3: “Resource Number Adjustment”. As cloud resources and, thus, the application components are distributed, communication errors may occur due to numerous reasons, such as network outages, failure of communication partners, or resource number adjustments. The speed at which communication partners exchange information may also introduce complexity, if one communication partner cannot process information as quickly as others. Regarding rapid elasticity of the environment, the provisioning and decommissioning of resources becomes more complex the more these actions affect other resources and application components. For example, this could be an issue if application components have to be notified that another component instance with which they interact is being removed or has been added. Therefore, a cloud application should enable the following autonomies of loose coupling among the application components of which it is composed:

- (i) *Platform autonomy:* Application components may be implemented in different programming languages and run on different middle-ware.
- (ii) *Reference autonomy:* Interacting application components are unaware of their addresses.
- (iii) *Time autonomy:* Application components may exchange information at different speeds and at different times.

- (iv) *Format autonomy*: Application components may use different data formats to exchange information and, especially, do not necessarily have to know the formats supported by their communication partners.

These five cloud application properties constitute the architectural baseline of cloud applications. They provide a first orientation when designing a cloud application and have been used to structure the cloud computing domain within the information collection phase of the pattern engineering process: information sources from which patterns were abstracted have been reviewed for provider-specific guidance regarding how to enable the IDEAL cloud application properties in order to identify patterns.

## 3.2 Cloud Application Properties in Other Architectural Styles

Cloud computing has evolved from well-established concepts and technologies [Ley09]. Therefore, the architectural principles of cloud applications should also be visible in other existing architectural styles, while still providing a unique combination of such principles. Here, the properties of distributed systems in general [TS06; CDK05; Neu94], messaging applications [HW03], the REST architectural style [Fie00; FT02], and service-oriented architectures (SOA) [KBS05] have been evaluated. It is shown that many of the cloud application properties can be found at least partially in these established architectural styles. Applications following these architectural styles are, therefore, good candidates for migration to the cloud [FLR+13]. The differences are, however, still significant enough to justify the definition of architectural properties specific to cloud computing.

### 3.2.1 Distributed Systems

As cloud environments are comprised of multiple resources, they are distributed systems. Table 3.2 maps the properties and goals of distributed systems covered in this section to the IDEAL architectural principles of cloud applications. Most of the distributed system properties are also visible in the IDEAL principles. The grey markings indicate where a distributed system property is only considered partially or not at all. These properties have been considered during pattern identification to identify patterns handling these aspects of a distributed system in cloud applications. Tanenbaum and Steen [TS06] define such distributed systems as follows:

*“A distributed system is a collection of independent computers that appears to its users as a single coherent system.”*

Similarly, a cloud application is composed of different components (*distribution property*), yet should hide this fact from its users. Coulouris, Dollimore, and Kindberg [CDK05] describe the following properties as consequences of the structure of distributed systems:

*Concurrency:* The entities comprising the distributed system operate independently and in parallel. For example, multiple servers perform tasks independently and simultaneously while exchanging information via a network. Adding more of these entities can, therefore, increase the capacity of the system. This property matches the *distribution property* and *elasticity property* of cloud applications: the components of a cloud application also operate in parallel, and resources are added for reasons of elasticity.

*No Global Clock:* The entities of distributed systems coordinate their actions by exchanging messages rather than sharing a common notion

### 3 Identification of Patterns for Cloud Computing

**Table 3.2** – Mapping of distributed system properties (left) to cloud application properties (top) (in gray: unmapped properties that were especially considered during pattern identification)

	Isolated State	Distribution	Elasticity	Automated Management	Loose Coupling
Concurrency (Parallel Processing)		✓	✓		
No Global Clock (Coordination via Messages)					✓
Independent Failures				✓	
Connecting Users and Resources					
Access Transparency					✓
Location Transparency					✓
Migration Transparency					(✓)
Relocation Transparency					(✓)
Replication Transparency			✓		
Concurrency Transparency (User Accesses)					
Failure Transparency					✓
Persistency Transparency	✓				
Openness					
Scalability			✓	✓	

of time to determine when interactions should occur. This property is similar to *loose coupling* of cloud applications: messaging reduces the dependencies among components to make the communication more reliable.

*Independent Failures*: Entities of the distributed system may fail individually and the network connectivity may be unavailable. The system

designer has to respect and handle such failures. This property matches *automated management* of cloud applications: if resources fail, the cloud application should react automatically to enable resiliency.

Tanenbaum and Steen [TS06] furthermore introduce the following goals that should be achieved in a distributed system. As every cloud application is also distributed, these goals may also be relevant within this scope:

*Connecting Users and Resources:* Users should be enabled to access remote resources to enable sharing these resources. This is also a goal of cloud computing, which becomes apparent in the *resource pooling property* of clouds as defined by NIST [MG11]. The service models of cloud computing (see Section 3.1.1) also reflect this property of distributed systems. The style of user access does not impact the architectural principles of the cloud application itself. However, during pattern research, how these user accesses are handled should be investigated.

*Transparency:* The distributed system should hide the fact that it relies on multiple distributed resources that operate independently. This goal is visible in the *distribution property* of cloud applications: the cloud application comprises multiple components and provides one homogeneous user experience. Furthermore, Tanenbaum and Steen [TS06] describe eight aspects of transparency that should either be visible in the architectural principles of cloud applications or would form a valid aspect to be investigated during pattern research:

- (i) *Access transparency:* Different data representations of resources and the way they are accessed are hidden.
- (ii) *Location transparency:* The physical and logical location of resources is hidden.
- (iii) *Migration transparency:* Resources can be moved without impacting interacting resources or users of the system.

- (iv) *Relocation transparency*: Similar to migration, but resources can be moved while in use.
- (v) *Replication transparency*: It is hidden that multiple instances of the resource exist.
- (vi) *Concurrency transparency*: Different users may access the same resources without noticing that resources are shared.
- (vii) *Failure transparency*: If a resource fails and has to be recovered, other resources and users are not impacted.
- (viii) *Persistence transparency*: The type and location of storage used by a resource is hidden.

The *loose coupling* property of cloud applications summarizes access, location, and failure transparency. Migration transparency and relocation transparency are also related to the *loose coupling* property, as they consider the impact of resource moves. As these goals are more specific than the definition of the cloud application property, they have been investigated specifically during pattern research. *Replication transparency* is respected by the *elasticity* cloud application property: Tanenbaum and Steen [TS06] define concurrency transparency differently from the concurrency property introduced by Coulouris, Dollimore, and Kindberg [CDK05]. Rather than considering processing handled by resources, they focus on the access concurrency of application users. This goal is also not made explicit in the cloud application properties, because a cloud application does not necessarily have to be accessed by multiple users. Nevertheless, it should be investigated during pattern research. Persistence transparency is visible in the *isolation of state* cloud application property, which already states that the location where data is handled should be considered in a cloud application. Hiding the specifics of this storage, for example, the consistency behavior of data is a new aspect that was investigated during pattern research.

*Openness:* The distributed system should offer services according to standardized rules, with clear semantics and via well-defined interfaces. This enables accessibility, interoperability, and portability of distributed systems. The service models of cloud computing describe the semantics of services offered by a cloud provider. However, few standardized interfaces exist for cloud offerings. For example, standardization efforts have been undertaken by OASIS<sup>6</sup>, IEEE<sup>7,8</sup>, and DMTF<sup>9</sup>. A cloud application often has to handle challenges related to interoperability and portability. These aspects of the openness goal should, therefore, be considered during pattern research.

*Scalability:* The distributed system should be scalable with respect to three different dimensions according to Neuman [Neu94]:

- (i) *Size:* More resources can be added to the system in order to increase its capabilities.
- (ii) *Geographic:* Users and resources may be distributed globally. Thus, large physical distances between users and resources may exist, as well as significant distances between different resources.
- (iii) *Administration:* With a growing size, the system remains manageable. Especially, the system is able to span different independent administrative organizations.

Scalability with respect to the system size is covered by the *elasticity* property of cloud applications. Elasticity only places a stronger focus on the decrease of resource numbers and the time that adjustments take (see [FLR+14] for a detailed discussion). The other three aspects

---

<sup>6</sup><http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>

<sup>7</sup>[http://standards.ieee.org/develop/wg/CPWG-2301\\_WG.html](http://standards.ieee.org/develop/wg/CPWG-2301_WG.html)

<sup>8</sup>[http://standards.ieee.org/develop/wg/ICWG-2302\\_WG.html](http://standards.ieee.org/develop/wg/ICWG-2302_WG.html)

<sup>9</sup>[http://dmf.org/sites/default/files/standards/documents/DSP0263\\_1.0.1.pdf](http://dmf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf)

### 3 Identification of Patterns for Cloud Computing

---

**Table 3.3** - Mapping of messaging properties (left) to cloud application properties (top) (in gray: unmapped properties that were especially considered during pattern identification)

	Isolated State	Distribution	Elasticity	Automated Management	Loose Coupling
Asynchrony	✓				✓
Message Transformation					✓
Message Routing					✓
Frequent Collaboration					
Pipes and Filters Architecture		✓			
Messaging Endpoints					

are not covered by the cloud application properties. Therefore, these three distributed system properties constitute a valuable consideration for pattern research.

#### 3.2.2 Messaging

The properties of a distributed system according to Coulouris, Dolimore, and Kindberg [CDK05] already consider the exchange of messages to coordinate the work of entities comprising the distributed system. Hohpe and Woolf [HW03] cover patterns for such messaging systems in detail and apply them in the domain of enterprise application integration (EAI). In this domain, many existing applications supporting the business of a company need to work together by sharing data and



accessing the functionality of each other. Table 3.3 summarizes the overlap of messaging properties and cloud architecture principles. According to their messaging pattern and the covered messaging systems, the message-based integration style is successful due to the following properties:

*Asynchrony:* Communication partners do not have to be available at the same time, and delays introduced by working with a remote application are acceptable. The application components developed according to this approach, therefore, display high cohesion — they handle a high volume of local processing — and low adhesion — remote processing is made selectively.

*Message Transformation:* Integrated applications can use different internal data formats and produce differently-formatted messages. Messages are transformed during transit without affecting senders and receivers of messages.

*Message Routing:* The messaging system integrating multiple applications decides how to route and deliver messages. These messages can be prioritized differently, delivered to one or multiple receivers, etc.

*Frequent Collaboration:* Messages can be exchanged often, which allows applications to collaborate directly and immediately. In comparison with other integration styles covered by Hohpe and Woolf [HW03], such as transferring files between integrated applications, messaging enables a timely exchange of information among applications.

*Pipes and Filters Architecture:* This architectural style is used to realize message-processing components communicating through the channels. This enables the messaging system to operate on messages; for example, to handle format transformations outside the scope of the integrated applications.

*Messaging Endpoints:* Most applications do not support messaging systems natively. Messaging endpoints bridge the gap between the messaging system and the application and may, especially, hide the fact that messaging is used from integrated applications.

These properties of messaging applications are visible in many of the cloud architectural principles. Asynchronous exchange of messages nicely defines where the state information is handled: in the messages. While no behavior is specified regarding the storage of data inside the integrated applications, messaging helps to realize the *isolation of state* cloud application property. Asynchrony, message transformation, and message routing enable the aspects of autonomy demanded by the *loose coupling* property. The pipes and filters architectural style leads to application components fulfilling the *distribution* property. Frequent collaboration and messaging endpoints that hide the idiosyncrasies of messaging are not specifically respected in the cloud architectural principles, as these properties of messaging are not necessarily required to derive benefit from a cloud environment. Nevertheless, the overlap of properties makes messaging applications suitable for a cloud environment. Therefore, the messaging patterns of Hohpe and Woolf [HW03] have been reviewed for their applicability in a cloud environment, and the properties of messaging applications have been used to structure the identification of cloud computing patterns, which are presented in Section 4.4.

### 3.2.3 Representational State Transfer (REST)

REST is the architectural style of the World Wide Web. The REST architectural style is centered around resources managed on distributed

### 3.2 Cloud Application Properties in Other Architectural Styles

**Table 3.4** – Mapping of REST constraints (left) to cloud application properties (top) (in gray: unmapped properties that were especially considered during pattern identification)

	Isolated State	Distribution	Elasticity	Automated Management	Loose Coupling
Layered Client Server		✓			✓
Cache	✓				
Stateless Server	✓				
Uniform Interface					
Identification					
Manipulation through Representation					
Self-Descriptive Messages	✓				
Hypertext as the Engine of Application State					

servers. A resource in this scope can be a data element or some functionality. Fielding and Taylor [Fie00; FT02] define more than 10 different constraints that REST applications have to fulfill. These constraints form a complex graph with dependencies and refinements. For a comparison with the cloud application properties, the summary of the REST properties presented by Haupt et al. [HKLS14] is used. Table 3.4 shows these dependencies among REST constraints and the cloud application properties. The summarized properties of REST applications are:

*Layered Client Server:* A separation is introduced between clients accessing the application and servers hosting the application’s functionality and data. Functionality should be divided into layers such that one layer provides functionality to the layer above it and uses functionality of

the layer below. The result is that the dependencies among application functionalities are easier to manage.

*Cache:* Response data, i.e., the result returned by a function hosted on a server, is declared as “cacheable” or “non-cacheable”. If a result is cacheable, the client, server, or an intermediary along the communication path may store the result and use this information to answer any future requests to the same function directly and, thus, more quickly.

*Stateless Server:* The interaction of the client with the server does not use a session state, i.e., one request to the server is independent of any prior requests to that server or the information contained therein.

*Uniform Interface:* All interaction between the client and the server is based on a well-defined and fixed set of methods. In the case of HTTP, the interaction protocol of the World Wide Web, these methods are PUT, GET, POST, and DELETE [FGM+99]. PUT creates new resources on the server. GET retrieves resources. POST passes information to a resource that processes it. DELETE removes a resource.

*Identification:* resources are precisely addressable in order to be found in the distributed environment in which the REST application is hosted. In the case of the World Wide Web, this addressing is enabled by unique resource identifiers (URI).

*Manipulation Through Representations:* Each resource can have different representations to support different clients. In the scope of the World Wide Web, this means that resources handled by servers can be visualized differently for mobile phones than they are visualized for desktop browsers.

*Self-Descriptive Messages:* all information to understand and process a request is provided with that request. This is achieved through separa-

tion between the data contained in a request and metadata describing the semantics of the request.

*Hypertext as the Engine of Application State (HATEOAS)*: the resources accessed by a client directly control the next possible actions of that client. Websites of the World Wide Web enable this property by containing links to other sites. Therefore, if a client accesses a website resource, links to related resources are provided that can be used by the client application to navigate to related websites.

With respect to the cloud application properties, the layered client server constraint introduces some decomposition, similar to that of the *distribution* cloud application property. The management of dependencies among application components by introducing layers enables some of the autonomies demanded from the *loose coupling* property. The *cache* REST property is related to the *isolation of state*, because it considers where data is stored and replicated to increase performance. Caching has also been investigated during the search of cloud computing patterns. The same is true for the *stateless server* and *self-descriptive messages* REST properties – both consider where state is handled in the application. As they are more specific than the properties of cloud applications, these aspects have also been investigated during pattern research. The REST properties regarding *uniform interfaces*, the *identification* of resources, *manipulation through representations*, and *hypertext as the engine of application state* are not explicitly visible in cloud application properties. Again, as these concepts make REST applications reliable and scalable, it has been investigated whether they are also found in existing cloud applications during the pattern research presented in this work.

### 3 Identification of Patterns for Cloud Computing

---

**Table 3.5** – Mapping of SOA concepts (left) to cloud application properties (top) (in gray: unmapped properties that were especially considered during pattern identification)

	Isolated State	Distribution	Elasticity	Automated Management	Loose Coupling
Application Frontends					
Services	✓	✓		✓	
Service Repository			✓	✓	
Service Bus					✓

#### 3.2.4 Service-Oriented Architectures

Service-oriented computing (SOC) and the resulting service-oriented architectures (SOA) of applications decompose application functionality into individual services accessed via well-defined interfaces. This seems similar to the decomposition and distribution of cloud applications. According to Krafzig, Banke, and Slama [KBS05], a SOA is comprised of the following entities. Table 3.5 summarizes the relationships between these SOA concepts and cloud application properties.

*Application Frontends:* These interfaces for the human user orchestrate multiple services to provide a homogeneous user experience.

*Services:* Functionality provided by one organizational unit of a company is offered to other companies or departments [WCL+05; Zim09]. Such a service is constituted by its implementation, interface descriptions, and a service contract. The implementation realizes the provided

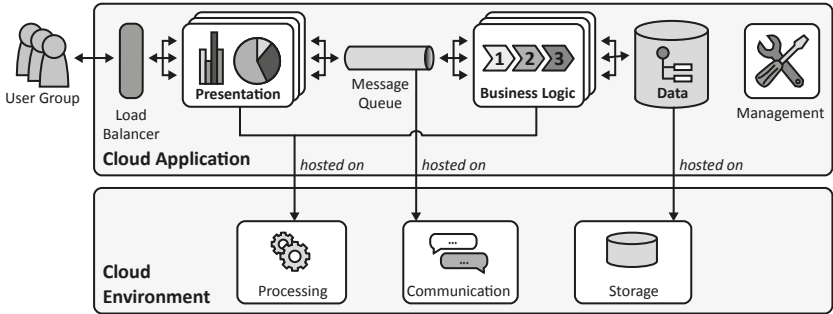
functionality. A service may have multiple interface specifications describing multiple means to access the service. The service contract describes the function of the service and its behavior – the agreement between service provider and consumer.

*Service Repository:* Services are described in a repository to be discovered by potential consumers. Discovery may be based on the service contract or other properties of the service, such as location or usage fees.

*Service Bus:* An (enterprise) service bus establishes the connectivity between service provider and consumer. By interaction through an intermediary, the assumptions that the service provider and consumer have to make about each other, such as location, availability, or data format, are reduced. These idiosyncrasies of communication are handled in the service bus. Having a service bus as an intermediary communication partner is therefore motivated by the same separation of concerns ensured by using a messaging system.

Information processed by the service functionality is often provided with each request. This concept is also part of the *isolation of state* cloud application property: state is held in a minimum number of components and is provided with requests, if possible. The concept of services also demands that the application functionality is decomposed as described by the *distribution* cloud application property. Individual services can easily be distributed among multiple cloud resources. As services have an always-on semantic, their implementation often includes some *automated management* to handle failures and changes in access numbers. This cloud application property is, however, not directly visible in the properties of a SOA. The service bus enables the cloud application properties of *elasticity* and *automated management*: if services are added, the service repository can serve as a coordinator for the number of

### 3 Identification of Patterns for Cloud Computing



**Figure 3.2** – Reference cloud application [FLR+14]

currently active service instances. The service bus also enables the same *loose coupling* property that is desired in cloud applications or message-based applications: the service bus serves as a communication intermediary that hides the concrete location of communication partners, their internal data format etc. As the notion of application frontends is not specifically mentioned in the cloud application properties, this concept has accordingly been especially considered for pattern research.

### 3.3 Reference Cloud Application

To structure and organize the search for cloud computing patterns, a reference cloud application has been used. This reference application is comprised of abstract application components, runtime offerings, and cloud environments. It is depicted in Figure 3.2. The separation between cloud runtime environment and cloud application has been made to differentiate between two categories of patterns early on: the application developer implements patterns related to the cloud application. These



patterns, thus, describe how they are realized in the scope of the cloud application and when they should be used. Patterns associated with the cloud runtime environment are not implemented, but are instead offered by the cloud provider. These patterns describe different types of environments and the offerings available in them. Therefore, they do not cover how the environment and the offerings are implemented – the cloud provider handles this. Instead, these patterns describe how the environment and offerings behave, how they are used in cloud applications, and under which conditions certain environments and offerings should be used in applications. This differentiation respects that cloud applications are almost never completely implemented by the application developer, but rely heavily on cloud-provided offerings.

The user group is accessing the reference application. Patterns associated with the user group cover different types of user behavior such as access frequency and how the cloud application can handle such varying behavior. Even though not all cloud computing patterns are implemented by the developer and capture other aspects regarding the user group and the cloud runtime environment, they ensure easier accessibility of the pattern language: through interrelations between patterns, cloud application patterns can be connected to different user behavior and cloud offerings. The entities comprising the cloud application and the cloud runtime environment are described in the following sections.

### **3.3.1 Cloud Runtime Environment**

This part of the cloud reference application contains the cloud offerings on which the cloud application is hosted. Three different types of offerings are differentiated:

*Processing:* Offerings of this type allow the execution of application functionality to handle user requests and other tasks. These offerings either host application components or provide functionality that can be integrated with application functionality.

*Communication:* Offerings of this type can be used to exchange information among application components and with other applications. Due to the high overlap among cloud architectural principles and properties of message-based applications (see Section 3.2.2), offerings for asynchronous communication have been especially considered.

*Storage:* Offerings of this type can be used to handle data. Cloud providers commonly suggest managing data in provider-supplied offerings. Some providers even demand this separation of concerns to simplify the hosting of application components, as application components of customers can be scaled more easily and replaced in case of failures if these components do not handle data.

### **3.3.2 Cloud Application**

This part of the cloud reference application is comprised of the application components that are created by the application developer. One exception is the load balancer through which users access the application:

*Load Balancer:* This component is often part of the cloud offering or enabled by well-established technology, such as the domain name system (DNS). Customers, however, need to configure this functionality to the requirements of their applications.

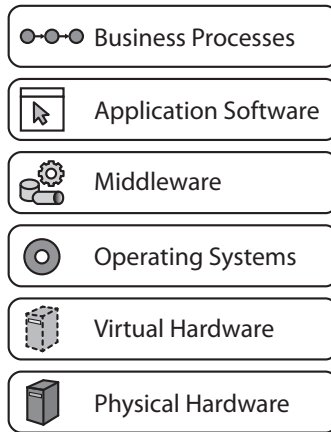
*Presentation:* This component provides the user interface through which humans can access the functionality of the application. Multiple instances of this component may be used to scale out the application. The presentation component is hosted on processing offerings of the cloud provider.

*Message Queue:* As asynchronous communication is beneficial to the cloud application properties, messaging is part of the cloud reference application. While the interaction between human users and the presentation component is synchronous, the communication between other application components is asynchronous, as enabled by message queues. The message queue is hosted on communication offerings of the cloud provider.

*Business Logic component:* This component realizes the functionality supporting the business case of the application. This functionality is generally more complex and possibly longer running than functionality provided by the presentation component. Again, the business logic component is scaled out. The business logic component is hosted on processing offerings of the provider.

*Data:* This component represents the data handled by the application and its structure. This structure and the contained data elements are specific to the application's business case. The data is hosted by the storage offerings of the provider.

*Operation Management:* This component provides functionality that is not directly required to realize the functionality offered to application users. Instead, it implements management functionality that ensures the proper operation of the application in an automated fashion. Such management functionality, for example, includes elastic scaling of the application or handling of failures. Commonly, the cloud provider



**Figure 3.3** – Application stack (adapted from [FLR+14])

supplies some offerings that can be used to configure operations management.

#### 3.3.3 Application Stack

The hardware and software stack depicted in Figure 3.3 has been defined, to further characterize cloud offerings and application components in order to guide pattern identification. It is comprised of the following layers.

*Physical Hardware:* Tangible and physical IT resources comprise this layer. Common examples are servers, networking infrastructure, power lines, etc.

*Virtual Hardware:* Physical IT resources can be abstracted to virtual ones with similar properties. For example, a physical server can host

multiple virtual servers. Virtualization is commonly used to share the underlying physical hardware among multiple virtual instances and increase the flexibility of hardware configurations.

*Operating Systems:* This software is installed directly on the physical or virtual hardware to provide functionality to higher layers, i.e., to access networking cards, hard drives, etc.

*Middleware:* Similar to an operating system that manages an application's access to physical or virtual hardware, this software provides higher-level functions to be used by custom applications. Such functionality may subsume, for example, the exchange of messages or the storage of data.

*Application Software:* Software on this layer offers business-centric functionality, such as customer relationship management (CRM) or document processing.

*Business Processes:* The application software supports the actual business activities of a company. These may, for example, include credit approvals, billing, order processing, etc.

## **3.4 Architectural Topics for Cloud Applications**

Up to this point, this chapter has introduced architectural principles of cloud applications, similar established architectural styles, and a reference cloud application with a related application stack to structure the cloud computing domain. All of these aspects have been investigated in existing applications, provider documentation, and other sources to identify patterns. Furthermore, a collection of topics that are affected by cloud computing has been distilled from interviews with industry

partners. The cloud properties and cloud application properties influence how these topics are addressed in cloud applications. Therefore, information sources and existing applications were reviewed to identify how they address these topics. As the list of topics has been created based on interviews with industry partners, it represents the view of these companies and is not necessarily complete. Topics are covered in alphabetical order.

### 3.4.1 Accounting and Controlling

The metered service NIST cloud property often manifests as pay-per-use pricing models (see Section 3.1.1). Some providers may offer lower prices for long-term provisioning, such as Amazon Reserved Instances<sup>10</sup>. Variable resources prices may also be used by cloud providers. If the utilization of the cloud itself decreases, prices are reduced accordingly. During periods of high utilization, customers have to pay more for used resources. This is, for example, the pricing model of Amazon Spot Instance<sup>11</sup>.

Therefore, the *automated management* of cloud applications should respect different pricing models. Active workload management to handle processing during times of low resource prices is also beneficial. The various payment models used by cloud providers also challenge companies to support these models in their accounting systems, keep track of running expenses at different cloud providers, and select the most economical cloud provider for their applications. The comparability of providers using different pricing models is especially challenging:

---

<sup>10</sup><http://aws.amazon.com/ec2/purchasing-options/reserved-instances/>

<sup>11</sup><http://aws.amazon.com/ec2/purchasing-options/spot-instances/>

providers may charge for different usage aspects, such as data transfer or active time of virtual servers, making an effective comparison difficult.

### 3.4.2 Application Migration

Recalling that a large overlap in architectural principles of established architectural styles has been identified in Section 3.2, existing applications are likely to exist that are suitable for a cloud environment without significant adaptation. In particular, those applications that provide information to public users are promising candidates as (i) the handled data is public, and thus, less legal regulations and corporate regulations have to be respected and (ii) the workload behavior of a public user group likely changes more frequently and to a higher degree; thus, these applications may benefit from the elasticity of clouds.

Therefore, means should be investigated to migrate applications to a cloud environment in a coordinated fashion. This is especially important because the environment found in clouds may differ significantly from the company-internal environment. The cloud provider controls more functionality, and some functionality may even be unavailable from a cloud provider. Therefore, the application's infrastructure requirements and how it integrates with cloud offerings has to be considered.

Some migration patterns have been described as an extension of the cloud computing patterns in [FLR+13]. The migration of existing applications is also considered by Andrikopoulos et al. [ASL13; ADKL14] providing a decision support framework for this task. Strauch et al. [SAB+13a], especially, consider the migration of data to a cloud

environment. Binz, Leymann, and Schumm [BLS11] propose a framework for the migration of applications based on application models. Optimization of the deployment of migrated applications is presented in [LFM+11].

### 3.4.3 Cloud Integration

It is likely that many companies will use multiple cloud environments and other data centers in parallel, as some applications may not be suitable for a cloud environment due to various reasons, such as legal or security aspects. Some legacy applications may reside in existing data centers, and moving them to a cloud environment would be too complex or uneconomical.

Therefore, means should be investigated to integrate different cloud environments with a company's data centers. The integration complexity should not be handled by every use case individually.

Existing cloud provider functionalities, such as Amazon Virtual Private Cloud (VPC)<sup>12</sup>, WSO2 Enterprise Service Bus<sup>13</sup>, or Microsoft Azure Connect<sup>14</sup>, mostly focus on the integration of a customer data center with a cloud and neglect the integration between multiple cloud environments. Cloud computing patterns for the integration of multiple clouds and data centers have been presented in Fehling et al. [FLR+14] and are summarized in Section 4.4.

---

<sup>12</sup><http://aws.amazon.com/vpc/>

<sup>13</sup><http://wso2.com/products/enterprise-service-bus/>

<sup>14</sup><http://www.windowsazure.com/en-us/home/tour/virtual-network/>



### 3.4.4 Compliance to Laws and Regulations

Companies are required to respect numerous laws and corporate regulations in their IT infrastructure setup, runtime management, and applications. Regulations often target the storage location and accessibility of data as described, for example, by the Federal Data Protection Act in Germany [Ger90]. Also, regulations often involve audits performed by external third parties. In a virtualized environment, these compliance requirements can be difficult to fulfill: cloud providers may obfuscate the exact location of data, and examining the physical hardware can be impossible for human auditors. Another challenge that companies face can be the unauthorized use of external cloud resources. As public cloud offerings may be more flexible than company internal services, departments or individual employees may be driven towards using cloud services, such as Dropbox<sup>15</sup>.

Therefore, new means have to be investigated regarding how cloud applications can fulfill existing and possibly arising compliance requirements. This especially affects the management processes for audits. Unauthorized use of cloud offerings in companies also has to be addressed, as well as strategies concerning how to reintegrate externalized data and business functionality into the corporate-internal IT infrastructure.

Regarding compliance, cloud computing introduces the challenge that a company's IT has to compete with public cloud services. Methods concerning how to address these issues have been investigated [BDA+11], but there are currently no practical solutions from which patterns can be extracted.

---

<sup>15</sup><http://www.dropbox.com/>

### 3.4.5 Data Storage

The consistency behavior of cloud storage offerings has a strong impact on cloud applications: data inconsistencies have to be respected, especially when multiple clients manipulate data. Inconsistency of data replicas in distributed systems has been well researched by Tanenbaum and Steen [TS06]. In cloud computing, consistency is often decreased to increase availability and performance of the cloud offering [GL02; Ram12; Bre12]. However, the detailed consistency behavior of cloud offerings is very provider-specific: consistency behavior may be different after creation of a data element than after an update; consistency behavior for one client can be different for two concurrently accessing clients, etc.

Therefore, the different consistency behavior of providers should be investigated to identify similar behaviors. Existing cloud applications should be considered to identify best practices to manage this storage offering behavior.

The storage offering patterns presented in Fehling et al. [FLR+14] cover some of the consistency behaviors experienced when using cloud offerings. Also, some patterns are covered to encapsulate the idiosyncrasies of storage offering interactions into specific application components. These patterns are also summarized in Section 4.4.2. Unfortunately, the consistency behavior of cloud providers is very diverse. Currently, the specific behavior of every storage offering has to be considered when developing a cloud application. Strauch et al. [SAB+13b] aim to develop a data access layer that allows interactions with different cloud providers in a standardized fashion. Strauch et al. [SAB+12] also cover patterns for the management and migration of data in a cloud environment.

### 3.4.6 License Management

Cloud properties impact existing licensing models. Hardware virtualization is often used in cloud computing to enable resource pooling among customers. This virtualization of physical hardware resources can hinder or void licenses issued per central processing unit (CPU) [Fer06] on which the licensed software is executed. Network-based licensing models [Fer06] consider the running instances of a software on a network. In an elastically scaled cloud application, these instances would be adjusted automatically to fulfill the *automated management* property of cloud applications.

Therefore, different license models have to be respected during the management of cloud applications. In particular, the optimization of resource provisioning and decommissioning should respect license costs.

In [FLM10], some means are investigated to optimize the distribution of application users among multiple instances of a software stack. These means especially consider resource and licensing costs.

### 3.4.7 Monitoring, Analysis, and Reporting

Cost savings is a major factor for the consideration of cloud computing. However, the sharing of resources among customers may also be used to reduce the environmental impact of applications [NL13a]. Either motivation for reducing resources demands that information about resource use is collected and reported to increase the desired impact of cloud computing. For example, the emission of carbon monoxide may have to be correlated to the products a company produces as customers demand information about the ecological footprint of products [Gar10].

The sharing of resources among different customers or different departments of a company may complicate such calculations.

Therefore, best practices and architectural concepts should be investigated to handle such crosscutting concerns in the monitoring and reporting of cloud applications. The *resource pooling* property of clouds introduces new challenges to this functionality.

Nowak et al. [NLS+11; NL13b] have identified patterns to manage the environmental impact of process-based applications. One aspect of these patterns is the reporting and visualization of ecological resource consumption [NKL+12].

#### **3.4.8 Multi-Tenant Cloud Middleware**

Cloud providers exploit economies of scale by sharing resources among multiple tenants [CC06]. A tenant is a legal entity, such as a company or a private customer. Each tenant may have multiple associated users who access the services of the cloud provider on behalf of the tenant. Hardware virtualization is one technology to enable the sharing of physical hardware among tenants. Sharing middleware among tenants can increase the beneficial effects of sharing resources [GSH+07]. Isolation between tenants regarding the accessibility and the experienced performance, however, has to be ensured.

Therefore, middleware, for example, the database layer [CC06], has to be redesigned to respect the fact that multiple tenants use it. Tenant isolation has to be enabled regarding accesses to the offering, experienced performance, and handled data [GSH+07]. Further challenges

regarding tenant isolation, version management, customer-specific configurations, and the shifting of tenants among different versions of the software and the runtime environment have to be considered.

The multi-tenancy patterns presented in Fehling et al. [FLR+14] and summarized in Section 4.4 cover different sharing options for application components among multiple tenants. These concepts will have to be refined for different middleware offerings, such as databases, enterprise service buses, etc. Schiller et al. [SSBM11] describe how multi-tenant functionality can be added to relational database management systems by sharing a common database schema among tenants and allowing tenant-specific schema extensions. Strauch et al. [SAGL13] present a multi-tenant enterprise service bus to enable access isolation among tenants.

### 3.4.9 Organizational Structures

The use of cloud computing can be challenging to the organizational processes of companies. Departments lose a certain degree of control over their IT infrastructure by consuming services from the cloud. This loss of control hinders the adoption of cloud computing. While the topic of organizational structures is not directly relevant to the application architecture, organizational changes may be a limiting factor for cloud computing [KSS10]. In large companies, provisioning new IT resources or accessing new IT services commonly require human management approval. Such human interactions may hinder the *automated management* property of cloud applications. If the organizational structure and processes of a company are not adjusted to respect cloud computing, the beneficial effects can be significantly reduced.

Therefore, the necessary organizational changes required for the introduction of cloud computing should be investigated. Cloud computing not only changes application architectures, but also the development and management tasks that have to be handled by company employees. The changes in human tasks and how the workforce and organizational structure of a company have to be adapted should be approached in an organized fashion.

Manns and Rising [MR04] presented patterns to introduce organizational change in companies. Coplien [Cop06] presented organizational patterns for software development. These catalogs may form a basis for the process adaptations required to use cloud computing efficiently in organizations.

### **3.4.10 Security**

Many security issues of non-cloud applications also arise in cloud environments. New threats may arise from malicious use of cloud resources and from other cloud users. In the first case, cloud resources are used for attacks, such as distributed denial of service (DDOS). The latter case includes attacks of cloud users on applications of other customers; for example, by exploiting bugs in hardware virtualization technology.

Therefore, the new security challenges arising in cloud environments should be investigated. Existing best practices likely have to be extended to manage new threats.

Many security issues of cloud applications are similar to those of non-cloud applications. Best practices to address these issues have been captured in a pattern format [SFH+06]. Encina, Fernandez, and Monge [EFM14]; Fernandez, Yoshioka, and Washizaki [FYW14]; and

Hashizume, Yoshioka, and Fernandez [HYF11] have addressed misuse and security patterns for cloud components. These patterns extend the security patterns presented by Schumacher et al. [SFH+06].

### 3.5 Information Sources for Cloud Computing Patterns

The information sources from which the cloud computing patterns presented in this work have been identified are referenced from the respective patterns. In general, the set of information sources can be divided into (i) cloud providers and cloud applications, and (ii) existing patterns in domains having architectural properties that are similar to those of cloud computing.

#### 3.5.1 Cloud Providers and Cloud Applications

The complete list of cloud offerings and existing applications that were considered can be obtained from the bibliography of [FLR+14]. These are especially referenced from cloud offerings patterns that capture the abstract behavior of cloud offerings and help pattern users to select the offering most suitable for their use case. The following large cloud providers and tools were investigated in detail. In particular, their guidelines and best practices on how to build applications using their offerings were considered: Amazon Web Services (AWS)<sup>16</sup>, Windows Azure<sup>17</sup>, VMware<sup>18</sup>, and OpenStack<sup>19</sup>.

---

<sup>16</sup><http://aws.amazon.com/>

<sup>17</sup><http://azure.microsoft.com/>

<sup>18</sup><http://vmware.com/>

<sup>19</sup><http://www.openstack.org/>

### 3.5.2 Existing Patterns

Patterns of other domains may solve similar problems to those arising in the domain of cloud computing. Some of them may be applied without alterations and should, therefore, be referenced from the cloud computing patterns. Others may be beneficial in the scope of a particular problem arising in cloud computing, even though this use may not have been intended in the original version of the pattern. Therefore, existing pattern catalogs were considered to (i) add references to existing patterns if a pattern can be applied without alterations in the domain of cloud computing and to (ii) create translation of the original pattern concept to the domain of cloud computing if a similar use could be found in the considered information sources.

Patterns for object-oriented applications by Buschmann et al. [BMR+96] and Gamma, Helm, and Johnson [GHJ94] describe objects and their relations. The components of which a cloud application is comprised, can conceptually be mapped to objects covered by these patterns. They have, therefore, been considered to identify similar concepts; for example, to enable the separation of concerns in cloud applications.

Hohpe and Woolf [HW03] describe how applications can be integrated using message exchange. Their pattern catalog starts with describing abstract concepts related to how messages are exchanged (the message channel), and how applications interact with a messaging system (the messaging adapter). These abstract concepts are subsequently refined by covering different types of adapters and message channel components. Iteratively, the pattern users select patterns to handle problems of message loss, message duplicity, ordering of messages, etc. Because messaging applications and cloud applications are both distributed



applications, these patterns were reviewed in order to be integrated with the created cloud computing patterns.

Hanmer [Han07] addresses patterns for fault-tolerant software. As cloud applications rely on many resources, errors likely occur during the runtime of a cloud application. The information sources considered in this work were reviewed to determine the use of these patterns with the aim to integrate them with the cloud computing patterns.

Schumacher et al. [SFH+06] provides security patterns that appear to be generically applicable to IT applications. They have been investigated for their applicability in cloud environments to be referenced by the cloud computing patterns, as well.

Fowler [Fow02] describes patterns for enterprise application architectures. Many of these patterns describe how business requirements can be modeled and refined toward application functions supporting the business case. Such generic patterns have been considered for integration with the cloud computing patterns. Thus, if the enterprise architecture patterns are used during the functional design of an application, they are mapped to cloud computing patterns to guide their further refinement toward a cloud application.

### **3.6 Chapter Summary**

Based on the properties of clouds defined by the NIST, requirements that a cloud application should fulfill have been deduced to benefit from such a runtime environment. The IDEAL cloud application properties have been defined with respect to these requirements. Then, related architectural styles – distributed systems, messaging, REST, and service-oriented architectures – have been compared to the cloud application

properties. This comparison has provided similarities showing that cloud computing has evolved from a set of well-established concepts and technologies. These similarities also suggest that existing applications following the compared architectural styles might be suitable for a cloud environment, as well. A cloud reference application has been introduced, and each of its abstract components was covered. The functionality of these components may then be identified in existing applications to discover best practices for their creation. Furthermore, a list of topics affected by the use of cloud computing has been introduced. This list has covered new challenges in these topics and possible solution strategies, as well as existing patterns and other related work.

Together, the (i) cloud properties, (ii) cloud application properties and properties of similar architectural styles, (iii) reference cloud application, as well as (iv) architectural topics affected by cloud computing have been used to structure the cloud computing domain and coordinate pattern search: for each information source, such as existing applications or provider documentation, it was investigated how properties have been enabled, how abstract components of the reference application have been implemented, and how the affected architectural topics have been addressed. Properties of the related architectural styles have also been investigated, especially if these properties were not covered explicitly by the IDEAL cloud application properties. The overlay of properties of the related architectural styles to those of cloud applications make applications following the related architectural styles well-suited for a cloud environment. Therefore, it is likely that cloud applications enable similar properties in their implementation without explicitly mentioning the related architectural styles.

## CHAPTER 4

---

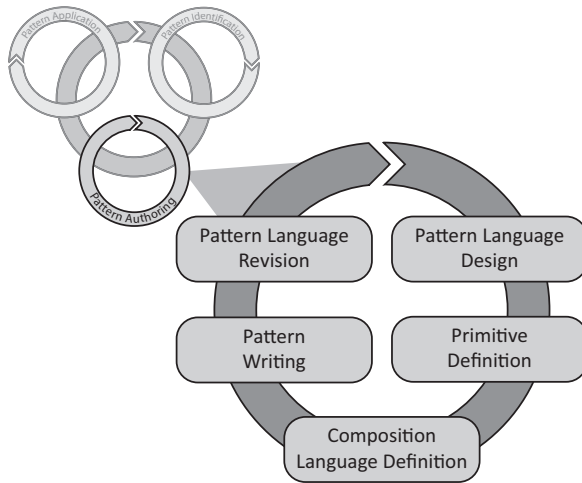
### Design of Cloud Computing Patterns

---

In this chapter, the cloud computing pattern language is presented. This includes the contributions C-3: “Format of Pattern Documents and Graphical Elements”, C-2: “Cloud Computing Patterns”, and C-4: “Pattern Language Metamodel”. A consolidated metamodel for the pattern format and the interrelations among the patterns forming the cloud computing pattern language is presented using the Unified Modeling Language (UML)<sup>20</sup>. Then, the cloud computing patterns are summarized and their evolution over time is discussed.

---

<sup>20</sup><http://www.uml.org/>



**Figure 4.1** – Pattern authoring phase for cloud computing patterns [FBBL14]

The covered topics can be mapped to the steps of the pattern authoring phase of the pattern engineering process depicted in Fig. 4.1 and introduced in Section 1.3. After the pattern identification phase, these steps iteratively define the pattern language structure and create patterns identified in the collected sources. These steps are summarized here to describe how they have been applied in the domain of cloud computing. Refer to [FBBL14] for detailed information about the pattern engineering process.

*Pattern Language Design:* The structure of pattern documents and their interrelations are defined during this step. Section 4.1 covers the pattern document format and the semantics of its graphical elements and sections. Section 4.2 describes the language structure of the cloud computing patterns by employing a UML model.

---

*Primitive Definition:* This step defines the basic elements used by patterns of a domain. This can include naming conventions and definitions that ensure a consistent use of such terms and concepts in the pattern documents. The cloud computing patterns have been homogenized using the graphical primitives covered in Section 4.3.

*Composition Language Definition:* Many pattern languages use icons and sketches to support the written text. Some pattern languages have a formal specification for the composition of such icons and sketches. For example, Buschmann et al. [BMR+96] and Gamma, Helm, and Johnson [GHJ94] use UML as a composition language for their pattern of object-oriented applications. For the cloud computing patterns, composition was not specified in such a formal manner, as the conformity of created icons and sketches to composition guidelines could be ensured manually. Section 4.3.3 covers guidelines addressing the composition of graphical elements. Together with the graphical elements created during the last step, these guidelines ensure that the created cloud computing patterns have a consistent look and feel. The graphical layout of the pattern document format has been similarly homogenized for the cloud computing pattern language.

*Pattern Writing:* This step includes the actual creation of pattern documents. The cloud computing patterns have undergone shepherding and were the subject of a writers' workshop during the PLoP 2011 conference [FLR+11]. They were subsequently revised, and a second three-day writers' workshop was held during the ChiliPLoP 2012<sup>21</sup>. After these events, the patterns were continuously shepherded by two members of the pattern community, and were finally published in 2014 [FLR+14]. Section 4.4 summarizes all of these cloud computing patterns. For each pattern, its evolution from its original version

---

<sup>21</sup><http://hillside.net/chiliplop/2012/>

presented in [FLMS11] is covered to demonstrate how the activities prescribed by the pattern writing phase improved the resulting cloud computing patterns. Information is presented concerning how patterns have been identified, how they were named, and how these decisions may have changed between the initial version of a pattern and its current one. Section 4.5 concludes and summarizes the chapter.

*Pattern Language Revision:* As patterns are added to the pattern language over time, references among patterns may have to be revisited. Patterns that were created during the first iterations of the pattern engineering process tend to have few references to other patterns. For the cloud computing patterns, each time a new pattern was written and it referenced an existing pattern, adding a backward reference was considered. The tooling described in Chapter 6 introduces a pattern repository that may recommend such references to be added by analyzing existing ones in the pattern language.

### 4.1 Pattern Document Format

The pattern format of the cloud computing patterns has been inspired by the pattern format of Hohpe and Woolf [HW03] and guidelines of the pattern research community provided by Meszaros and Doble [MD98], and Wellhausen and Fießler [WF11]. As covered in Section 3.1, the messaging applications discussed by Hohpe and Woolf have similar architectural principles as cloud applications. The cloud reference application also uses asynchronous communication in the form of messaging. Therefore, users of cloud computing patterns are likely to access the patterns of Hohpe and Woolf [HW03], as well, at some point during the application design. Having a similar format is, therefore, motivated

by the fact that humans can access information more easily if it is presented in a homogeneous fashion [Pet95]. All cloud computing patterns comprise the same document sections with the following semantics.

*Pattern Name:* This name is used to identify each pattern. Names have been chosen to be usable as nouns in sentences. Pattern users may, thus, refer to patterns easily, and patterns can be used as subjects in sentences. For example, storing data in the form of key-value pairs in a cloud offering was called *key-value storage* (119) rather than “storing key-value pairs”. In the scope of cloud offerings, this was especially helpful to introduce a common vocabulary to discuss offerings: products, such as Amazon SimpleDB<sup>22</sup> or the table storage offered as part of Windows Azure Storage,<sup>23</sup> may now be referred to as *key-value storage*, according to the pattern they implement.

*Intent:* This is a short summary of the pattern, i.e., the problem and solution in one or two sentences. It enables the reader to quickly grasp the essence of the pattern. Similar to newspaper articles, pattern documents state all of the contained information at the very beginning, providing increasingly detailed information as the reader progresses. Initially stating all relevant information enables readers to understand the pattern quickly and progress to detailed information only if needed.

*Icon:* Each pattern has an icon of the same size as all other patterns and with an identical border. If a pattern composes multiple other patterns, this icon is used for graphical illustrations of this composition (see Section 4.3.3 for the implicit composition language). Pattern icons can be used similarly by IT architects in architectural diagrams of their applications (see Section 7.1.1 for models created by an industry partner).

---

<sup>22</sup><http://aws.amazon.com/simpledb/>

<sup>23</sup><http://azure.microsoft.com/services/storage/>

*Driving Question:* The problem answered by the pattern is stated as a question. The reader is enabled to relate to the pattern by comparing this question to the architectural problem he is trying to solve.

*Context:* This section of the pattern document describes the setting in which the problem arises. It may, especially, cover forces and challenges that make the problem difficult to solve; for example, a certain user behavior that cannot be controlled by the IT architect. The context section may also cover naive solutions to the problem and explain why they are commonly unsuccessful. In earlier versions of the cloud computing patterns [FLMS11; FLR+11], a discrete “challenges” section existed that followed the context section. During the evolution of the cloud computing patterns, this separation proved difficult to maintain, as entities forming the context of a pattern, such as a user group or a certain cloud offering, often directly led to these challenges. Therefore, these sections have been merged into a single context section so that the description of entities forming the context can be directly followed by a description of why their behavior and other considerations lead to challenges.

*Solution:* This section briefly states how the pattern solves the problem. It is kept brief and only covers the essence of the solution. This enables readers to quickly assess a pattern through its intent, driving question, and solution.

*Sketch:* The solution is supported by at least one graphical sketch. It depicts the functionality of the solution or the resulting architecture after application of the pattern. In particular, this sketch may use the icons of other patterns if they are composed by the pattern described in the current document.

*Result:* This section describes the effects achieved by following the steps prescribed by the solution. It also provides details about the



implementation of the pattern. In particular, new challenges arising during or after the implementation of a pattern may be covered, possibly with pointers to other patterns describing solutions to these problems.

*Variations:* Often, patterns can be applied in various ways, depending on slightly different or additional challenges. These differences may not be significant enough to justify description of their respective solutions in individual pattern documents. Such slight alterations of the context and solution are covered as variations of the pattern.

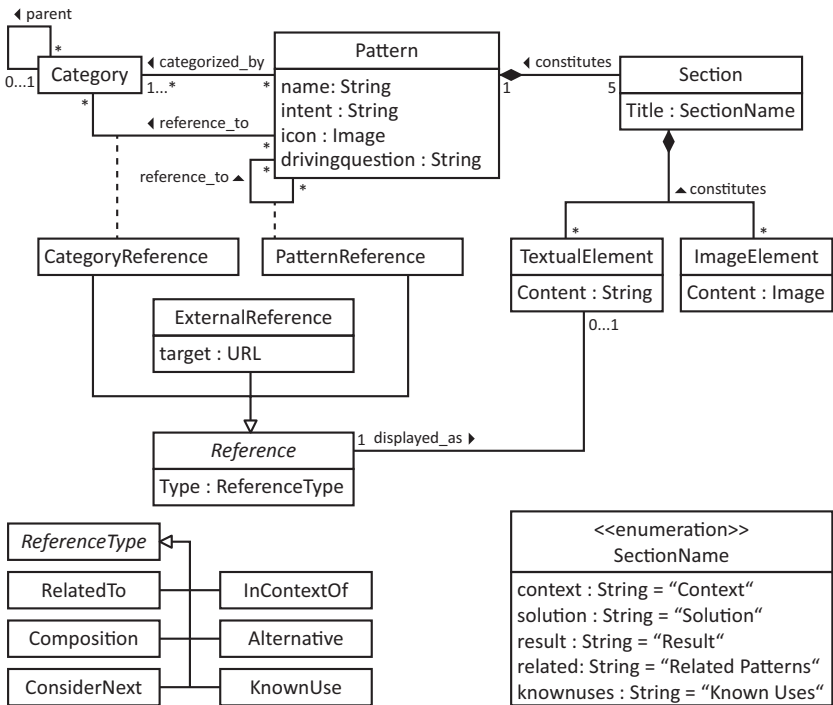
*Related Patterns:* Patterns reference each other to express, for example, that they describe alternative solutions in similar contexts or that the application of one pattern leads to problems solved by another pattern. Most of these references are covered in the related patterns section. The pattern metamodel (see Section 4.2) presents the reference types to be used by the cloud computing patterns — textually in the book and with tool support, as described in Chapter 6.

*Known Uses:* This section summarizes the known applications of the pattern. In particular, this subsumes the information sources from which the pattern has been abstracted.

## 4.2 Pattern Language Structure

The structure of the cloud computing pattern language is governed by the metamodel shown in Fig. 4.2. The central element of the pattern metamodel is the **Pattern**. Each pattern has a name as identifier and is further characterized by an intent, icon, and driving question. In addition to the Sections constituting the pattern, these elements define the pattern document format detailed in the previous section of this work. A section of a pattern may contain written text

## 4 Design of Cloud Computing Patterns



**Figure 4.2** – Metamodel of the cloud computing pattern language

as `TextualElement` and images as specified by `ImageElement`. Each `Section` has a title, as specified by the enumeration depicted in the lower right of Fig. 4.2. Patterns are categorized by one or more `Categories`, which form a category tree through a `parent` relationship. The interconnections among pattern documents, categories, or external sources are handled by `References`. In particular, `PatternReferences` point to other patterns, `CategoryReferences` point to categories, and `ExternalReferences` point to documents external to the pattern lan-

guage, such as websites, research paper, and other resources. Most references are represented by a `TextualElement` part of a `Section`. All `References` have a `ReferenceType` specified by the hierarchy seen on the lower left of Fig. 4.2. The semantics of these reference types are covered in the following section. Restrictions on the pattern language structure that could not be expressed by UML itself are captured by additional constraints covered in Section 4.2.2.

### 4.2.1 References among Patterns

The semantics of the reference types introduced by the pattern language metamodel shown in Fig. 4.2 are as follows. The toolchain described in Chapter 6 allows bidirectional querying of these references.

`RelatedTo`: This reference type is neutral: it only states that the referenced pattern has some relationship to the referencing one. The surrounding text of the `Section` in which this reference type is used describes why the reference was established. Other reference types were preferred over this one during pattern writing, as they have a well-defined semantic and, thus, can be interpreted more easily during (automated) queries in a toolchain (see Section 6.3).

`InContextOf`: References of this type to other patterns describe the setting in which the referencing pattern can be applied. Often, multiple patterns are referenced. This reference type is important to interconnect cloud offering patterns of different categories. Patterns describing the user group and the cloud runtime environment are not implemented by the pattern user, but characterize these entities and provide the user advice regarding under which conditions to choose a certain offering.

Once this is done, the pattern user may employ the `InContextOf` reference to find the patterns that are applicable to the cloud application architecture and are appropriate to this environment.

**Alternative:** This references a pattern that can be applied alternatively to the referencing one. These patterns, therefore, solve a similar problem in a different way, from which a reader may select the most fitting one.

**Composition:** The referencing pattern uses the referenced pattern in its solution to the problem. Such composite patterns, therefore, describe often-used combinations of other patterns and use the icons of the composed patterns in their sketches.

**ConsiderNext:** After reading the referencing pattern and deciding to use it, the pattern user is pointed to another pattern that may be considered next. This reference type is used to describe a common order in which cloud computing patterns should be considered; for example, to provide a reading order that is other than linear.

### 4.2.2 Constraints on the Pattern Language Metamodel

In addition to the structure of the cloud computing pattern language described by the model shown in Fig. 4.2, the following constraints specified in OCL<sup>24</sup> should be fulfilled by any instance of this model.

The references among patterns need to be described by the pattern author, i.e., for each reference, explanatory text has to be given describing why the referenced patterns are related to each other. Therefore, each `Category Reference`, `Pattern Reference` and `External Reference` must have an associated textual element by which it is displayed in order

---

<sup>24</sup><http://www.omg.org/spec/OCL/>

to be included in the explanatory text of a pattern section. References of the ConsiderNext reference type are excluded from this constraint, because ConsiderNext references are visualized outside of the pattern document without additional text (see Section 5.1.2).

*Constraint 1:* A Reference with zero associated textual elements must be of type ConsiderNext:

```
context Reference inv:  
self.textualelement.size() = 0 implies  
self.type.isTypeOf(ConsiderNext)
```

**Listing 4.1** – OCL constraint for ConsiderNext references

The cloud computing patterns use images only as part of the solution section. These sketches describe the abstract solution of the pattern.

*Constraint 2:* ImageElements must only be used in the section named “Solution”:

```
context ImageElement inv:  
self.section.name = "Solution"
```

**Listing 4.2** – OCL constraint for sketches

The pattern metamodel specifies the allowed names of sections in the form of the SectionName enumeration and that there are five sections. In addition, no section name should be used twice.

*Constraint 3:* Each section name must only be used once within the scope of a pattern:

```
context Section inv:
self.allInstances -> forAll(s1|s1 <> self implies
if self.pattern <> s1.pattern then true
else self.name <> s1.name
endif
endif
)
```

**Listing 4.3** – OCL constraint for section names

References can be added freely to the sections. However, all references to other patterns that set the context of a patterns, i.e. references of the `InContextOf` type, should be contained in the context section.

*Constraint 4:* References of type `InContextOf` must only be used in the context section of a pattern:

```
context Reference inv:
self.type.isTypeOf(InContextOf) implies
self.textualelement.section.name = "Context"
```

**Listing 4.4** – OCL constraint for `InContextOf` references

The pattern metamodel specifies `KnownUse` as a `ReferenceType`. However, this type should only be used for external references, i.e. to point to solutions implementing the pattern or to sources from which it has been abstracted.

*Constraint 5:* The `KnownUse` reference type must not be used for `Category References` or `Pattern References`:

```
context Reference inv:
self.type.isTypeOf(KnownUse) implies
self.isTypeOf(CategoryReference) = false
AND self.isTypeOf(PatternReference) = false
```

**Listing 4.5** – OCL Constraint for the `KnownUse` Reference Type

## 4.3 Graphical Design

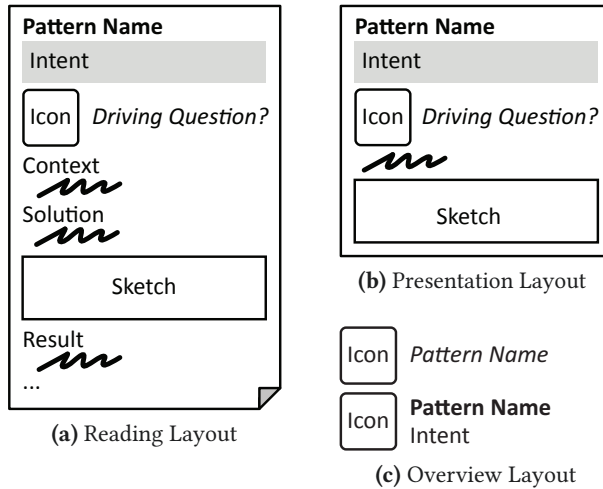
During the authoring of cloud computing patterns, a common look and feel should be ensured (see Section 2.1.1). This homogenization of the pattern document structure has been extended to the graphical elements used in patterns, again to increase perceptibility: humans can access information that is characterized by a common look and feel more easily [Pet95]. The following sections describe the homogenization of the pattern document layout and the graphical elements in greater detail.

### 4.3.1 Pattern Document Layout

In addition to comprising the same sections, the layout of the pattern format should be homogenized for different purposes. The layouts covered in the following are: reading layout, presentation layout, and overview layout. The similarity of the layouts is intentional to increase accessibility of the patterns.

*Reading Layout:* This is the layout of the pattern format used in books and on websites. The conceptual reading format is shown in Fig. 4.3a. The pattern name is used as a heading, followed by a full-width intent of the pattern that is separated by a different background color. The pattern icon is depicted below this intent box, with the driving question located next to it in italics. After the document has been started in this fashion, the remaining sections of the pattern format follow.

*Presentation Layout:* The layout of the pattern format to be used in presentations, as shown in Fig. 4.3b, starts similarly as the reading format with intent, icon, and driving question. Below this heading, the separate pattern sections are summarized. If the pattern is covered on



**Figure 4.3** – Layouts of cloud computing patterns

more than one slide, the heading is repeated on each slide. In this way, the intent, icon, and driving question remain visible at all times.



*Overview Layout:* If multiple patterns are to be included in a layout with limited space, the icon is shown with the pattern name in italics below or to the right. As an extended form, the intent may be included, as well. In this case, the pattern icon is used in combination with the pattern name in bold, and the intent on its right. This layout has also been used for the summarized patterns Section 4.4. Both overview layouts are shown in Fig. 4.3c.



### 4.3.2 Graphical Elements Used in Patterns












The icons and sketches of patterns use similar graphical elements that were homogenized to increase the accessibility of the pattern. Whereas pattern icons are designed to be used in other patterns' sketches, graphical elements covered in this section can be considered as *architectural primitives* [ZA08]. These graphical elements are, therefore, not patterns but concepts that are used by multiple patterns. In the domain of cloud computing, such concepts are, for example, an application component, a virtualized server, or a communication channel. Such graphical elements should be homogenized in the patterns of a language to ensure a consistent look and feel. Whereas Zdun and Avgeriou [ZA08] defined architectural primitives to be used in UML models, for the cloud computing patterns, the following graphical primitives have been used in a homogeneous fashion throughout all pattern documents. For the definition of composition rules covered in the following section, the list of graphical elements shown in Table 4.1 considers the types *entity*, *annotation*, *connector*, and *region*. Entities constitute modeling elements used in pattern icons and sketches. They can be connected with each other or annotated to each other using the connectors and annotations, respectively. Multiple entities may also be summarized using regions.












**Table 4.1** – Graphical elements used by cloud computing patterns

<i>Entities</i>	
	<i>Application Component</i> : Independent and isolated part of an application that offers a certain functionality.
	<i>Certificate</i> : Used to depict that encryption or signatures are involved in a communication.

## 4 Design of Cloud Computing Patterns











---










	<i>Cloud Icon</i> : Used to depict a cloud in a pattern icon.
	<i>CPU</i> : Central processing unit of a computer.
	<i>Crash</i> : Depicts a failure or erratic behavior. Unlike an <i>error</i> , a crash may not be notified and is thus harder to detect.
	<i>Data</i> : Depicts state information to denote that state is stored by an entity.
	<i>Data Elements</i> : Depict state information in a smaller granular form than the data element.
	<i>Delay</i> : Depicts a period of time during which no activity occurs.
	<i>Development</i> : Depicts the activity of creating an application or another software artifact.
	<i>Error</i> : Depicts a failure. Unlike a <i>crash</i> , errors are commonly reported and can, thus, be more easily reacted upon.
	<i>File</i> : Depicts a collection of data.
	<i>Firewall</i> : Depicts an entity that controls and possibly restricts the communication passing through it.
	<i>Folder</i> : Depicts a logical set that may contain multiple <i>files</i> .

	<i>Foreign Key:</i> Depicts a dependency of one data element on other data elements.
	<i>Hard Disk:</i> Depicts the local persistent storage device of a computer.
	<i>Load Balancer:</i> Depicts an entity that distributes communication passing through it among a set of receivers.
	<i>Management:</i> Depicts an activity or functionality handling the configuration of other entities.
	<i>Metering and Billing:</i> Depicts an activity or functionality measuring the use of an entity to charge customers.
	<i>Memory:</i> Depicts the local volatile storage of a computer.
	<i>Message:</i> Depicts a small amount of information exchanged asynchronously by communication partners.
	<i>Message Channel:</i> Depicts a connection between two communication partners to exchange <i>messages</i> .
	<i>Monitoring:</i> Depicts functionality used to monitor the use, behavior, health, or other properties of an entity.
	<i>Networking Card:</i> Depicts the hardware of a computer used to exchange information over a network.
	<i>Read Access:</i> Depicts an activity that retrieves data.

## 4 Design of Cloud Computing Patterns

---

	<i>Server</i> : Depicts a physical or virtual computer hosting application functionality.
	<i>Table Data</i> : Depicts data that is organized in tabular form. Dependencies among tables may exist.
	<i>Tree Structure</i> : Depicts data that is organized as a tree with one root element and multiple child elements.
	<i>User Group</i> : Depicts the set of users accessing an application or function.
	<i>Write Access</i> : Depicts an activity that creates or changes data.
<i>Annotations</i>	
	<i>Function Execution</i> : Denotes that the annotated entity executes an internal function.
	<i>Process</i> : Indicates the order in which annotated entities are active; for example, for sending and receiving messages.
	<i>Processing</i> : Indicates that the functionality offered by the annotated entity can be used to process the workload.
	<i>Prohibition</i> : Indicates that the annotated entity disallows access of a certain type.
	<i>User Interface</i> : Indicates that the annotated entity offers interfaces that can be accessed by humans.

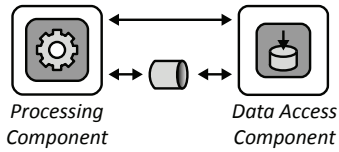
<i>Connectors</i>	
	<i>Annotation Link</i> : Annotations can be associated with annotated entities using these links.
	<i>Communication</i> : Asynchronous or synchronous exchange of information.
	<i>Inside View</i> : Internals of an entity, i.e., the component on left of the connector can be detailed in the box on the right.
	<i>Interaction</i> : Complex interaction among partners, including accesses to provided functionality.
<i>Regions</i>	
	<i>Cloud Environment</i> : Depicts the boundaries of a cloud hosting environment.
	<i>Composite Component</i> : Summarizes multiple entities or pattern icons into one application component.
	<i>Data Center</i> : Depicts the boundaries of a traditional data center.
	<i>Tier</i> : Summarizes multiple logical application components into a deployable unit.
	<i>Transaction</i> : Depicts the boundaries of a transactional interaction that guarantees ACID properties.

### 4.3.3 Composition of Graphical Elements

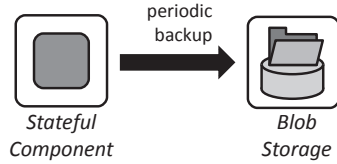
Patterns of object-oriented software architectures [BMR+96; GHJ94] use UML as a modeling language for sketches. The graphical elements of the cloud computing patterns are also composed to form such sketches, but are not restricted to a formal modeling language. Such a formal language was not used, as the number of types of graphical elements was not completely determined when the authoring of the cloud computing patterns was initiated. Therefore, a higher degree of freedom was left to the pattern authors. Now that the types of graphical elements have been defined as described in the previous section, a formal specification seems feasible. The outlook given in Chapter 8 discusses the requirements to be fulfilled in order to tackle this issue. Even though the current sketches of the cloud computing patterns are not governed by a formal composition language, the following composition rules were followed during their creation.

*Entities and Connectors:* The graphical elements representing an entity should be connected through graphical connectors. The graphical connectors are: interaction, annotation, and inside view. All other graphical elements are entities. In particular, the pattern icons may also be used as entities. Entities may be placed anywhere on the canvas, and two entities may then be connected by one or more connectors. Examples for such compositions are given in Fig. 4.4.

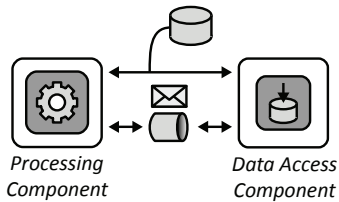
*Communication and Interaction:* A uni- or bi-directional *communication* may be expressed by the use of an arrow to indicate that two entities communicate by exchanging information. This information exchange can be synchronous or asynchronous using message channels as shown in Fig. 4.4a. If the interaction is more complex, i.e., functionality of another component is accessed multiple times in a certain order or



(a) Two patterns and connectors



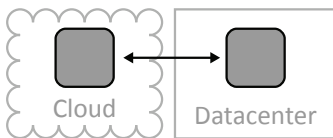
(b) Two patterns and a complex interaction



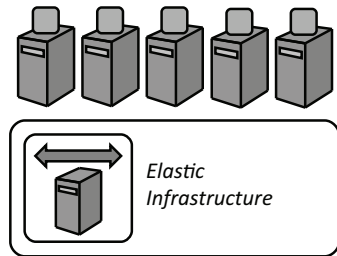
(c) Two patterns and connectors with annotations



(d) Inside view of a component with internal state



(e) Enclosing regions and application components



(f) Enclosing region and servers hosting application components

**Figure 4.4** – Exemplary compositions of graphical elements used in cloud computing patterns

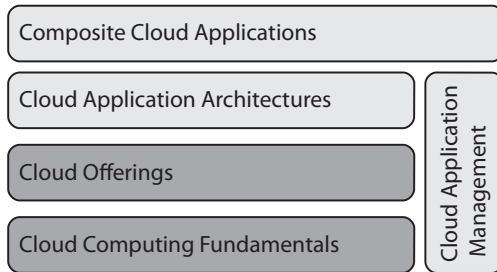
complex dependencies are to be expressed, more general *interaction* may be shown using a larger arrow than for communication. Explanatory text is added to such an arrow, as can be seen in Fig. 4.4b, to describe the complex interaction.

*Annotations:* Additional information can be given for entities using the annotation connector, as seen in Fig. 4.4c. This denotes that the annotated information is relevant to or contained in the annotated entity. In the case of *message channels*, the contained messages may also be positioned on top of the message channel without an additional annotation link. A *data entity* and *server entity* may be used in a similar fashion to express that they contain other entities.

*Inside View:* The inside view connector is used to depict even more detail about an entity than using an annotation connector. The inside of an entity is detailed in an enclosing region (refer to the next definition) to show internal components of the entity, as shown in Fig. 4.4d.

*Enclosing Regions:* The graphical elements of *architecture layer*, *cloud environment*, and *data center* are regions; thus, they may contain enclosed entities. These entities comprise the region or are hosted in it, as shown for application components in Fig. 4.4e and servers in Fig. 4.4f. Connectors may pass over region boundaries, but entities are commonly fully enclosed in a region and not positioned on its border. Architecture regions can be used to model an application stack. Thus, an architecture region placed above another architecture region is considered to use the functionality of the region below. If an architecture region contains patterns, it implements these patterns. The boundary of icons included in architecture regions may be omitted, as the boundary of the architecture region and the boundaries of pattern icons appear identical.





**Figure 4.5** – Categories of cloud computing patterns (light gray: implemented by cloud applications, dark gray: implemented by cloud environments)

## 4.4 Summary of Cloud Computing Patterns

This section summarizes all cloud computing patterns considered in this work and describes how they evolved from their initial publication [FLMS11] to the version published as a book [FLR+14]. Whenever extensions or changes of patterns are discussed, this alteration has been performed on their initial version and resulted in the version published in the book. The following summary is structured according to the categories of patterns also used by this book. Fig. 4.5 provides an overview of these pattern categories.

Patterns of the *cloud computing fundamentals* category describe cloud service models and cloud deployment types analogous to the NIST cloud definition [MG11]. They extend this definition by covering the conditions under which a certain service model and deployment type should be used for a cloud application. Patterns of the *cloud offerings* category describe the functionality offered by cloud providers to be used by an application for processing of workload, communication, and data storage. Again, the patterns cover the conditions under which an

offering should be selected, as well as the implications on the application using them. Therefore, these two categories of the cloud computing patterns are not implemented by the cloud applications, but by the cloud environment. They describe how provider offerings behave and when they should be used. An IT architect may use these patterns to characterize the cloud environment to be used by an application. Subsequently, patterns to be used in this environment can be selected, through interrelation of patterns implemented by the environment, with the following patterns implemented by cloud applications.

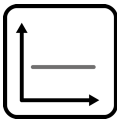
Patterns of the *cloud application architectures* category describe the general structure of the cloud application and specific application components for user interfaces, processing, and data handling. Patterns of the *cloud application management* category describe how these applications can be managed during runtime using additional management components, which rely on functionality provided by the application itself, cloud offerings, and the cloud environment. Patterns of the *composite cloud applications* category cover frequent combinations of patterns from all other categories in various use cases.

The pattern summaries given in the following sections use the overview layout introduced in Section 4.3.1. The pattern icon, name, and intent used by this layout are cited from [FLR+14]. Pattern names are followed by page numbers in parentheses for easier reference to their complete version in [FLR+14]. To increase readability and to ensure conformity to the overview layout, citations are not provided individually.

### 4.4.1 Cloud Computing Fundamentals

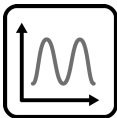
Patterns of this category describe the user group and tasks that have to be handled by the cloud application in the form of workload patterns. These are then followed by patterns for cloud service models and cloud deployment types that characterize the cloud environment according to the NIST cloud definition [MG11].

**Application Workloads:** This category of patterns characterizes the degree to which an application is utilized. Utilization can have different forms. For example, it can originate from user requests to the application, from tasks that have to be executed automatically, or from data that have to be handled by the application. In the first version of the cloud computing patterns [FLMS11], these workload patterns have not been included. However, patterns were needed to specify the behaviors of users accessing the application. Analogously, these workload patterns can also characterize the occurrences of automated tasks handled by the application. The workload patterns presented in [FLR+14], therefore, characterize the environment in which other patterns can be applied, in order to handle that workload. Throughout this work, pattern names are followed by the pattern's page number in [FLR+14] for convenient reference.



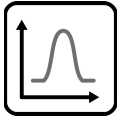
#### **Static Workload (26)**

IT resources with an equal utilization over time experience static workload.



#### **Periodic Workload (29)**

IT resources with a peaking utilization at reoccurring time intervals experience periodic workload.



### **Once-in-a-lifetime Workload (33)**

IT resources with an equal utilization over time disturbed by a strong peak occurring only once experience once-in-a-lifetime workload.



### **Unpredictable Workload (36)**

IT resources with a random and unforeseeable utilization over time experience unpredictable workload.



### **Continuously Changing Workload (40)**

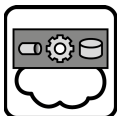
IT resources with a utilization that grows or shrinks constantly over time experience continuously changing workload.

**Cloud Service Models and Cloud Deployment Types:** Patterns of this category describe the cloud runtime environment of the cloud reference application (see Section 3.3). They have been created analogously to the NIST cloud definition: cloud service models describe the type of resources a cloud provider offers: *infrastructure*, *platform*, and *software as a service*. The cloud deployment type patterns cover the different hosting environments that clouds may use: *public cloud*, *private cloud*, *community cloud*, and *hybrid cloud*.



### **Infrastructure as a Service (IaaS) (45)**

Physical and virtual hardware IT resources are shared between customers to enable self-service, elasticity, and pay-per-use pricing.



### **Platform as a Service (PaaS) (49)**

An application hosting environment is shared between customers to enable self-service, elasticity, and pay-per-use pricing.



### **Software as a Service (SaaS) (55)**

Human-usable application software is shared between customers to enable self-service, elasticity, and pay-per-use pricing.



### **Public Cloud (62)**

IT resources are provided as a service to a very large customer group in order to enable elastic use of a static resource pool.



### **Private Cloud (66)**

IT resources are provided as a service exclusively to one customer in order to meet requirements on privacy, security, and trust.



### **Community Cloud (71)**

IT resources are provided as a service to multiple customers trusting each other in order to enable collaborative elastic use of resources.



### **Hybrid Cloud (75)**

Different clouds and static data centers are integrated to form a homogeneous hosting environment.

Similar to the workload patterns, the application developer does not implement these patterns of cloud service models and cloud deployment types. Nevertheless, they have been created to extend the rather short NIST cloud definition [MG11] and to give advice to the developer regarding under which conditions (context of the patterns) he should choose a service model or deployment type (solution of the patterns). Due to the nature of the service models, application architecture patterns are only connected to the *Infrastructure as a Service (IaaS)* (45) and *Platform as a Service (PaaS)* (49) models: using these service models, a developer

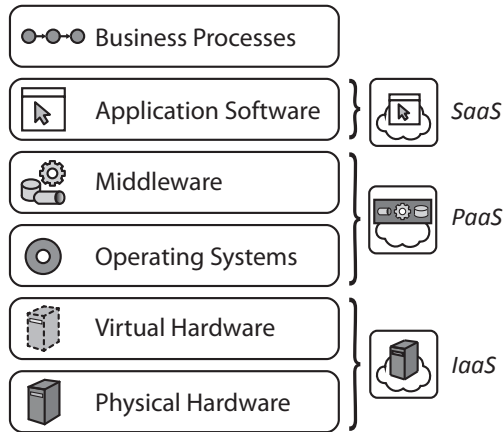
may create custom applications. Using the *SaaS* model, the customer is merely a user of a provided application. Patterns addressing how to use the *SaaS* model are, therefore, not covered, as this service model is not used to host custom applications. The first version of the cloud computing patterns [FLMS11] also contained a pattern for Composite as a Service (CaaS). This service model described how providers offer configurable application components that may be composed by users to form custom applications. This functionality is still offered by some cloud providers; for example, Amazon CloudFormation<sup>25</sup> allows the configuration of multiple virtual servers that are deployed as a unit. However, these aspects are generally part of an *IaaS* (45) or *PaaS* (49) offering. A separation of this concept from other service models seems unjustified, especially since the term “CaaS” has not been standardized in the same form as the other service models by the NIST.

The cloud service models characterize the cloud environment according to the portion of the application stack (see Section 3.3.3) that is controlled by the provider, as depicted in Fig. 4.6. *IaaS* (45) indicates that the provider controls the physical and virtual hardware on which customers may install their own operating systems. *PaaS* (49) means that the provider controls the operating systems and, possibly, installed middleware, as well. Often, the notion of servers is invisible to the customers, who only deploy their applications in this environment. *SaaS* (55) means that the provider controls the applications as well, which are only used by customers to support their business processes. A proper characterization of the portion of the application stack controlled by providers is especially important if an existing application is to be migrated to a cloud environment. See the migration patterns provided in [FLR+13] for a detailed discussion.

---

<sup>25</sup><http://aws.amazon.com/de/cloudformation/>

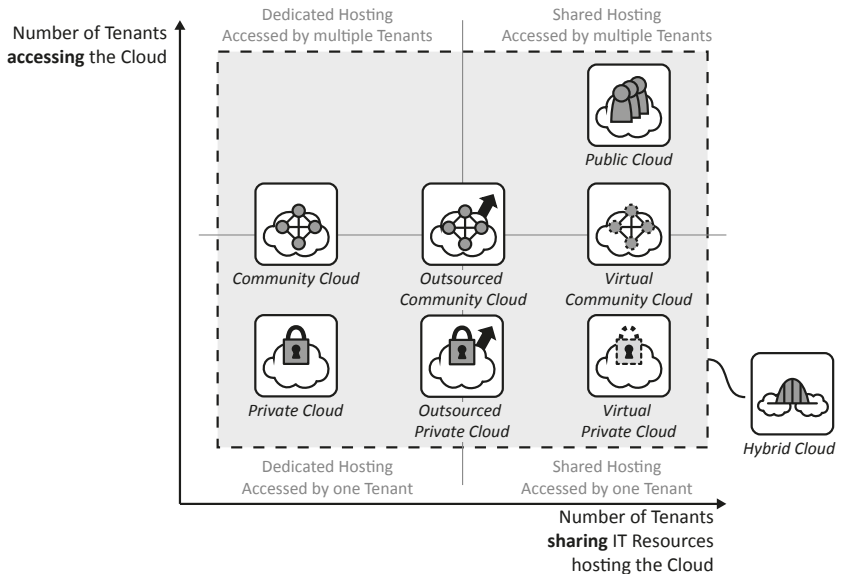
## 4.4 Summary of Cloud Computing Patterns



**Figure 4.6** – Cloud service models and the application stack (adapted from [FLR+14])

The cloud deployment types are characterized by the number of tenants that may access them and the degree of resource sharing between tenants. Tenants accessing the cloud may provision and decommission resources in the cloud. The degree of resource sharing means that the IT resources constituting the cloud environment are used to serve a varying number of tenants. These two aspects can be used to describe the cloud deployment types and their variations, as depicted in Fig. 4.7. In the first version of the cloud computing patterns [FLMS11], the entity that is hosting a cloud environment has been used as a characterizing factor: one company in the scope of a *private cloud* (66), multiple companies in the scope of a *public cloud* (62), etc. This did not match the variations during the evolution of these patterns: a *virtual private cloud* is accessed only by a few tenants – usually only one, but the underlying IT resources are shared with all customers of the public cloud.

## 4 Design of Cloud Computing Patterns



**Figure 4.7** – Cloud deployment types and variations (consolidated from [FLR+14])

Therefore, in addition to the number of tenants accessing the environment (Y-Axis), the number of tenants sharing the IT resources hosting the cloud (X-Axis) has been introduced as a characterizing factor. In the case of a *public cloud*, both characteristics are very high, as many tenants access the environment and share the hosting resources. A *community cloud* (71) is accessed by fewer tenants, as it is used by companies that collaborate on a particular topic or share a certain domain, such as healthcare. It can be hosted by one company collaborating with the other companies, resulting in the smallest degree of resource sharing. If the *community cloud* is outsourced, an external provider manages the IT resources on which the cloud is hosted, leading to a higher degree of



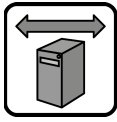
resource sharing, while the same number of collaborating companies accesses the cloud environment. IT resources of a *public cloud* can be used to host a *virtual community cloud*. In this case, a large number of tenants share the IT resources hosting the *virtual community cloud*, but access to this environment is still limited to the collaborating companies. A *private cloud* is used only by a single tenant. Different versions exist analogous to the *community cloud* by increasing the degree of IT resource sharing. Finally, a *hybrid cloud* (75) has been introduced as a combination of multiple clouds and other data centers.

### 4.4.2 Cloud Offerings

Patterns of this category describe the functional and nonfunctional behavior of offerings provided in cloud runtime environments. This category is divided into four sub-categories. Cloud environments characterize the technical behavior of clouds in general and cover commonly-used combinations of the other offerings. Processing offerings describe how workload can be executed in a cloud environment. Storage offerings describe how data can be handled using provider-supplied functionality. Communication offerings describe how a cloud application may enable communication among its distributed components as well as with the outside world.

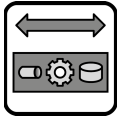
**Cloud Environments:** Patterns of this category describe the functional components of *Infrastructure as a Service (IaaS)* (45) and *Platform as a Service (PaaS)* (49) offerings. The specific functions of *Software as a Service (SaaS)* (55) offerings were not covered in [FLR+14], as the *SaaS* market is still very versatile and patterns in this category would describe how often-used applications, for example, for Enterprise Resource Planning (ERP) or Content Management Systems (CMS),

behave on an abstract level and when they should be used. As the focus of the cloud computing patterns is, however, the building of custom cloud applications, this service model is inapplicable. The cloud environment patterns describe how cloud providers commonly compose the other offering patterns to form *PaaS* and *IaaS* offerings:



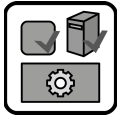
### **Elastic Infrastructure (87)**

Hosting of virtual servers, disk storage, and configuration of network connectivity is offered via a self-service interface over a network.



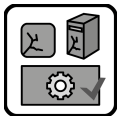
### **Elastic Platform (91)**

Middleware for the execution of applications, their communication, and data storage is offered via a self-service interface over a network.



### **Node-based Availability (95)**

A cloud provider guarantees the availability of individual nodes, such as virtual servers, middleware, or hosted application components.



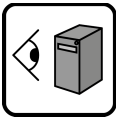
### **Environment-based Availability (98)**

A cloud provider guarantees the availability of the environment hosting individual nodes, such as virtual servers or application components.

In the first version of the cloud computing patterns [FLMS11], these patterns were included as processing patterns (the next category) and only focused on the *IaaS* (45) model. They have been extended to respect the *PaaS* (49) model, as well. The most significant change has been in the *node-based availability* (95) and *environment-based availability* (98) patterns, which had initially been named “low availability computing node” and “high availability computing node”, respectively. At first, these two patterns, therefore, made a qualitative statement about the

availability of offered IT resources (high or low), which ultimately proved inadequate, as this categorization of availability could only be made with respect to the requirements of an application. Therefore, the final versions of these patterns focus on the entity for which cloud providers assure availability. This can be done either for individual nodes or for the environment as a whole. In the former case, individual nodes such as servers or application components are said to be available a certain percentage of the time. In the latter case, individual nodes may fail, but functionality is available to provision replacements.

**Processing Offerings:** Patterns of this category describe how workload-handling functionality offered by a cloud provider behaves and under which conditions it should be used. Covered functionality ranges from the hosting of customer-specific servers and custom applications on provided middleware, to provider-supplied environments that can be directly used for execution of tasks:



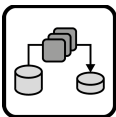
### **Hypervisor (101)**

To enable the elasticity of clouds, the time required to provision and decommission servers is reduced through hardware virtualization.



### **Execution Environment (104)**

To avoid duplicate implementation of functionality, applications are deployed to a hosting environment providing common functionality.

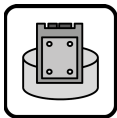


### **Map Reduce (106)**

Large data sets to be processed are divided into smaller data chunks and distributed among processing application components.

At the time of their first publication [FLMS11], these patterns subsumed only the *elastic infrastructure* pattern and the availability patterns that are now part of the cloud environment category. The other patterns have been included to describe the functional behavior of a *PaaS* (49) environment in a similar fashion. The *map reduce* (106) pattern was originally part of the application architecture pattern category. However, the provider market evolved and the functionality of this pattern — distributed execution of data query tasks — is now offered by many providers directly. Therefore, this functionality no longer has to be implemented by the application developer, but can be accessed as an offering to which only the data and queries to be executed have to be provided. The *map reduce* pattern, therefore, focuses on how such offerings behave conceptually and under which conditions they should be used, rather than how to build such an offering.

**Storage Offerings:** Patterns of this category describe how offerings to store data behave and under which conditions they should be used. They are differentiated by the style in which they store data and how it may be accessed:



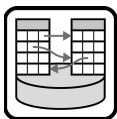
### **Block Storage (110)**

Centralized storage is integrated into servers as a local hard drive to enable access to this storage via the local file system.



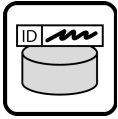
### **Blob Storage (112)**

Data is provided in form of large files that are made available in a file system-like fashion.



### **Relational Database (115)**

Data is structured according to a schema that is enforced during data manipulation and enables expressive queries of handled data.



### Key-Value Storage (107)

Semi-structured or unstructured data is stored with limited querying support but high-performance, availability, and flexibility.

While *block storage* (110) behaves similarly to hardware and is, therefore, associated with *IaaS* (45) offerings, *blob storage* (112) can often be found in both *IaaS* and *PaaS* (49) offerings. The remaining storage offerings are commonly part of *PaaS* offerings. Two adjustments have been made to these storage offerings patterns since their initial publication [FLMS11]. First, the *relational database* (115) pattern was originally named “relational data storage”, which proved unfitting, as the concept of relational databases is well established. Naming a pattern describing this concept differently was inadequate. Second, the *key-value storage* (119) pattern was originally named “NoSQL storage” to indicate that query capability is “Not only SQL” [Fow12; SF12]. It was renamed to state rather something this pattern *is* rather than to define it by what it is *not*. Also, the NoSQL databases that are currently evolving are quite different from a functional perspective. If offerings evolve in this market that behave similarly, it is likely that additional patterns need to be written. The remaining two patterns of this category describe the consistency behavior that may be displayed by all storage offerings:



### Strict Consistency (123)

Data is stored at different locations to improve response time and failure resiliency while consistency of replicas is ensured at all times.



### Eventual Consistency (126)

Performance and availability of data are increased by ensuring data consistency eventually and not at all times.

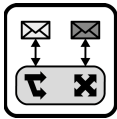
These patterns describe the considerations made by providers regarding the consistency, availability, and partitioning tolerance of the offering and the resulting impact on the application using the offering. Refer to the CAP theorem [GL02] and its discussions by Ramakrishnan [Ram12] and Brewer [Bre12] for detailed information.

**Communication Offerings:** Patterns of this category describe offerings to exchange information between application components and with the outside world. Due to the architectural similarities of cloud applications and message-based applications (see Section 3.2.2), the cloud computing patterns mostly consider messaging as a style of information exchange:



### **Virtual Networking (132)**

Networking resources are virtualized to enable customers to configure networks, firewalls, and remote access using a self-service interface.



### **Message-oriented Middleware (136)**

Asynchronous communication is made robust and flexible by hiding the complexity of addressing, routing, or data formats.



### **Exactly-once Delivery (141)**

The messaging system ensures that each message is delivered exactly once by filtering possible message duplicates automatically.



### **At-least-once Delivery (144)**

In case of failures that lead to message loss, messages are retransmitted to assure they are delivered at least once.



### **Transaction-based Delivery (146)**

Clients retrieve messages under a transactional context to ensure that messages are received by a handling component.



### **Timeout-based Delivery (149)**

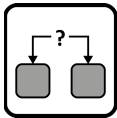
Clients acknowledge message receptions to ensure that messages are received properly.

The *virtual networking* (132) pattern considers networking hardware and is, therefore, part of the *IaaS* (45) model and the *elastic infrastructure* (87) environment. The remaining patterns consider messaging, most often part of a *PaaS* (49) offering, which is functionally described by the *elastic platform* (91) pattern. Compared with their initial publication [FLMS11], the message communication patterns focus more on a summary of the messaging patterns described by Hohpe and Woolf [HW03], rather than providing new versions of these patterns. Therefore, the pattern of reliable messaging has been omitted, as Hohpe and Woolf [HW03] already describe it. New patterns have been described regarding the delivery of messages: *transaction-based delivery* (146) and *timeout-based delivery* (149). In addition to the number of times a message may be delivered (exactly-once or at-least-once), these patterns describe that a message can be delivered within a transactional context or on a retry-basis if delivery fails. The following cloud application architecture patterns then describe, among other aspects, how to implement the client side to interact with messaging systems displaying such behavior.

### 4.4.3 Cloud Application Architectures

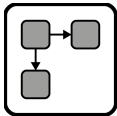
Patterns of this category cover fundamental cloud architecture principles that should be respected by all cloud applications in order to benefit from a cloud environment. Cloud application component patterns refine these principles to implement certain application functionality, such as user interfaces, processing, or data access. Patterns for cloud integration cover how information can be exchanged among application components hosted in different clouds and data centers. Multi-tenancy patterns describe how the cloud application itself may be offered as a service to multiple customers.

**Fundamental Cloud Architectures:** Patterns of this category should be implemented as a minimal set by cloud applications to benefit from a cloud environment:



#### Loose Coupling (156)

A broker encapsulates concerns of communication partner location, implementation platform, time of communication, and data format.



#### Distributed Application (160)

A cloud application divides provided functionality among multiple application components that can be scaled out independently.

These two patterns describe how two of the cloud architecture principles (see Section 3.1.3) can be implemented: *distribution* and *loose coupling*. These two architectural principles form the basis for the other architectural principles: *isolation of state* requires that application components exist that may handle the state information. *Elasticity* and *automated management* may not be required in all cloud applications, but if so, these properties are significantly simplified by loose



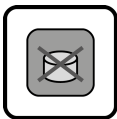
coupling. In the first version of the cloud computing patterns [FLMS11], the *distributed application* pattern was named “composite application”. However, this pattern name was changed, as it did not focus on the fact that a cloud application comprised of multiple components should be hosted in a distributed fashion, in accordance with the *distribution* cloud computing property (see Section 3.1.3).

**Cloud Application Components:** Patterns of this category describe how application components comprising a distributed application implement certain application functionality. Furthermore, these patterns consider the interaction of application functionality with provider offerings.



### **Stateful Component (168)**

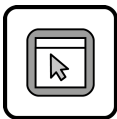
Multiple instances of a scaled-out application component synchronize their internal state to provide a unified behavior.



### **Stateless Component (171)**

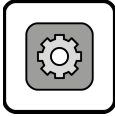
State is handled external of application components to ease scaling-out and to make the application more tolerant to component failures.

Application components can generally be divided into *stateful components* and *stateless components*, i.e., those that maintain an internal state and that do not, respectively. These patterns ensure the *isolated state* property of cloud applications. Either type of component can realize different functionality of the application, such as user interfaces, processing, or data access:



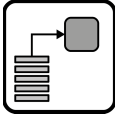
### **User Interface Component (175)**

Customizable user interfaces are accessed by humans. Application-internal interaction is realized asynchronously to ensure loose coupling.



### **Processing Component (180)**

Processing functionality is handled by elastically-scaled components. Functionality is made configurable to support different requirements.



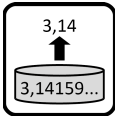
### **Batch Processing Component (185)**

Requests are delayed until environmental conditions make their processing feasible.



### **Data Access Component (188)**

Access to data is handled by components that isolate complexity, enable additional consistency, and ensure adjustability of data elements.



### **Data Abstractor (194)**

Data is altered to inherently support eventually consistent data storage through the use of abstractions and approximations.

The *processing component* and *batch processing component* patterns both handle workload. They differ regarding the time at which they process this workload. While the *processing component* handles requests immediately and relies on the elasticity of the cloud environment for scaling, the *batch processing component* may delay the workload for future processing. This distinction is made because some cloud providers flexibly change resources prices over time. Also, this delay can be used to introduce some flexibility to applications that do not run in the cloud: resources that do not handle time-critical workload can delay it to support other application components during times of high demand. Later, these components handle their own workload that has been delayed.

At some point, the application will have to access provider offerings. The following processor patterns describe the interactions with these offerings:



### **Idempotent Processor (197)**

Application functions detect duplicate messages and inconsistent data or are designed to be immune to these conditions.



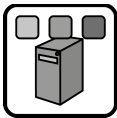
### **Transaction-based Processor (201)**

Components receive messages or read data and process the obtained information under a transactional context to ensure processing.



### **Timeout-based Message Processor (204)**

Clients acknowledge message processing to ensure that messages are processed. If a message is not acknowledged it is processed again.



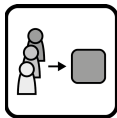
### **Multi-Component Image (206)**

Virtual servers host multiple components that may not be active at all times to reduce provisioning and decommissioning operations.

Message duplicates or inconsistent data can be handled using *idempotent processors*. In its original version [FLMS11], this pattern only considered message duplicates. It has been extended to interact with storage offerings displaying *eventual consistency* (126) in a similar manner. Transaction-based interaction may be used to interact with messaging offerings or storage offerings. Initially, the *transaction-based processor* only considered interactions with a *message-oriented middleware* (136) and has been extended to additionally cover the interaction with storage offerings. Timeout-based interaction is only available with

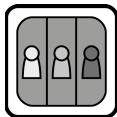
*message-oriented middleware* (136), in which messages are retransmitted if their receipt is not acknowledged; thus, there is a *timeout-based message processor*. This interaction style is currently not used by storage offerings. The cloud application architecture category was not initially used [FLMS11]. The *stateless component* (171) and *idempotent processor* (197) – both originally members of the fundamental patterns category – have been moved to this category. To indicate the affiliation of idempotent processing to these processor patterns, the original “idempotent component” pattern [FLMS11] has been renamed to the *idempotent processor*.

**Multi-Tenancy:** Patterns of this category describe how application components comprising the cloud application can be shared among multiple tenants. This is important if the cloud application itself is to be offered as a service: the larger the portion of the application stack that may be shared among tenants, the more efficient the offering of the application [FLR+14].



### **Shared Component (210)**

A component is accessed by multiple tenants to leverage economies of scale.



### **Tenant-isolated Component (214)**

A component avoids influences between tenants regarding assured performance, available storage capacity, and accessibility.



### **Dedicated Component (218)**

Components providing critical functionality are provided exclusively to tenants while still allowing other components to be shared.

A different multi-tenancy pattern should be used with respect to the required tenant isolation regarding accessibility, performance, data, and other considerations. In their first version [FLMS11; MLU09], the multi-tenancy patterns focused on the instance of application components and were named “single instance component”, “configurable instance component”, and “multiple instance component”, accordingly. An instance was always associated with a particular tenant or multiple tenants. In their current version, the multi-tenancy patterns do not focus on the instance anymore – multiple instances are considered to exist for all types of multi-tenancy components due to the elastic scaling of the application.

**Cloud Integration:** Patterns of this category are used to enable interaction between components that are hosted in different environments. This functionality should be separated from other application components, which implement application functionality, to ensure separation of concerns. The integration patterns cover aspects of access to application functions, access to data, and message exchange. If the locality of the data is important, for example, to ensure high-performance access to it, data replication is also covered.



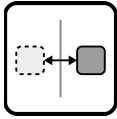
### **Restricted Data Access Component (222)**

Data provided to clients from different environments is adjusted based on access restrictions.



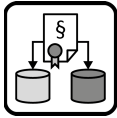
### **Message Mover (225)**

Messages are moved automatically between different cloud providers to provide unified access to application components using messaging.



### **Application Component Proxy (228)**

An application component is made available in an environment from where it cannot be accessed directly by deploying a proxy.



### **Compliant Data Replication (231)**

Data is replicated among multiple environments. Data is automatically obfuscated and deleted to meet laws and security regulations.



### **Integration Provider (234)**

Integration functionality such as messaging and shared data is hosted by a separate provider to enable integration of hosting environments.

The first version of the cloud computing patterns [FLMS11] did not include any of these integration patterns.

## **4.4.4 Cloud Application Management**

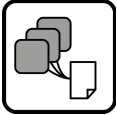
In addition to the functional implementation of application components and their composition to form a cloud application, runtime management has to be handled after deployment. This is described by patterns of this category. Management components are added to the cloud application in order to handle runtime management. They implement management processes captured as separate patterns describing their behavior.

**Management Components:** Patterns of this category describe interactions with management interfaces of providers and management of application configurations. Realizations of elasticity based on utilization of IT resources, number of synchronous requests, and number of messages is discussed. Resiliency of the application is also considered.



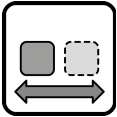
### **Provider Adapter (243)**

Provider interfaces are encapsulated to separate concerns of interactions with the provider from application functionality.



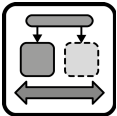
### **Managed Configuration (247)**

Application components use a centrally stored configuration to provide a unified behavior that can be adjusted simultaneously.



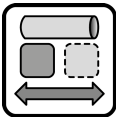
### **Elasticity Manager (250)**

The utilization of IT resources on which an application is hosted is used to adjust the number of required application component instances.



### **Elastic Load Balancer (254)**

The number of synchronous accesses is used to adjust the number of required application component instances.



### **Elastic Queue (257)**

The number of accesses via messaging is used to adjust the number of required application component instances.



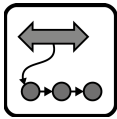
### **Watchdog (260)**

Applications cope with failures automatically by monitoring and replacing faulty application component instances.

Initially, only the elasticity patterns and the *watchdog* were published [FLMS11] and were part of the application architecture pattern category. As more management patterns were found, they have been summarized in a separate category. The *elasticity manager* was first named the “elastic component”. It was renamed to shift the focus from what is scaled

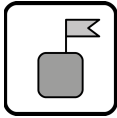
(the component) to the entity that handles the scaling (the *elasticity manager*).

**Management Processes:** Patterns of this category describe the behavior of the management components. This behavior has been kept separate from the management components, as the cloud provider may offer the functionality of management components. In this case, the management processes may be implemented as a configuration of this management functionality or may be executed by system administrators who manually access provider management functionality.



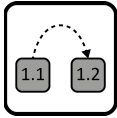
### **Elasticity Management Process (267)**

Application component instances are added and removed automatically to cope with increasing or decreasing workload.



### **Feature Flag Management Process (271)**

If the cloud cannot provide required resources in time, some application features are degraded in order to keep vital features operational.



### **Update Transition Process (275)**

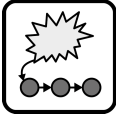
When a new application component version becomes available, running application components are updated seamlessly.



### **Standby Pooling Process (279)**

Application component instances are kept on standby to increase provisioning speed and utilize billing time-slots efficiently.





### **Resiliency Management Process (283)**

Application components are checked for failures and replaced automatically without human intervention.

These patterns are different from other cloud computing patterns: they do not use the same graphical elements and implicit composition language that the other patterns use (see Section 4.3.3). Instead, the Business Process Model and Notation (BPMN)<sup>26</sup> is used to describe the sketches that are part of their solutions. In the first version of the cloud computing patterns [FLMS11], no management processes were included.

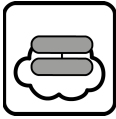
### **4.4.5 Composite Cloud Applications**

Following their initial publication [FLMS11], the cloud computing patterns summarized in the previous sections have been used in several applications. Recurring compositions of patterns in commonly occurring scenarios have again been described in a pattern format. There is a distinction between native cloud applications, which are common combinations of patterns to fulfill the cloud architectural principles, and hybrid cloud applications, which use different cloud environments.

**Native Cloud Applications:** Patterns of this category describe common pattern compositions to form applications. They are refined and extended with other patterns to suit a use case. However, their essential structure remains the same in many cloud applications.

---

<sup>26</sup><http://www.omg.org/spec/BPMN/2.0/>



### **Two-Tier Cloud Application (290)**

Presentation and business logic is bundled to one stateless tier that is easy to scale. It is separated from the data tier that is harder to scale.



### **Three-Tier Cloud Application (290)**

Presentation, business logic, and data handling are realized in separate tiers to scale them according to their individual requirements.



### **Content Distribution Network (300)**

Applications component instances and data handled by them are globally distributed to meet access performance requirements.

The *two-tier cloud application* and *three-tier cloud application* patterns describe how application components are often summarized to physical tiers; for example, virtual servers. These tiers are then deployed at a cloud provider and may be managed independently of each other. The difference between these patterns is the number of tiers: the three-tier cloud application has user interface, processing, and data handling tiers, while the two-tier cloud application summarizes the user interface and processing functionality within one tier. Three tiers are more applicable if the user group is large or if many data sources are to be accessed by the cloud application. Another concept found in many cloud applications is a content distribution network. If the user group of the application is globally distributed, multi-media content and other large files served by the application are replicated globally to ensure timely access to data.

**Hybrid Cloud Applications:** Patterns of this category describe common use cases for cloud applications especially with respect to how data centers that do not use cloud technologies may be integrated. Such

a distribution of application components among multiple environments can solve problems of existing applications; for example, by moving part of the functionality to a cloud environment during periods of high workload.



### **Hybrid User Interface (304)**

Varying workload from a user group interacting asynchronously with an application is handled in an elastic environment.



### **Hybrid Processing (308)**

Processing functionality is hosted in an elastic cloud while the remainder of an application resides in a static environment.



### **Hybrid Data (311)**

Data of varying size is hosted in an elastic cloud while the remainder of an application resides in a static environment.



### **Hybrid Backup (314)**

Data are periodically extracted from an application to be archived in an elastic cloud for disaster recovery purposes.



### **Hybrid Backend (317)**

Backend functionality (data-intensive processing and storage) is experiencing varying workloads and is hosted in an elastic cloud.



### **Hybrid Application Functions (320)**

Some application functionality provided by user interfaces, processing, and data handling is hosted in an elastic cloud.



### **Hybrid Multimedia Web Application (323)**

Website content is mainly served from a static environment. Multimedia files are served from an elastic high-performance environment.



### **Hybrid Development Environment (326)**

A production runtime environment is replicated and mocked in an elastic environment where applications are developed and tested.

Moving the user interface, processing and data handling functionality to a cloud is discussed by the patterns of *hybrid user interface* (304), *hybrid processing* (308), and *hybrid data* (311), respectively. These patterns may also be used in combination. A distribution based on the offered functionality or media content is also discussed. The *hybrid development environment* (326) pattern addresses the special use case regarding how applications may be created in the cloud during development and testing. This pattern may be used to create a cloud application, as well as an application that is not running in a cloud environment.

## **4.5 Chapter Summary**

This chapter has covered the structure of the cloud computing pattern language. This has subsumed the pattern document format and the allowed relationship types among patterns. Graphical elements used in pattern icons and sketches have been homogenized to ensure a common look and feel. Guidelines addressing the composition of these graphical elements have been provided. Finally, all of the cloud computing patterns contained in [FLR+14] have been summarized. If applicable, patterns have been compared with their first version, published in [FLMS11].

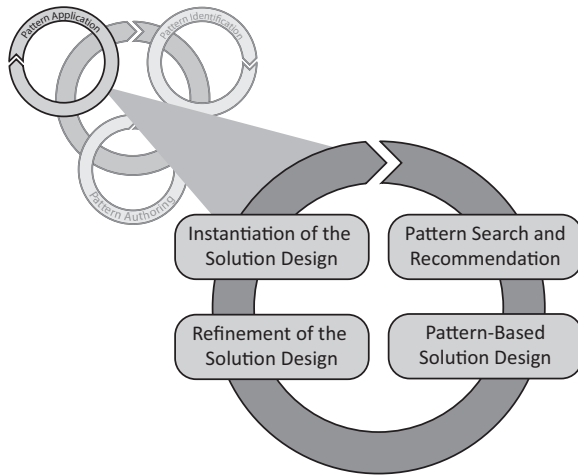
## CHAPTER 5

---

# Application of Cloud Computing Patterns

---

In this chapter, the process of searching and applying the cloud computing patterns presented in Section 4.4 is covered. This chapter addresses the conceptual organization of patterns to enable tool support: C-5: “Pattern Organization Tool”. Also, C-6: “Pattern-Based Design Method for Cloud Applications” is presented in detail, describing the steps to be followed when creating a cloud application. The covered topics can be mapped to the pattern application phase of the pattern engineering process introduced in Section 1.3. After a significant set of patterns has been authored, the steps of this phase consider making the pattern language accessible to pattern users and enable them to create an architecture based on these patterns. The steps followed in the scope of the



**Figure 5.1** – Pattern application phase for cloud computing patterns [FBBL14]

cloud computing patterns are covered in the following paragraphs. Refer to [FBBL14] for a generic version of the pattern engineering process that may be applied in other domains.

*Pattern Search and Recommendation:* The structure of the pattern language is used to enable the recommendation of patterns to readers: references among patterns are followed to navigate between patterns. Accessibility of the cloud computing patterns is detailed in Section 5.1. It relies on a categorization of the cloud computing patterns with respect to the cloud reference application (see Section 3.3) and the references of the patterns among each other, which are defined by the cloud computing pattern language structure (see Section 4.2).

*Pattern-Based Solution Design:* This step develops a method to apply the cloud computing patterns in a guided fashion. For the cloud com-

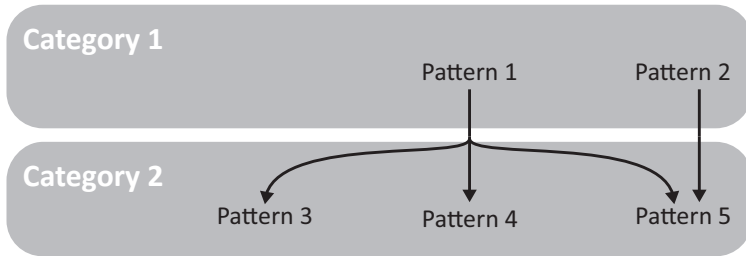
puting patterns, a design method for cloud applications is covered in Section 5.2, describing how to create new cloud applications using the cloud computing patterns.

*Refinement of the Solution Design:* This step considers the transformation of the application architecture towards its deployment. For the cloud computing patterns, this means that the set of provider offerings to be used is determined based on the abstract architecture. A methodology to map application requirements to provider offerings using patterns is detailed in [FLR+14].

*Instantiation of the Solution Design:* This step includes the development to create the cloud application code, deployment artifacts, provider offering configurations, and other features. The tool chain discussed in Chapter 6 supports this implementation step as well as the previous refinement step. Activities performed during these steps are always provider- and use case-specific; therefore, they are not discussed in general for the cloud computing patterns in this chapter. Refer to the evaluation in Section 7.1, which focuses on these aspects in the scope of two corporate settings of industry partners.

### **5.1 Accessibility of Cloud Computing Patterns**

When users need to apply patterns to their problems at hand, methods are necessary to find an initial set of applicable patterns. From there, interrelations of the patterns can be followed to navigate from pattern to pattern. In this section, these interrelations between cloud computing patterns are initially covered. Second, finding and selecting an initial set of patterns is addressed by means of pattern categorization and ordering their consideration in a use case.



**Figure 5.2** – Pattern map for two exemplary categories and five patterns

### 5.1.1 Categories of Cloud Computing Patterns

The cloud computing patterns have been categorized according to the reference cloud application (see Section 3.3). Each pattern, therefore, describes aspects of the user group, the cloud runtime environment, or the cloud application. This helps the reader to find applicable patterns related to the entity of the cloud reference application he needs to characterize or build. The user will not implement all of the cloud computing patterns: some describe entities of the reference applications that the pattern user cannot influence, such as user behavior or offering functionality. Such patterns help to characterize the context of the problem at hand. Then, through the interrelations, the IT architect using the patterns can find the applicable patterns that he may implement. The individual categories of cloud computing patterns are not covered here, but are provided as part of the pattern summaries in Section 4.4.

### 5.1.2 Order of Pattern Consideration

The `considerAfter` relationship has been used to generate pattern maps analogous to the one depicted in Fig. 5.2 for each chapter of the

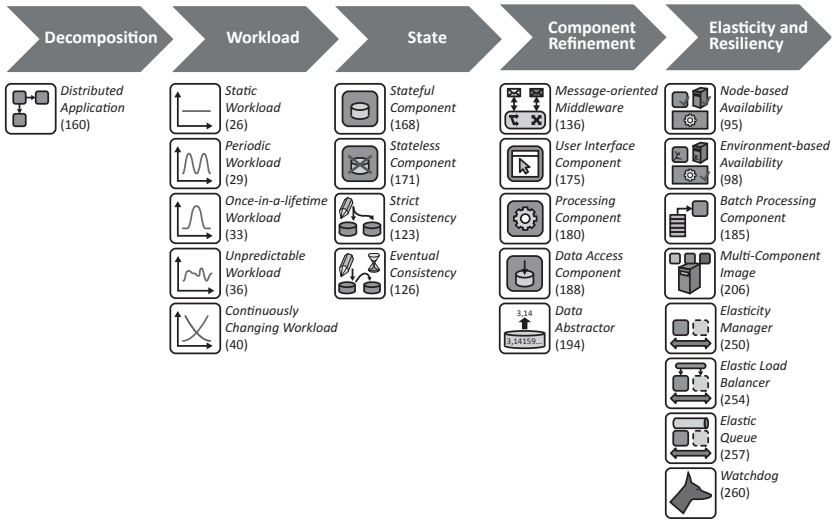


book [FLR+14] depicting all patterns covered in the respective chapter. At the beginning of each chapter, such a pattern map indicates the reading order suggested by the `considerAfter` relationship among patterns. This enables readers to identify patterns by which the chapter can be accessed: these patterns are not referenced by other patterns using the `considerAfter` reference type. If one of these patterns is then identified to be applicable to the use case at hand, the pattern map indicates which patterns to consider next. Analogously, if a pattern is inapplicable, the pattern map provides advice about which pattern of the chapter may be left out.

## 5.2 Pattern-Based Design Method for Cloud Applications

The method presented in Fig. 5.3 considers the creation of a new application, which will use a cloud as the runtime environment. In contrast to the general accessibility of the cloud computing patterns covered in the previous sections, this method describes a process to follow and the patterns to consider in each phase. Thus, it provides a design path in the pattern language to create a new cloud application or to restructure an existing one. Similar to the other means to access the cloud computing pattern language, this set of patterns should not be considered finite for this design goal. Instead, the covered patterns are fundamental for use with the respective design phase and provide entry points to the pattern language from which additional patterns can be accessed using references among them. The method itself has not been modeled as references in the pattern language to denote the order of pattern consideration in the respective phases for the following reasons. First, each phase should be described in detail and should cover the

## 5 Application of Cloud Computing Patterns



**Figure 5.3** – Phases of the pattern-based design method for cloud applications and patterns considered during each phase

reason why a pattern is considered. Second, the description of this methodology should discuss other design methods and, especially, how these methods can be integrated with the cloud-specific design method. After all, cloud computing essentially introduced additional aspects to augment the architecture of IT applications. General design methods, thus, are still applicable. The pattern-based design method for cloud applications was originally introduced in [FLR14]. It is presented here in an extended and revised form, especially since integration with other existing design methods could not be covered in the original publication due to space limitations. An overview of the phases comprising the method and the patterns considered during each phase is shown in

Fig. 5.3. The method has been applied in two industry settings as part of the validation presented in Chapter 7.

### 5.2.1 Decomposition

The architectural principles of cloud computing covered in Section 3.1 lead to a requirement for cloud applications: cloud resources are pooled among customers. Therefore, a fundamental cloud application property was identified to be the *distribution* of its application functionality among multiple application components. To enable this distribution, the application functionality is decomposed during this phase to obtain multiple application components that may be hosted independently on cloud resources.

**Involved Patterns:** *distributed application* (160).

The *distributed application* pattern covers three different strategies to decompose application functionality. First, *layer-based decomposition* groups the application functionality by purpose; for example, user interaction, business logic, or access of data storage. These functional sets form application components, which are then arranged as logical layers to govern the allowed interaction among functional groups. Components on a higher layer may only interact with components residing on the same layer or the layer below. This decomposition reduces the dependencies among components on the different layers and ensures that changes to one layer remain manageable with respect to the impact they have on other layers.

Second, *process-based decomposition* relies on a model of the business process supported by the application. For example, an insurance claim

process could be specified using modeling languages such as the Business Process Execution Language (BPEL)<sup>27</sup> or the Business Process Model and Notation (BPMN)<sup>28</sup> to capture involved activities, the control flow among them, as well as the data generated and required by the process. Each activity may be handled by humans or IT resources. In the latter case, application functionality is often provided as Web services [WCL+05], which are then enacted by the business process. The decomposition of application functionality into Web services can, thus, be governed by the business process activities they support.

Third, *pipes-and-filters-based decomposition* also groups application functionality by its purpose, similar to the layer-based decomposition approach. The interaction among components is, however, restricted to be asynchronous; for example, using messaging. Each application component (or filter) has input queues from which it retrieves asynchronous requests and output queues to which it forwards processed requests. The decomposition approach, in particular, ensures the *loose coupling* property introduced in Section 3.1.3.

**Related Design Methods:** Eeles and Cripps [EC09] present a general process to design an application architecture and cover detailed tasks for requirements engineering, documentation, definition of a logical architecture, and its refinement toward a physical architecture. They also cover software engineering methods and project management processes to coordinate developers, reviewers, and other involved parties. The design method for cloud applications can be integrated into these broader and more extensive approaches during the creation of the architecture specification. Methods to develop the application, manage

---

<sup>27</sup><http://docs.oasis-open.org/wsbpel/2.0/05/wsbpel-v2.0-05.html>

<sup>28</sup><http://www.omg.org/spec/BPMN/2.0/>

requirements, review code, and perform other functions can form the broader setting in which the design method presented here is used.

Fowler [Fow02] provides enterprise application architecture patterns and, especially, a domain model pattern that covers how the required functionality and the application data can be expressed during the design of the application. Other patterns of this pattern catalog then describe how the domain model may be refined to logical application components and to a physical system architecture. The domain model pattern and the refinement of logical application components to a physical system architecture are used by the *distributed application* (160) pattern.

Daigneau [Dai11] covers service design patterns that can be consulted during the refinement of application components if the cloud application is to comprise a service-oriented architecture (SOA). In particular, the authors cover the design of service interfaces, service implementations, and interactions among services. These patterns can be used to decompose functionality into components and and further refine them.

If the application is to be implemented using object orientation, Cheesman and Daniels [CD01] may be considered during the decomposition of application functionality into components based on UML<sup>29</sup> models.

If existing software or provider-supplied offerings are to be used to realize application functionality, these artifacts should be considered during the decomposition, as well. The design may be simplified by mapping the complete functionality of application components to existing software and vice-versa, instead of implementing application

---

<sup>29</sup><http://www.uml.org/>

components using existing software in combination with custom code. For example, if existing user management software is to be used, it may simplify the application design if this software is mapped to a single user management component. However, existing functionality should not influence the application design too significantly, as such general implementations may be suboptimal for the business case at hand. In [FLR+14], a meet-in-the-middle mapping method of application functionality and provider-supplied functionality is covered to respect existing functionality during the decomposition.

### 5.2.2 Workload

The *loose coupling* cloud application property (see Section 3.1.3) enables the cloud application to scale out and cope with resource failures, as dependencies between application components are reduced. However, the introduction of loose coupling can impact performance negatively. For example, data have to be serialized and de-serialized by interacting application components, even though they may be implemented in the same programming language, which would allow a more direct interaction. Therefore, the *distributed application* (160) pattern also covers summarizing logical application components into tiers that may optionally be more tightly integrated for deployment. Components summarized to such tiers, for example, virtual servers, should have similar requirements, so that they may be handled similarly during application management. A central aspect that affects the efficient management of application components is the workload they experience, which is considered during this phase of the cloud application design method.

**Involved Patterns:** *static workload* (26), *periodic workload* (29), *once-in-a-lifetime workload* (33), *unpredictable workload* (36), *continuously changing workload* (40).

Workload consideration targets the application user groups, which are characterized by one or more of the workload patterns. A user group generating *static workload* (26) accesses the application evenly over time and does not grow significantly or suddenly. If access behavior changes over time, any workload change that is very small can still be considered static. For example, if the application is only accessed by a few users during the week, the workload can be considered static even though no accesses occur during the weekend. *Periodic workload* (29) is generated if the access behavior of the user group differs significantly over time; for example, at the end of every month or once per year. *Once-in-a-lifetime workload* (33) is generated if the application is used significantly only during a certain event, which can often be planned in advance. *Unpredictable workload* (36) is generated by a user group that can grow or shrink very quickly without any prior notification; for example, if a mobile application suddenly becomes popular. *Continuously changing workload* (40) is generated by a user group that grows or shrinks consistently over time; for example, if the application supports a discontinued product that slowly vanishes from the market. After the behavior of each user group is described by workload patterns, the workload characteristics may be transferred to application components accessed by these user groups and further application components accessed by these components. Subsequently, the types of workload experienced by each application component can then be characterized.

In addition to the workload type, the impact of the workload on the application component is now considered. With respect to the complexity involved in processing one request, workload may impact application

components differently, even though it may be of the same type. For example, accepting a customer order in the user interface results in roughly the same processing complexity, regardless of the size of the placed order. Other application components handling the collection of ordered items and shipping may face more complex processing for larger orders. The impact of workload is, thus, characterized for each component. Those components, which experience similar workload types and workload impact, are good candidates to be summarized into tiers; thus, they are deployed and managed as holistic entities. For example, tiers could be deployed as virtual servers. Instances of these virtual servers are then added and removed from the application to manage resource demand and resiliency. Refer to the last phase of the design method for cloud applications for a detailed discussion.

**Related Design Methods:** Allspaw [All08] reviews experience from capacity planning in Yahoo! data centers and those of other companies. Such experience may be relevant during the considerations of workload in cloud applications, as well.

Menasce and Almeida [MA01] present methods of capacity planning for Web services. Therefore, if a SOA is used as the architectural style for the cloud application, the covered methods may be considered during the refinement of services to physical tiers.

### 5.2.3 State

The location where state information is stored significantly affects the design of the application as well as later management tasks, such as elasticity and resiliency handling. In this scope, the design method for cloud applications differentiates between two types of state: *session state* and *application state*. Session state represents the interaction of



users and other applications with the state-handling application. This could be, for example, the shopping cart of a customer interacting with an online shop. Application state represents the actual data handled by the application. This could be, for example, the currently processed orders of the online shop, the customer database, or other lists.

**Involved patterns:** *strict consistency* (123), *eventual consistency* (126), *stateful component* (168), *stateless component* (171).

During this phase, the IT architect specifies which application components handle state and which application components rely on other parts of the application for this purpose. Respectively, application components implement the *stateful component* (168) pattern or the *stateless component* (171) pattern. The IDEAL cloud computing properties recommend the *isolation of state* during this phase: state should be kept in a minimal number of application components to ease later management tasks, such as scaling. If no state has to be synchronized or extracted upon provisioning or decommissioning of application component instances, management tasks become simpler. At best, the state information should be kept solely in provider-supplied storage offerings. This strategy is also motivated by cloud provider guidelines [Var10a; Var08]. Prior to selecting any provider offerings for this purpose, required data consistency behavior of the application has to be considered. The patterns of *strict consistency* (123) and *eventual consistency* (126) describe these aspects in detail. Storage offerings that support *strict consistency* of data ensure that all accesses following a data update immediately see this change; thus, these offerings always provide the most recent version of data upon access operations. In large distributed environments, this behavior may be difficult to ensure. Network partitioning may occur, reducing the availability of strict consistent storage offerings, as data replicas cannot be synchronized. Refer to the CAP theorem [GL02] for a detailed discussion of the interdependence of the

storage offering properties of consistency, availability, and partitioning tolerance. Due to these limitations, many cloud storage offerings display *eventual consistency* to increase performance, improve availability, and also reduce the complexity to be handled by the provider, ultimately leading to reduced offering costs. Some storage offerings, such as Amazon DynamoDB,<sup>30</sup> even charge more for strict consistent accesses to data.

Therefore, the IT architect has to identify what state information has to be managed in a strict consistent manner. He must also identify where eventual consistent behavior is acceptable in favor of availability, performance, and operating cost. A common example where eventual consistent data is acceptable in the business case is the number of items in stock displayed by an online store. Keeping this number consistent if the store plans to reach a globally-distributed user group may be too complex, costly and performance-reducing; thus, customers are provided approximated information. The *data abstractor* (194) pattern provides more detail about such data approximations that may be acceptable with respect to a particular business case.

**Related Design Methods:** Strauch et al. [SBK+12] describe cloud data patterns to enable confidentiality in the cloud. These patterns, especially, support the migration of existing application data to the cloud [SAB13]. Fowler [Fow02] presents additional patterns for server-side state and client-side state that can be considered during the state design phase. Furthermore, if the domain model described by Fowler [Fow02] has been used to decompose the application, the structure of this model can support the selection of storage offerings after the required consistency behavior has been specified. Storage offerings can enforce more or less structure on the stored data. Offerings implementing the *relational*

---

<sup>30</sup><http://aws.amazon.com/dynamodb/>

*database* (115) pattern can be used to enforce structural consistency among data elements. For example, if customers and sales agents are to be managed, it can be ensured that each customer has an associated sales agent. Thus, the removal of a sales agent is only allowed if none of the customer data elements reference the sales agent data element to be removed.

Other storage offerings, such as a *key-value storage* (119), enforce less structural consistency on handled data elements. This complexity then has to be handled in the application using such storage offerings. However, storage offerings with fewer structure-enforcing mechanisms may display a higher performance, lower price, and other favorable characteristics, as they are easier to implement for the cloud provider. For more information regarding how cloud providers ensure consistency and, especially, why eventual consistency reduces implementation performance, refer to Tanenbaum and Steen [TS06]. Furthermore, offerings enforcing less structural consistency can handle subsequent changes to the data structure more easily.

Therefore, the use case supported by the domain model may be an indicator for a suitable cloud storage offering. A domain model with many interdependencies among described data elements may be more suitable for a relational database, as the structural consistency of these interdependencies can be enforced by the storage offering and does not have to be handled by the application. If the domain model shows few such interdependencies and, especially, if the domain model may change at a later point in time, the use of storage offerings enforcing less structural consistency of data, such as a *key-value storage* (119), may be more fitting. Refer to the *key-value storage* (119) and the *relational database* (115) patterns for more details on the context in which these offerings should be used.

Codd [Cod70] provides more information on the design of data structures that can be handled by relational databases. Silberschatz, Korth, and Sudarshan [SKS10] and Elmasri and Navathe [EN10] add more detail on this topic and also cover queries to the storage offering in greater detail. Storage offerings enforcing less structural consistency on the handled data are often summarized as NoSQL storage in the literature: Tiwari [Tiw11] provides an overview of involved concepts and current implementations. More details on such storage offerings, the underlying concepts, and differences in their implementations are provided by Sadalage and Fowler [SF12].

Transactions are atomic operations on data elements that may be kept in distributed storage offerings. This concept is relevant during the design of applications if concurrent read and write accesses are executed on the storage offerings. Tanenbaum and Steen [TS06] cover basics of such transactional data processing. More information about how to build transactional applications can be obtained from Gray and Reuter [GR93].

### **5.2.4 Component Refinement**

The components into which the application has been decomposed each handle different application functionalities. For their realization, additional patterns to be implemented by these components are identified based on the provided function. Thus, the functional purpose and behavior of the components is refined further. In addition to the application functionality subsuming user interfaces, processing, and data handling, functionality may be required to enable the interaction among the application components, thus integrating them to form a single application after their decomposition. During this integration, performance has

to be weighted against *loose coupling* to ensure resiliency, simplify management tasks, and ease later adaptation.

**Involved Patterns:** *message-oriented middleware* (136), *user interface component* (175), *processing component* (180), *data access component* (188), *data abstractor* (194).

The functional components into which the application has been decomposed are categorized by their main purpose: user interfaces, processing, or data access, described by the respective patterns. During this mapping, application components described during the decomposition phase may be split into finer granular components to separate different functionalities offered by them. In this case, the workload and data considerations for the split-up application component have to be performed again for the newly-created components. To ensure the *loose coupling* cloud application property among the application components as described by the *loose coupling* (156) pattern, their interaction has to be realized in a way that reduces the dependencies among the application components. The desired degrees of autonomy among components described by the *loose coupling* pattern are: platform autonomy, in which application components may be implemented in different programming languages running in different execution environments; reference autonomy, in which application components are unaware of the physical and logical address of other application components; time autonomy, in which application components may exchange information at different speeds and times; and format autonomy, in which application components may send and receive diversely-formatted data.

A common means to ensure these degrees of autonomy is to communicate asynchronously through an intermediary, which interfaces with different programming languages (platform autonomy), routes requests among communication partners (reference autonomy), stores

and forwards messages (time autonomy), and handles data format translation (format autonomy). This functionality is provided by a *message-oriented middleware* (136) that enables applications and application components to exchange asynchronous messages managed in queues asynchronously.

However, *loose coupling* can come at the expense of performance. In particular, data transformation to and from the message format and transport over a network may add to the overhead. Therefore, some components may again be summarized to tiers as described by the *distributed application* (160) pattern. Commonly, the benefits introduced by messaging regarding the resiliency and flexibility of later changes to the application can be treated as more important than the possibly anticipated performance impact. In many cases, messaging has proven a better fit than remote procedure call (RPC) style integration [HW03]. Cloud providers also strongly motivate the use of messaging in applications [Var10a; Var10b; Var08; MM15].

**Related Design Methods:** Hohpe and Woolf [HW03] provide a pattern language for message-based applications that should be considered for the integration of application components, as well as for the integration of the cloud application with other existing applications. The Yahoo! design pattern library<sup>31</sup> contains user interaction patterns, which describe in detail the design of application interfaces to be used by humans. If the implementation of the application components follows an object-oriented design, the patterns of Gamma, Helm, and Johnson [GHJ94] as well as those of Buschmann et al. [BMR+96] can be considered as guidelines for their design and development.

---

<sup>31</sup><http://developer.yahoo.com/ypatterns>

### 5.2.5 Elasticity and Resiliency

During the runtime of a cloud application, the experienced workload may change as characterized during the workload phase of the cloud application design method. Therefore, the IT resources required by the application to handle this workload may also vary over time. With respect to the often used pay-per-use pricing models of cloud offerings, this number of resources should be adjusted during runtime. However, such an elastic increase and decrease of IT resource numbers demands that the application may be able to cope with their addition and removal. Furthermore, the monitoring of resource demand and the adaptation of resource numbers should be automated in order to exploit pay-per-use pricing models most effectively. However, resource removal may not only occur intentionally as part of elastic scaling. Resource failures may also lead to sudden and unplanned resource unavailability with which the application should be able to cope, as well.

**Involved Patterns:** *node-based availability* (95), *environment-based availability* (98), *batch processing component* (185), *multi-component image* (206), *elasticity manager* (250), *elastic load balancer* (254), *elastic queue* (257), *watchdog* (260).

During the elasticity and resiliency phase, additional application components are added to the cloud application that do not provide application functionality, but handle the automatic management of the application during runtime. This phase covers the two management tasks that most cloud applications will have to address during runtime: the elastic scaling of used resource numbers and the resiliency with respect to resource failures.

How elasticity is handled mainly depends on the means for monitoring the workload experienced by the application. Based on this workload,

the capabilities of the application are then adjusted by provisioning or decommissioning application component instances. The application should, therefore, be scaled out instead of being scaled up. Refer to [FLR+14] for a detailed discussion of both approaches and their feasibility.

The *elasticity manager* (250) scales an application based on the monitored utilization of resources used by the application. This can be, for example, the CPU load on a virtual service, disk usage, or network traffic. The *elastic load balancer* (254) scales an application based on synchronous accesses to the application; for example, to the Web-based user interface. The *elastic queue* (257) pattern describes how the number of messages in messaging queues can be monitored in order to determine the number of required resources. Elasticity based on such asynchronous communication also is beneficial to other approaches, as the workload does not have to be handled immediately when it occurs: messages can be stored in the queue for some time until their processing is feasible. Therefore, elasticity handling based on the *elastic queue* (257) pattern can provision and decommission resources more flexibly. Resources may even be provisioned only during certain time frames when processing is most feasible; for example, overnight or on weekends. The *batch processing component* (185) pattern describes such delay of non-time-critical workloads in greater detail.

Additional considerations for elasticity management may be necessary to respect the particular properties of the environment. Many cloud providers give no assurances regarding the time that will be required to provision new resources, and in *private clouds* (66), the number of available resources may be limited. The *multi-component image* (206) pattern describes how multiple application components can be supported by a single resource provided by the cloud; for example, one virtual server. Such instances may then be used for various purposes



by the application; for example, virtual servers hosting *user interface components* (175) and *batch processing components* (185) can be reassigned to handle processing tasks when few users access the application. Similarly, the instances can be quickly reassigned to handle user input during times of peak workload. All of this occurs without the need to actually provision or decommission the resources at the cloud provider.

The *watchdog* (260) pattern covers how the availability of application components can be monitored in order to detect and replace faulty instances. Similar to the elasticity management components, monitoring can be performed on various information sources that should be consolidated: cloud providers may offer “health” information about the IT resources used by an application; for example, the reachability of virtual servers over a network. Additional application-level checks should also be realized to ensure that an application component hosted on such a resource is actually available. Means for such checks include periodic heartbeat messages of application components and periodic requests initiated by the watchdog to verify application functionality.

An important factor to consider for resiliency management is the availability assured by cloud providers. According to the patterns *node-based availability* (95) and *environment-based availability* (98), two provider assurances can be differentiated. *Node-based availability* assures the correct functioning and accessibility of a resource for a certain percentage of the time period the resource is provisioned. For example, it can be assured that a resource is available 99.95% of each month during which it is provisioned. *Environment-based availability* relaxes this assurance and only guarantees that new resources can be provisioned, i.e., self-service interfaces of the environment are functioning and accessible. Often, this assurance is combined with a guarantee that a subset of provisioned resources is available; therefore, the provider-interface may

even be unavailable, as long as some of the provisioned resources are available. For example, virtual servers at the cloud provider Amazon are assured to have an availability of 99.95%. The service level agreement of Amazon EC2<sup>32</sup> defines “available” as follows: at least one of the virtual servers provisioned in at least two redundant hosting environments – so-called availability zones – is functioning and accessible, or the environment allows the user to provision new servers. To comply with this service level agreement, the cloud application properties of *distribution*, *elasticity*, and *automated management* are required. The cloud application has to use multiple resources (distribution). These resources have to be redundant, i.e., the application is scaled out (elasticity). Also, the application has to cope with resource failures at any time by provisioning replacements (automated management).

**Related Design Methods:** Scalability of applications is not a cloud-specific topic. The cloud only makes it easy and profitable to adjust resource numbers automatically and quickly in order to exploit pay-per-use pricing models. A general introduction to scalability is provided by Abbott and Fisher [AF09] covering many real-world use cases. Menasce and Almeida [MA01] also cover means for capacity planning. In particular, it is discussed how the optimal hardware configuration of servers on which an application is deployed can be measured to determine optimal configurations. With respect to fault-tolerance, Hammer [Han07] covers patterns for this domain. Recent cloud versions of these patterns [Han14] have also been integrated with the cloud computing patterns presented in Section 4.4. Breitenbücher, Binz, and Leymann [BBL14] describe an approach to automate management processes using patterns. This approach can be based on partially automated tasks that are combined into management processes automatically [BBKL14].

---

<sup>32</sup><http://aws.amazon.com/ec2/sla/>

## 5.3 Chapter Summary

This chapter has discussed the application of the cloud computing patterns. These patterns are made accessible with respect to the entity of the cloud reference application (see Section 3.3) they help to build. An IT architect may determine a set of applicable patterns based on their mapping to the reference application. The pattern-based application design is furthermore supported by a design method. This method guides IT architects during the design process by describing the decomposing application functionality, considering the experienced workload, planning how data is managed, refining components using additional patterns, and finally, enabling elasticity and resiliency.



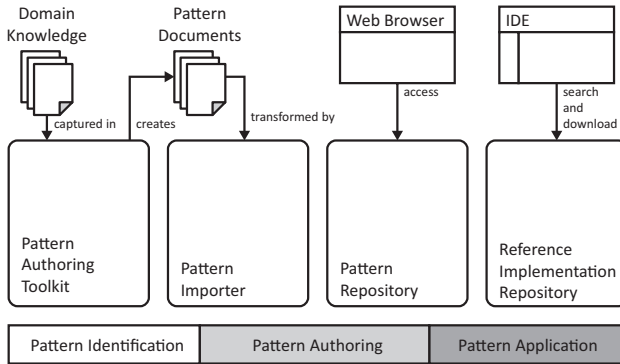
## CHAPTER 6

---

### Toolchain for Cloud Computing Patterns

---

The toolchain presented in this chapter supports the complete pattern engineering process introduced in Section 1.3. The presented tools have been designed with configurability in mind. Thus, they may be adapted to other domains than cloud computing to support pattern research in these domains, as well. For example, such adaptations have been feasible for green business process patterns [Now14], which help to reduce the environmental impact of business processes. Due to the iterative nature of the pattern engineering process and the level of uncertainty at the beginning of pattern research in a new domain, the presented tools leave a high degree of freedom during the initial phases of the pattern engineering process. For example, this ensures



**Figure 6.1** – Overview of the toolchain supporting the cloud computing patterns

that pattern researchers may easily adapt the pattern format or the graphical elements used in the domain. During subsequent iterations as these aspects of the domain become better known, the presented tools can be used to enforce a particular pattern format, reference types among patterns, and other features.

An overview of the toolchain and the supported phases of the pattern engineering process is given in Fig. 6.1. The individual tools and their subcomponents will be detailed in the following. A complete detailed view of the toolchain with all of its subcomponents is provided in Appendix C. The pattern identification phase is supported by a *pattern authoring toolkit* (Section 6.1) comprising several document templates to handle the following tasks. First, to collect, organize, and analyze information sources of the domain. Second, to draft pattern documents using a pattern document template and a stencil set for the employed graphical elements. The pattern authoring phase is further supported by a *pattern importer* (Section 6.2) that can be used to move from a

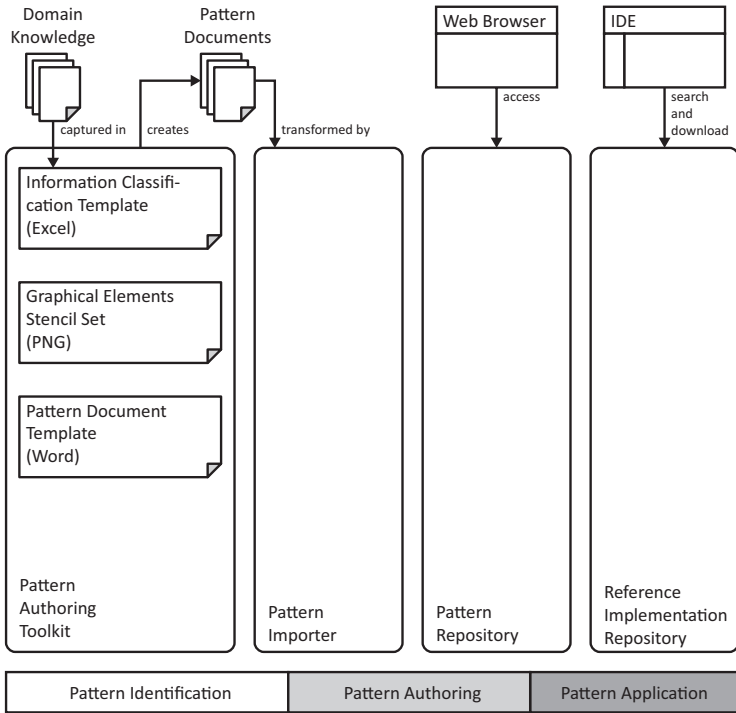
set of created pattern drafts and, possibly, existing patterns toward a pattern repository. This *pattern repository* (Section 6.3) is wiki-based, thus enabling the collaborative editing and interconnecting of pattern documents in a larger author group. It also supports the pattern user by recommendation of applicable patterns in the use case at hand. Finally, a *reference implementation repository* (Section 6.4) can be integrated with the pattern repository. This is commonly used in corporate settings, where pattern use and, thus, the created IT applications should be standardized in order to reduce the IT management effort and increase development speed. The following sections cover these four main components of the toolchain that support the pattern engineering process. The corresponding subsections each provide a detailed look at the respective components shown in Fig. 6.1.

### 6.1 Pattern Authoring Toolkit

The pattern authoring toolkit<sup>33</sup> supports the manual collection of information sources and their analysis for recurring concepts, which may then be abstracted into patterns. The work conducted using these templates is considered to be undertaken in small groups of pattern researchers, which collaborate according to the pattern authoring guidelines covered in Section 2.1, such as during shepherding iterations and writers' workshops. The pattern authoring toolkit is comprised of the following files, as shown in Fig. 6.2.

---

<sup>33</sup><http://cloudcomputingpatterns.org/authoringtoolkit.zip>



**Figure 6.2** – Detailed view of the pattern authoring toolkit

### 6.1.1 Information Classification Template

Patterns are not invented, but are rather identified in existing solutions. Therefore, the pattern authors have to support the created pattern with information about these existing solutions. Furthermore, recurring solution concepts that the pattern authors are unaware of should be identified in a set of domain documents, such as research papers, industry white papers, online resources, and other information sources. The



*information classification template* in the form of a Microsoft Excel template guides the process to collect information sources, classify them, and abstract common concepts. In the scope of the cloud computing patterns, documentation about cloud provider offerings, existing cloud applications, and architecture guidelines of cloud providers has been captured in this template according to the following steps.

1. The relevant cloud providers to be analyzed were identified. Then, documentation and architecture guidelines were collected in the information classification template.
2. The information sources were classified with respect to (i) the type of offering they provide to the cloud reference application (see Section 3.3), and (ii) the architectural challenge addressed in the documents, i.e., availability, resiliency, pay-per-use pricing models, tenant-isolation, etc. It should be noted that each document could potentially be assigned multiple classifications.
3. The content of information sources was abstracted into provider-independent summaries. These summaries are general statements about the covered architectural principles, the type of described offering, and other considerations. For example, some statements abstracted from information sources were: “Multiple instances of application components have to be used in order to increase availability.”; “Messaging should be used to exchange information among components.”; and “Storage offerings can provide access to managed files similar to a local file system.” These provider-independent abstractions were found in multiple information sources that had been classified similarly prior to the abstraction. Iteratively, the grouping of information sources was, therefore, increased with respect to their classification and abstract content.

By following these steps, information sources were collected, categorized and abstracted. Abstractions identified in multiple information sources formed the basis for drafting pattern documents, as supported by the following files of the pattern authoring toolkit.

### 6.1.2 Pattern Document Template and Stencil Set

A Microsoft Word template has been created in adherence to the pattern document sections specified by the pattern metamodel covered in Section 4.2. Thus, the pattern document template enforces this document structure on all drafted patterns supported by the created information classification and abstraction. This initial set of cloud computing patterns has been published [FLMS11; FLR+11; FEL+12] to be discussed in a larger research community, especially using shepherding and writers' workshops (see Section 2.1). Additional homogenization of the created pattern documents was ensured by using a set of graphical elements (see Section 4.3.2). This *graphical elements stencil set* is provided as multiple portable network graphics (PNG) files. In addition to the composition guidelines covered in Section 4.3.3, this stencil set ensures a common look and feel for the pattern icons and graphical solution sketches, thus increasing the perceptibility of pattern documents through format homogenization [Pet95].

After their initial draft as a technical report [FLMS11], the cloud computing patterns were shepherded and discussed during writers' workshops [FLR+11] and were presented to the cloud computing research community [FEL+12]. A dedicated three-day writers' workshop was held during the Chili PLoP conference in 2012<sup>34</sup> prior to the

---

<sup>34</sup><http://hillside.net/chiliplop/2012/>

eventual publication of the final version of the cloud computing patterns [FLR+14]. Following this publication of the patterns as a book, they were made accessible in a collaborative online tool. This enabled feedback to be obtained from a large group of pattern users and made the cloud computing patterns more easily accessible. A wiki was selected as the collaboration platform for this purpose. Prior to covering this wiki-based pattern repository in Section 6.3, the following sections address the import of existing pattern documents that were created using the pattern document template or by other means.

## 6.2 Pattern Importer

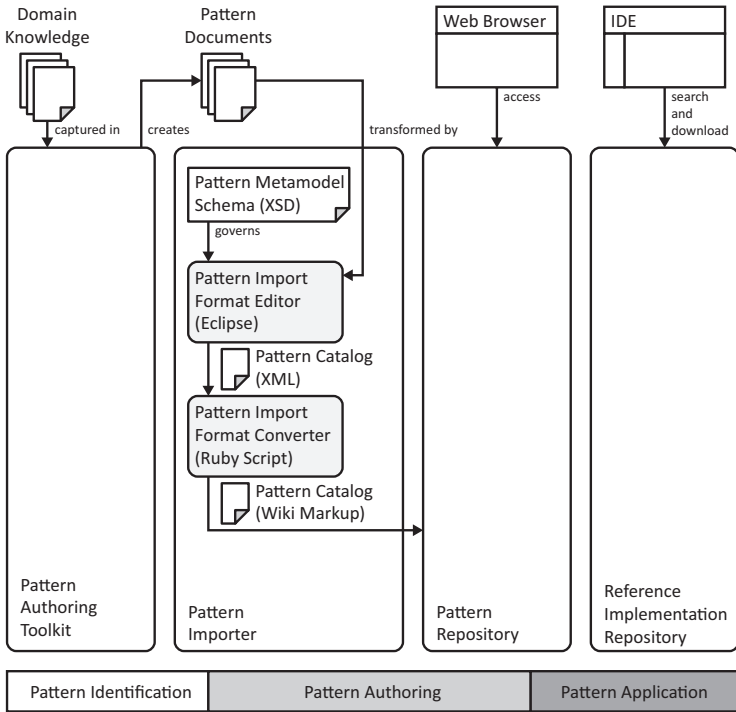
Patterns created using the pattern authoring toolkit are in a Microsoft Word document format. Other patterns of cloud computing or other domains may be available in other formats, such as research papers, books, or websites. In order to import such diverse formats into a wiki-based pattern repository, an import format has been defined in accordance with the pattern metamodel (see Section 4.2), which may then be converted into a format that can be imported into the wiki. The following components comprising the pattern import are shown in Fig. 6.3

### 6.2.1 Pattern Import Format Editor

The import format was specified in accordance to the pattern metamodel (Section 4.2) to ensure the following four properties.

1. The pattern import format should be human-readable, because manual work may be involved in its creation. Only some existing

## 6 Toolchain for Cloud Computing Patterns



**Figure 6.3** – Detailed view of the pattern importer

patterns are available in a machine-processable form; for example, hypertext markup language (HTML). In this case, certain elements of the existing format, for example, headings, could be identified and automatically transformed into the corresponding sections of the pattern metamodel. When patterns were extracted from research papers, books, and other sources not intended for automated processing, some manual work was required, especially if licensing of the original content required

an abbreviation or alteration prior to making the content available in a Web-accessible pattern repository. This was the case for the cloud computing patterns, where only 10% of the book's content [FLR+14] was allowed to be made accessible online.

2. The pattern import format should be generated easily from existing sources that were available in a machine-readable form.
3. The conformity of created import data to the pattern metamodel should be ensured as early as possible, i.e., during the manual generation, and should not be delayed to the import into the pattern repository.
4. Existing editors and format validation should be preferred over the creation of custom validation functionality. This ensures that the resulting toolchain can be more easily adapted to a changing pattern metamodel in other domains, as existing tools have to be reconfigured instead of re-implementing or adjusting custom tools.

The Extensible Markup Language was chosen for the pattern import format as it fulfills the abovementioned requirements. An XML-Schema has been created [FL14] that ensures conformity to most aspects of the pattern metamodel (Section 4.2). Using this schema, any XML Editor can be used to create the pattern import format. For the cloud computing patterns, the open source integrated development environment (IDE) Eclipse<sup>35</sup> has been used. The additional OCL constraints covered in Section 4.2.2 could not be ensured using the standard capabilities of XML editors. Instead, these constraints are ensured using additional validations handled during the conversion of the pattern import format

---

<sup>35</sup><http://www.eclipse.org>

into the data format used by the wiki-based pattern repository. This conversion is covered in the following section.

### 6.2.2 Pattern Import Format Converter

The XML-based pattern import file is converted into wiki-markup,<sup>36</sup> which is supported by the used wiki software, MediaWiki<sup>37</sup>. The pattern metamodel elements were mapped to MediaWiki elements. Pattern documents are persisted as wiki articles. Sections of the pattern documents correspond to sections in these articles. Categories of pattern documents are also supported by MediaWiki as categorizations of wiki articles. References among pattern documents are persisted using the semantic properties of the wiki article linking to the corresponding wiki article. To enable such semantic properties, the MediaWiki extension Semantic MediaWiki<sup>38</sup> was used.

The conversion is implemented as a Ruby<sup>39</sup> script, which generates the wiki-markup based on the pattern import file. Additionally, this input format converter also generates layout configuration files based on the pattern sections and the employed categories. This is a generalization in support of different pattern languages. If patterns in other domains do not use icons, driving questions, or other parts of the pattern document format presented in Section 4.1, the corresponding layout element is not generated by the converter. If more sections are used or the sections of the pattern document are named differently, the output of the converter is also adjusted accordingly. The same is the case if the reference types

---

<sup>36</sup>[http://www.mediawiki.org/wiki/Markup\\_spec](http://www.mediawiki.org/wiki/Markup_spec)

<sup>37</sup><http://www.mediawiki.org/>

<sup>38</sup><http://semantic-mediawiki.org/>

<sup>39</sup><http://www.ruby-lang.org/>

among pattern documents are named differently. The converter uses these references to create semantic links among wiki articles. These link types can then be used to query the pattern repository; for example, to find all alternatives to a certain pattern. The collaborative wiki-based pattern repository, its internal data structure, and its querying capabilities based on semantic links among contained pattern articles are covered in the following section.

### 6.3 Pattern Repository

Following the manual creation of pattern documents, a collaborative online tool makes patterns easily accessible and editable in a collaborative fashion. The resulting pattern repository is depicted in Fig. 6.4. To enable online editing and accessibility in a large pattern research group, a wiki was selected as the collaboration platform. Wikis have been employed extensively for the purpose of organizing patterns [LC01; HHA11; Now14]. In this work, MediaWiki was chosen as it supports Wikipedia,<sup>40</sup> most likely the largest wiki project and as it provides advanced semantic extensions: Semantic MediaWiki<sup>41</sup>. This extension enables typed references among pattern documents in the form of semantic properties that can be queried. The following sections cover the data structure used by this wiki to support the pattern metamodel. Furthermore, the query capability of the semantic extensions to increase accessibility of the cloud computing patterns is described.

---

<sup>40</sup><http://www.wikipedia.org/>

<sup>41</sup><http://semantic-mediawiki.org/>

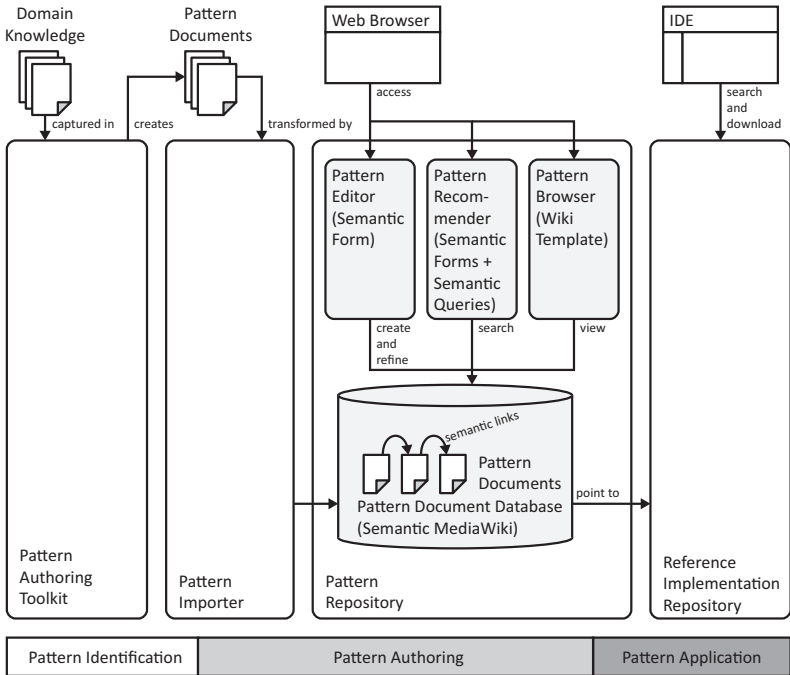


Figure 6.4 – Detailed view of the pattern repository

### 6.3.1 Pattern Document Database

Each pattern document is imported into the pattern repository as a wiki article. The structure of this wiki article was in conformance to the pattern metamodel as ensured by the import format converter. Conformity during later editing of such wiki articles is covered in Section 6.3.4. References among pattern documents took the form of typed links enabled by the Semantic MediaWiki extension, which implements the resource



description framework (RDF) standard<sup>42</sup>. Therefore, typed links are called “semantic properties” as in other semantic tools [AH11]. They enable the definition of properties for each wiki article, which may either have a data value or point to other wiki articles. For example, a wiki article about Amazon Web Services (AWS) may have a semantic property “Web address”, which points to the value “<http://aws.amazon.com>”, and another property “launched”, which points to the value “2006”. In the scope of pattern documents, these properties have been used to introduce typed references among patterns; for example, to express alternatives or an order of consideration (see Section 4.2 covering the reference types of the pattern metamodel). These semantic properties can be queried for each pattern document contained in the pattern repository; for example, to find alternatives for a pattern or to find all patterns that should be considered after a certain pattern. These queries are used to visualize references among patterns in the pattern browser and for pattern recommendation functionality, both covered in the following.

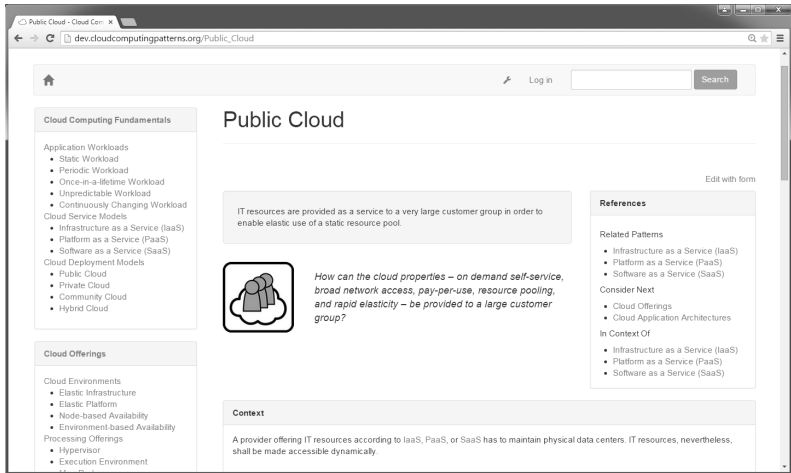
### 6.3.2 Pattern Browser

The pattern documents are visualized as wiki articles. A screenshot of the *public cloud* (62) pattern in the pattern browser is visible in the screenshot in Fig. 6.5. The layout specification is handled by a wiki article template that is generated by the pattern import format converter. It follows the *reading layout* of the cloud computing patterns introduced in Section 4.3.1. Additionally, alongside each pattern article, a references box is shown that summarizes all relations of the displayed pattern to other pattern documents. Internally, this references box is automatically

---

<sup>42</sup><http://www.w3.org/RDF/>

## 6 Toolchain for Cloud Computing Patterns



**Figure 6.5** – Screenshot of the pattern browser

generated by querying the semantic links among pattern documents. If links of a certain type can be found, the section corresponding to this links type in the references box lists the connected patterns. For example, to obtain the list of all patterns that should be considered after the *public cloud* (62) pattern, the following query can be used.

```
[[ #ask: [[ConsiderNext::Public Cloud]] ]]
```

**Listing 6.1** – Query to obtain patterns to be considered after the public cloud pattern

Therefore, the references box is filled by querying the typed links of the currently-displayed pattern document. Also, these querying capabilities have been used by the following pattern recommendation functionality.

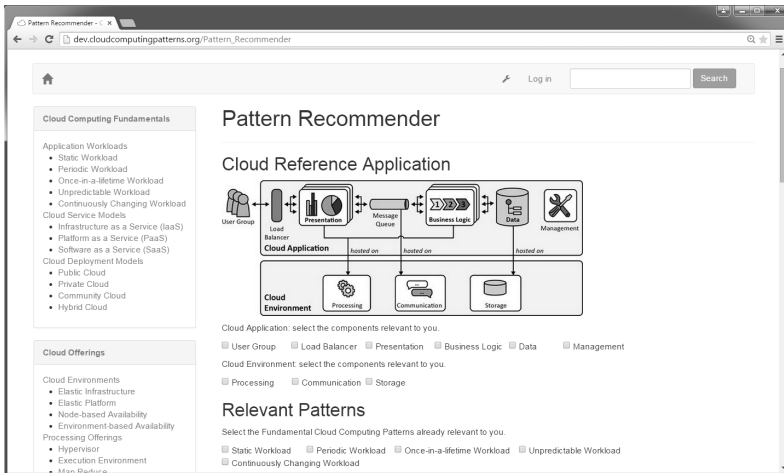


Figure 6.6 – Screenshot of the pattern recommender

### 6.3.3 Pattern Recommender

The accessibility of the cloud computing patterns is covered in Section 5.1 and is supported by the pattern recommendation functionality using the cloud reference application as the entry point. Similar recommendation tools for patterns have also been used in the domain of green business process management [Now14] and in the domain of data migration [SAB+13a]. A semantic form is used to acquire input from the pattern user about which component of the reference application is relevant to him, as shown in Fig. 6.6. Additionally, he may specify a set of patterns that have already been identified as relevant in the use case at hand. Given this specification, a set of relevant patterns is recommended to the user through their correlation to the cloud reference application and their relation to the user-specified patterns. By

default, only the `ConsiderAfter` link is evaluated, but the user may also specify other reference types that should be considered to find relevant patterns. Additional patterns that are referenced from the set of relevant patterns are queried: First, all patterns with incoming links from the set of relevant patterns are queried. Second, the set of links connecting these patterns with the relevant pattern set are queried. Finally, the list of user-specified patterns is ordered with respect to the number of links they have to the set of relevant patterns and presented to the user in descending order.

### 6.3.4 Pattern Editor

The pattern repository should also provide functionality to create new patterns after the initial import. Also, imported patterns should be editable to be refined and adjusted. The pattern format specified by the pattern metamodel is enforced within the pattern repository by editing pattern documents using the semantic form shown in Fig. 6.7. While this functionality is similar to using document templates of the pattern authoring toolkit in order to edit pattern documents, the references among pattern documents were used to simplify the pattern revision. Part of the pattern language revision phase of the pattern engineering process covered in Section 1.3 is the identification of references among patterns that are not bidirectional. If a pattern is, for example, an alternative to another pattern, the same is the case in the opposite direction. However, simply treating all references of a certain type among patterns as bidirectional is inefficient, as supporting explanatory text related to why a reference existed between two patterns had only been written in a single pattern document. Also, some reference types, such as `ConsiderNext`, are not bidirectional. Therefore, the pattern editor only suggests patterns that reference the currently-edited pattern

## 6.4 Reference Implementation Repository

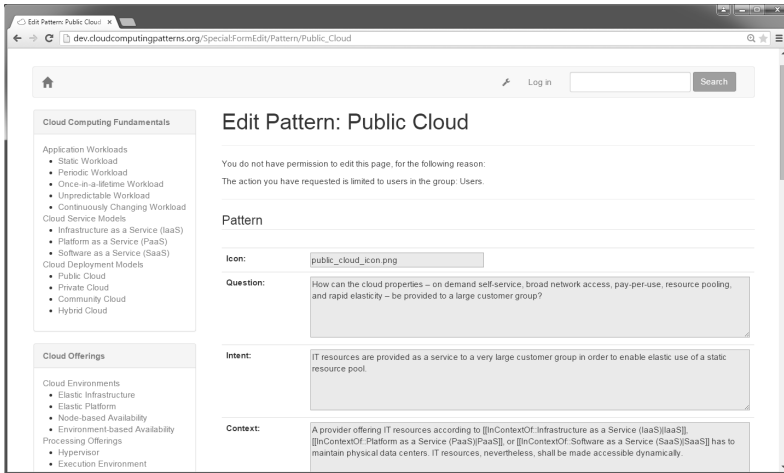


Figure 6.7 – Screenshot of the pattern editor

so that the pattern author could possibly add them to the currently-edited pattern.

## 6.4 Reference Implementation Repository

The pattern engineering process describes the use of reference implementations as part of the pattern application phase (detailed in the introduction to Chapter 5). This standardization of created implementations given a set of patterns is often used in corporate settings, where each pattern implementation uses a common IT infrastructure stack. Standardization of the IT stack is a common goal, as managing various combinations of operating systems, middleware, and other hard- and software forming custom IT stacks is a significant cost driver for IT

departments [DKPM07]. Also, development time may be reduced if developers can rely on existing code templates. Setup and management of the homogenized IT stacks can also be handled by IT departments, reducing the effort for the developer. The toolchain supporting the cloud computing patterns, therefore, incorporates standard tools to manage such reference implementations of cloud computing patterns that can be referenced from pattern documents in order to support their implementation and standardize the solutions created by pattern users. The resulting reference implementation repository is depicted in Fig. 6.8. Reference implementations are customized to each corporate setting: refer to Section 7.1.1 for an industry use case of the tools presented here.

In general, the toolchain supporting the cloud computing pattern implementation relies on standard tools for code management and development as much as possible, to simplify the integration with existing development tools and existing development processes. Therefore, the tools comprising the reference implementation repository are existing open source software. Maven<sup>43</sup> was selected as the central code repository to contain and organize the reference implementations. A front-end and management system for this repository, Artifactory,<sup>44</sup> was used to interface with existing integrated development environments (IDE) of developers.

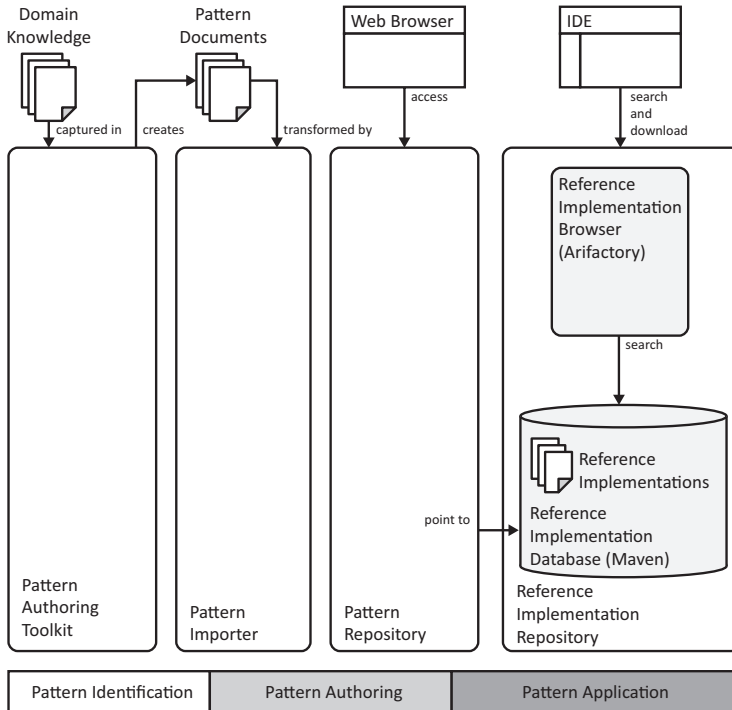
### 6.5 Chapter Summary

The presented toolchain supports all phases of the pattern engineering process introduced in Section 1.3. The pattern authoring toolkit sup-

---

<sup>43</sup><http://maven.apache.org/>

<sup>44</sup><http://www.jfrog.com/artifactory/>



**Figure 6.8** – Detailed view of the reference implementation repository

ports collecting relevant information sources for the identification of patterns, their classification, and abstraction into common concepts. The wiki-based pattern repository can be used for existing patterns using an importer that employs an XML import format, which can be generated or created manually. The pattern repository enables browsing and editing patterns. Patterns may also be recommended to pattern users based on the entity of the cloud reference application they describe and a provided set of previously-identified patterns. Reference

implementations can be used in a corporate setting to standardize the implementations of patterns in order to make created applications more manageable and reduce development time.



## CHAPTER 7

---

### Validation

---

The user group for cloud computing patterns is comprised of IT architects, developers, students, as well as lecturers (university and continuing education). As the usability and applicability of the cloud computing patterns manifests in the experience of these user groups, a questionnaire could have been used in the present work to validate the cloud computing patterns. However, the user group to be analyzed using this questionnaire remains rather small, because the cloud computing patterns are still new and have not been adopted by a very large number of individuals. Therefore, a questionnaire would not have been able to provide significant evidence. For such small groups, all members of the group would have to be questioned: Kasunic [Kas05] suggests this

complete coverage of investigated groups for group sizes smaller than 800 persons. Therefore, two other validation activities are employed: (i) action research [Wie14] in industry settings and (ii) an investigation of use of the cloud computing patterns in research and industry that is uninfluenced by any of the authors of [FLR+14].

### **7.1 Use of Cloud Computing Patterns by Industry Partners**

Action research [Wie14] can be applied as a validation technique in domains where statistical validation using a test group and a control group is infeasible. It is often used in education, where a new teaching method is communicated to teachers and then followed for a well-defined duration, i.e., for a specified number of lectures. Subsequently, feedback is collected from the teachers and reflected upon to identify deficiencies in the method. The use cases with industry partners have been conducted in this fashion: industry partners have been given an introduction to the cloud computing patterns and have had access to the book [FLR+14]. Afterwards, the industry partners developed solutions to their architectural problems at hand. Periodic meetings were held to collect information about the applicability of the cloud computing patterns.

#### **7.1.1 Daimler TSS GmbH**

Daimler TSS<sup>45</sup> considered the cloud computing patterns and the design method for cloud applications to create a vehicle telematics plat-

---

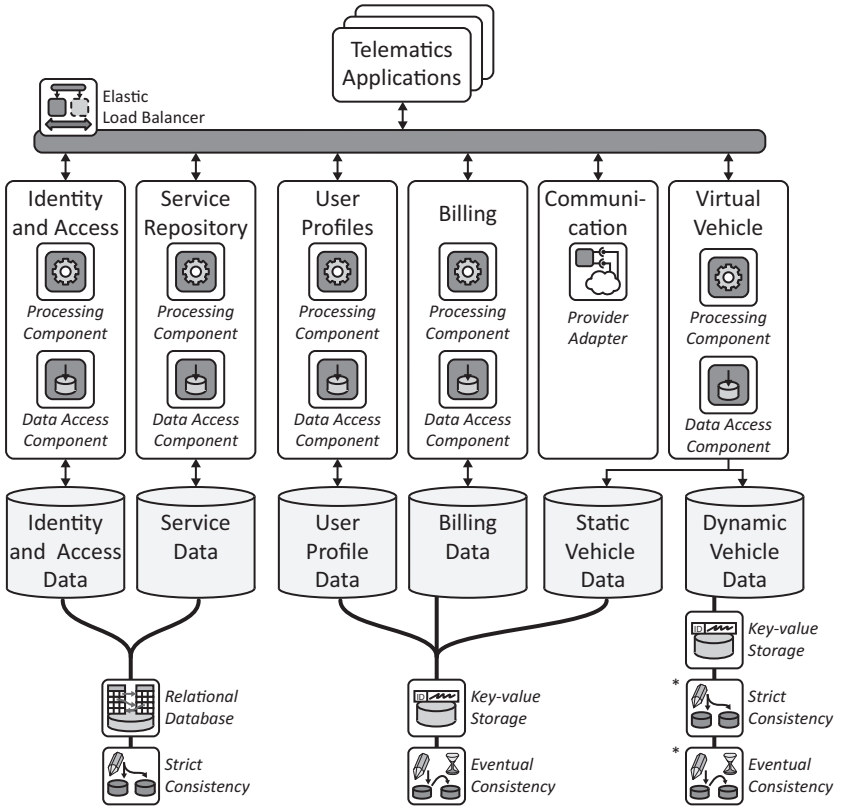
<sup>45</sup><http://www.daimler-tss.de/>

form [Hil14]. This platform serves as a runtime environment for applications that access telematics services; for example, to remotely view the status of a car, control some of its functions, or track its location. Use cases are, for example, applications that help users to manage service appointments in car shops, control their heating systems remotely, or restrict the movement of their car with respect to speed and location after handing it over to a valet parking service. The number of use cases is still undetermined; therefore, a platform should be created that makes vehicle telematics services accessible and serves as a runtime environment for rapidly-developed application prototypes.

The pattern-based design method for cloud applications (see Section 5.2) was followed as part of a larger requirements identification and architecture design process specified by Eeles and Cripps [EC09]. Figure 7.1 shows the architecture of the platform that was developed.

During the decomposition phase, the following platform services required by hosted applications were identified. The *identity and access service* handles user authentication and authorization. The *service repository* manages the set of available services, which can be those services of the platform or services offered by applications running on the platform. The *virtual vehicle service* provides an interface to cars, making them accessible through the platform. The *user profile service* contains additional information about each platform user that is not managed by the identity and access service. The *billing service* handles metering of platform use and respective billing to customers. The communication service offers interaction channels to users, such as e-mail and text message functionality.

The platform experiences *unpredictable workload* (36), as the behavior of users and other applications interfacing with applications running



\* Required level of consistency can be specified with each request.

**Figure 7.1** – Architecture of the telematics platform (adapted from [Hil14])

on the platform was unforeseeable during the workload phase of the design process.

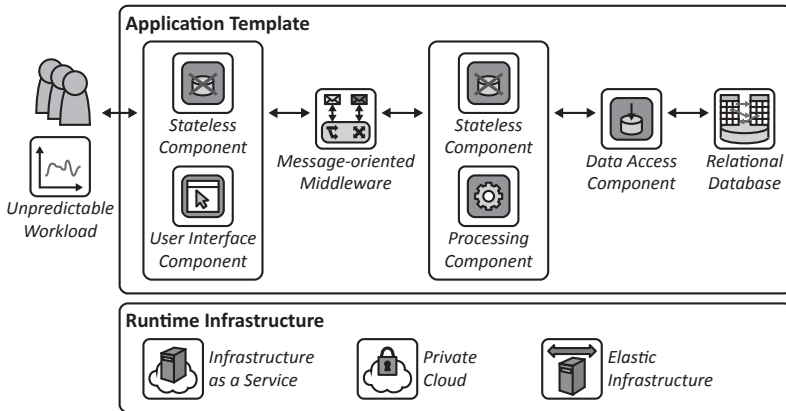
During the state design phase, the data handled by the decomposed services was characterized regarding the necessary consistency behavior and the type of storage offering used, as seen in the annotations of the patterns *key-value storage* (119), *relational database* (115), *strict consistency* (123), and *eventual consistency* (126) to the data elements accessed by the respective services. The data of the virtual vehicle service was separated to be handled in different storage offerings displaying different consistency behaviors. Static vehicle data, which changes seldom or never, such as production date or service intervals, are handled by an eventual consistent *key-value storage*. Dynamic vehicle data, which changes more often, such as door lock status or GPS location, are stored in a separate *key-value storage*. For this dynamic data, different consistency behavior is supported for each data entry: the door lock status, for example, is realized in a strictly consistent manner, while the GPS location may be obsolete due to signal unavailability; thus, it is provided eventually consistent.

During the component refinement phase, the *two-tier cloud application* (290) pattern was identified to be generally implemented by the services. Since most services only handle data access and non-compute-intensive processing, a separation of the user interface and processing functionality into separate tiers, as described by the *three-tier cloud application* (294) pattern, was not used. Most of the services, therefore, implement the *processing component* (180) and *data access component* (188) pattern. Exceptions are the communication service used to integrate external services for e-mail and text message communication into the platform, thus using the *provider adapter* (243) pattern.

During the elasticity and resiliency phase, the *elastic load balancer* (254) pattern was selected to distribute the workload among service instances. The *watchdog* (260) pattern, which is used to enable resiliency, is not shown in Fig. 7.1 due to space limitations. It is implemented by the *elastic platform* (91), on which the services are hosted to ensure availability. The *elastic platform* is also not depicted in Fig. 7.1 due to space limitations.

Custom applications may now be developed based on this platform. The development should be sped up by the creation of an application template and its guided configuration and refinement for each given use case. Therefore, a composition of the cloud computing patterns to form a template was defined, and a customizable reference implementation was created to support any future developments of applications. Fig. 7.2 depicts the resulting architecture of the application template. It was created by following the same design process governing the creation of the platform itself.

As the workload type depends on the hosted application and its users, the application template should be able to handle *unpredictable workload* (36). In order to cope with different workload types, the functionality of the application template was decomposed into application components as described by the *three-tier cloud application* (294) pattern. As an adjustment, the message-based interaction among the processing tier and data handling tier was omitted to simplify the template structure. Its reintroduction was kept as a guideline for very large applications. The tiers of the template were designed as *stateless components* (171), as suggested by the *three-tier cloud application* pattern. Data is handled in a *relational database* (115), but other storage offerings were planned to be supported in the future. The *three-tier cloud application* pattern also governs elasticity behavior, which the template foresees to be based on synchronous accesses to the user interface and



**Figure 7.2** – Architecture of the telematics application template (adapted from [Cha14])

messages for the processing tier. As an *elastic infrastructure* (87) is used for hosting the application, it is scaled by adjusting the number of virtual servers. Resiliency is realized using the functionality of this infrastructure, which monitors the availability of the virtual servers.

A reference implementation for the template was created based on Java<sup>46</sup> and was managed using Apache Maven,<sup>47</sup> as described by the tooling architecture in Section 6.4. Variability of this reference implementation was captured using the CAFE Variability Model [Mie10]. Custom adjustments to Maven enabled the interpretation of these models to configure the application template for a developed application, as well as its automatic deployment [Sch13] [Cha14].

<sup>46</sup><http://www.java.com/>

<sup>47</sup><http://maven.apache.org/>

### 7.1.2 Dr. Ing. h.c. F. Porsche AG

Dr. Ing. h.c. F. Porsche AG<sup>48</sup> used the cloud computing patterns for reengineering an information integration system, which provides generic access to multiple information sources. This system is used primarily during the development of new cars for the management and analysis of data gathered during the testing of new cars using physics simulations, prototyping, or physical tests. In contrast to other business intelligence systems and data warehouses maintaining historical data, the aim of this information integration system is to provide access and generic analysis capabilities to live data managed in various databases, each maintained by various departments of the Dr. Ing. h.c. F. Porsche AG. This system has been growing historically and should be analyzed and optimized to increase performance for its growing user group. The design process for cloud applications was followed in an adapted form for this existing application.

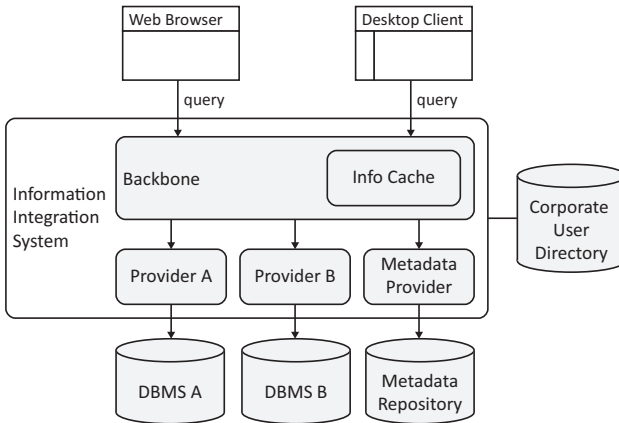
Instead of decomposing application functionality, as a first step, the existing application components comprising the information integration system were identified in the existing system to describe a componentized system architecture. The resulting abstract architecture is shown in Fig. 7.3. The information integration system supports a desktop client and a Web client. Both can be used to generate queries to integrated data based on a custom query language. This query language is based on a common data model and simplifies queries across various integrated data sources. It also supports the specification of user access rights to control which data elements may be visible and adjusted by users. Internally, the system is comprised of a *backbone* that handles the interpretation and execution of queries across multiple integrated databases. Each database is integrated using a *provider component* that

---

<sup>48</sup><http://www.porsche.com/>



## 7.1 Use of Cloud Computing Patterns by Industry Partners



**Figure 7.3** – Architecture of the information integration system

accesses the integrated database management system (DBMS). One specific database is the *metadata repository*, which is also integrated using a provider. It contains information about all other integrated databases. Data maintained by the metadata repository is cached in a so-called *info cache* in the backbone to increase performance. An external *corporate user directory* is employed to manage the users of the integration system.

In order to understand the workload behavior of users and its impact on application components, the workload experienced by the information integration system was recorded over an extended period of time. As most users work in the one geographic location, *periodic workload* (29) could be identified with low utilization during lunch hours, weekends, and other off-peak hours, and with peak utilization toward the end of each week and month, when certain report documents had to be created. After this recording, a testing instance of the information

integration system was deployed, to which the recorded workload could be replayed. This enabled a detailed analysis of the time required for processing certain parts of the queries.

Simultaneously, a list of optimization strategies was identified. The most successful strategies considered the management of state information within the information integration system. This optimization effort could, therefore, be mapped to the state phase of the cloud application design method. Both remaining steps, component refinement as well as elasticity and resiliency, have been omitted by this application of the method by the Dr. Ing. h.c. F. Porsche AG.

The state design phase first considered the patterns of *strict consistency* (123) and *eventual consistency* (126). With a growing user group of the information integration system, the formerly feasible strictly consistent management of all integrated data introduced a significant performance decrease. In particular, if a query had to be divided into a large number of subqueries to the integrated databases, user rights had to be computed within the scope of each sub-query. For long-running requests, *strict consistency* ensured that user rights removed during the query had an immediate impact on query results. As a means to increase performance, user rights are now only computed once for each initial query and are to be directly used by all sub-queries. This small adjustment was found to lead to a significant performance increase, while still ensuring strictly consistent behavior of the queried databases. Only the user access rights experience *eventual consistency*.

Second, the *data abstractor* (194) was considered to identify certain query results that may be abstracted or approximated in order to increase query performance. Again, a rather small adjustment to the application functionality could be identified that provided a significant performance increase. Originally, users were given the information

relating how many data objects returned by a query they were allowed to access. Providing this information required the evaluation of user rights for all of the returned data elements. Therefore, the accessible data objects, as well as the overall number of returned data elements, are now approximated in order to increase query performance. Nevertheless, users may specify that they require unapproximated data, if needed.

## 7.2 Uninfluenced Use of Cloud Computing Patterns

A literature survey and online search have been performed to identify scientific research publications, online industry resources, and presentations at industry conferences that cite any of the cloud computing patterns. Sources were considered that fulfilled the following criteria: (i) citation of one of the scientific publications concerning the cloud computing patterns [FLMS11; FLR+11; FEL+12; FLR+14; FLRS12; FLR+13; FLR14] or links to the website <http://www.cloudcomputingpatterns.org>; (ii) authors of the book on cloud computing patterns [FLR+14] did not co-author the source describing the use of the cloud computing patterns; and (iii) the source described an actual use of the cloud computing patterns or the related architectural principles. In particular, sources were omitted that only mentioned the cloud computing patterns briefly as motivation, i.e., in the introduction of a research paper or only in the bibliography, without incorporating these references in the written text of the publication.

### 7.2.1 Use in Research

Three use categories could be identified in scientific publications that are covered in the following: (i) the architectural properties and cloud computing patterns were used, (ii) implementation of one of the cloud computing patterns was described, (iii) formalisms were developed in order to use the cloud computing patterns for provider classification and selection or to facilitate the modeling of cloud computing applications using modeling languages such as UML.

*Use of the Architectural Properties and Cloud Computing Patterns:* Hammer [Han14] provides some cloud versions of his patterns on fault-tolerant software [Han07]. These cloud versions of the original patterns make extensive reference to the cloud computing patterns, thus extending the pattern language of cloud computing. Gangwar and Rana [GR14] provide a general introduction to cloud computing and compare its properties with those of grid computing. The authors used the offering types described by the cloud reference application (see Section 3.3) as a means to characterize cloud offerings. Gambi and Pautasso [GP13] identify the cloud architectural principles as important requirements for cloud applications to use cloud resources efficiently. The authors transfer these properties to a cloud-based business process management system that uses REST interfaces. Falatah and Batarfi [FB14] outline criteria to consider when using cloud resources and moving existing applications to the cloud. The IDEAL cloud computing properties have been included in this catalog to realize them in applications, in order to benefit from a cloud environment. The two master's theses of Vainikka [Vai14] and Araújo [Ara13] used the comparison of cloud application properties to SOA properties [FLR+13] and the classification of cloud offerings, in addition to the NIST cloud definition (Section 3.1.1).

*Implementations of the Cloud Computing Patterns:* The following sources referred to the cloud computing patterns to obtain abstract specifications for their implementations. Bien and Thu [BT14] provide a refinement of the multi-tenancy patterns *shared component* (210), *tenant-isolated component* (214), and *dedicated component* (218) to support a hierarchical user base: users are organized in hierarchical groups for which access rights can be expressed. These rules are then enforced by the refined multi-tenancy patterns. The results have been captured in a pattern format based on [GHJ94] and have been implemented in a *private cloud* (66). Jamshidi and Pahl [JP14] implemented the *batch processing component* (185) pattern in a *hybrid cloud* (75) environment. The presented architecture of a video processing application based on Google App Engine<sup>49</sup>, Amazon AWS<sup>50</sup>, and Windows Azure<sup>51</sup> also seems to resemble the *hybrid backend* (317) and *hybrid application functions* (320) patterns, even though this is not mentioned explicitly by the authors. Kourtesis, Alvarez-Rodríguez, and Paraskakis [KAP14] discuss and the implementation of the *watchdog* (260) pattern. In particular, the authors provide a semantic model of the information sources based on which a *watchdog* may identify component failure, which significantly extends the abstract description of monitoring information given by the *watchdog* pattern itself.

*Formal Modeling Based on Cloud Computing Patterns:* Sousa, Rudametkin, and Duchien [SRD14] envisioned a formal modeling language based on cloud computing patterns to describe an application regarding general hardware and software requirements, such as scalability, redundancy, and resource location. Suzuki et al. [SPH+12] describe a method and tools for model-driven engineering of cloud

---

<sup>49</sup><http://appengine.google.com/>

<sup>50</sup><http://aws.amazon.com/>

<sup>51</sup><http://azure.microsoft.com/>

applications based on the cloud computing patterns and messaging patterns of Hohpe and Woolf [HW03]. Fleck et al. [FTLW14] and Bergmayr et al. [BTN+14] use UML to formalize modeling with the cloud computing patterns. Templates in the form of UML<sup>52</sup> models are provided for cloud computing patterns to simplify their reuse during model-driven development for different cloud providers. Di Martino [DiM14] generally identifies that cloud computing patterns become part of the natural language of IT architects and developers to refer to common concepts. Furthermore, Di Martino, Cretella, and Esposito [DCE13] have formalized the document and pattern language structure of the cloud computing patterns, as well as other patterns from the same and related domains, using semantic models. These models describe existing cloud providers and their offerings to guide selection after relevant patterns have been identified. This approach has been studied in detail with respect to the Windows Azure<sup>53</sup> cloud in Di Martino et al. [DCES14].

### 7.2.2 Use in Industry

An online search has been conducted to find references to the cloud computing patterns in user groups, forums, blog entries, and especially in industry presentations for which slideshare<sup>54</sup> and similar platforms were the main source. The aim of this search was to determine the use of the cloud computing patterns in industry settings, which could be classified as follows: (i) use of the pattern names as language elements to refer to concepts described by the patterns and (ii) introduction

---

<sup>52</sup><http://www.uml.org/>

<sup>53</sup><http://azure.microsoft.com/>

<sup>54</sup><http://www.slideshare.net/>

of abstract concepts using patterns followed by technology-specific details.

*Pattern Names as Language Elements:* Ian van Reenen, the CTO of CentraStage, refers to the *loose coupling* (156) pattern to describe the concept of how the application run by his company could scale with the increasing workload:<sup>55</sup>

*“In the last thirty days the device count on CentraStage Online has grown by more than the total devices added in our first 4 years of business! We built it to scale so it’s all good news and happy days and that scale is what allows us to build new and better features, and provide some great offers like our current 500 free OnDemand devices. At the heart of our ability to scale lies the concept of loose coupling in an elastic cloud. There is a brief description of the concept here [link to loose coupling pattern] and [link to other source].”*

In the Google group of a London-based DevOps Community, a question is posed to find a tool that manages configurations and inventory databases, both of which have to be accessed by different parts of the application. The *managed configuration* (247) pattern is used to describe the abstract functionality of the desired tool:<sup>56</sup>

*“I want to raise a question to the community about tooling for configuration management or inventory database. [...] The first problem is that we can assemble an inventory from different sources: [...]. The second problem is that we need to consume an inventory in different tools [...]. So we were*

---

<sup>55</sup>[https://community.centrastage.com/centrastage/topics/centrastage\\_customer\\_service\\_update\\_friday\\_19\\_10\\_2012](https://community.centrastage.com/centrastage/topics/centrastage_customer_service_update_friday_19_10_2012)

<sup>56</sup><https://groups.google.com/forum/#!topic/london-devops/RfMs0Kr1hr4>

*thinking to have a “inventory or configuration manager” which we will feed from these different sources, and export to different targets. Something like this pattern: [link to managed configuration pattern].”*

*Introduction of Abstract Concepts:* The cloud computing patterns are used in various presentations to communicate the abstract concepts they capture as a means of education. Jeff Chu, Microsoft MVP, uses the cloud computing patterns as part of an introduction to development with Windows Azure.<sup>57</sup> Florian Goerg, IBM Solution Architect, uses the patterns to introduce abstract concepts, which are then mapped to IBM open cloud components.<sup>58</sup> Romeo Kienzler, Data Scientist and Architect at IBM, used the cloud computing patterns in a similar manner to describe the abstract concepts for big data processing, which were then mapped to concrete implementations.<sup>59</sup>

### 7.3 Chapter Summary

The cloud computing patterns have been successfully applied to use cases of two industry partners. The pattern-based design method for cloud applications has been applied completely or in an adapted form to improve an existing application. Where the use of the patterns was not actively influenced by an author of the cloud computing patterns, patterns have been used by other researchers and in the industry. In research, the cloud computing patterns have been used to characterize

---

<sup>57</sup><http://slideshare.net/regionbbs/designing-cloud-application-architecture-with-windows-azure-platform>

<sup>58</sup><http://docslide.net/internet/the-ibm-open-cloud-architecture-and-platform.html>

<sup>59</sup><http://de.slideshare.net/ormium/the-datascients-workplae-of-thefuture>



clouds and introduce cloud architectural properties. Some implementations of the cloud computing patterns have been found that explicitly gave reference to the implemented cloud computing pattern. Other work has started to use the cloud computing patterns for modeling of cloud applications. In industry, the use of cloud computing patterns as common language elements has been identified in online blogs and user groups. Also, the cloud computing patterns have been used in presentations to introduce abstract concepts, which were then technically refined.



## CHAPTER 8

---

### Conclusions and Outlook

---

Based on the pattern engineering process introduced in Chapter 1 and refined in the introductions of Chapters 3 to 5, the identification, authoring, and application of cloud computing patterns were covered. This process was generalized [FBBL14] to be applicable in other domains where pattern research is to be conducted. After covering related work on pattern identification and authoring in Chapter 2, the cloud computing domain was structured for pattern research in Chapter 3, and an overview of the architectural properties of clouds and cloud applications was provided. Furthermore, a cloud reference application was introduced. The structure of the cloud computing patterns themselves was covered next by defining the pattern language metamodel

in Chapter 4. All cloud computing patterns were also summarized in this chapter. For the application of the cloud computing patterns, a pattern-based cloud application development method was introduced in Chapter 5. It can be applied to create new applications that use the cloud as runtime environment. The complete pattern engineering process is supported by a toolchain covered in Chapter 6. The validation in Chapter 7 has shown that the cloud computing patterns were applied successfully in two industry use cases and were used in both research publications and industry.

### 8.1 Answers to Research Questions

The research questions raised in Chapter 1 have been answered as follows.

*Q-1: “Architectural Baseline”: How can the architectural properties of cloud applications be identified?*

The pattern engineering process has structured the domain of cloud computing for pattern identification. One means to structure the domain has been the analysis of cloud properties (Section 3.1) in order to deduce the architectural properties of cloud applications enabling them to benefit from a cloud hosting environment. The contributed *C-1: “IDEAL Cloud Application Properties”* have been compared with other architectural styles (Section 3.2) related to cloud computing in order to identify similarities and differences.

*Q-2: “Pattern Coverage”: How can it be ensured that a set of patterns enables building a cloud application?*

The pattern engineering process has led to the definition of a cloud reference application (Section 3.3) in order to abstractly describe components

and functionality to be identified in provider-supplied documentation and guidelines as well as existing applications to identify patterns. As a consequence, the contributed *C-2: “Cloud Computing Patterns”* cover all of the components and functionality prescribed by the cloud reference application; thus, the architecture of a cloud application can be described using these patterns.

*Q-3: “Homogeneous Representation”: How can a common textual and graphical representation of patterns increase perceptibility?*

The document format of the cloud computing patterns (Section 4.1) has been homogenized and formalized using a metamodel (Section 4.2) described in UML with additional OCL constraints. The use of graphical elements in the icons and sketches of the cloud computing patterns have also been homogenized (Section 4.3) by providing a graphical stencil set and composition rules. These means ensure that the contributed *C-3: “Format of Pattern Documents and Graphical Elements”* are similar throughout the cloud computing pattern documents, increasing their perceptibility [Pet95].

*Q-4: “Pattern Organization”: How can patterns be organized and presented so that users find relevant patterns quickly?*

The metamodel formalizing the pattern format also prescribes how pattern documents can be interrelated to form a pattern language. Through these interrelations, an IT architect may navigate the pattern documents by following references among pattern documents. Therefore, the contributed *C-4: “Pattern Language Metamodel”* ensures that patterns do not have to be accessed in a linear form. It also forms the basis for the contributed *C-5: “Pattern Organization Tool”* (Chapter 6), which manages the pattern documents and their interrelations. Furthermore, it enables the querying of the pattern languages based on the typed references among patterns.

*Q-5: “Pattern-Based Design”: How can the selection of patterns be guided to create the architecture of a cloud application?*

While the metamodel of the cloud computing pattern language has enabled the navigation among patterns, some means are required to identify the initially applicable patterns. The categorization of the cloud computing patterns (Section 4.4) has provided some orientation for selecting patterns: first the environment is characterized, then offerings are selected. Through references from patterns of these categories, related cloud application architecture patterns can be selected, followed by patterns for their management. The contributed C-6: “*Pattern-Based Design Method for Cloud Applications*” (Section 5.2) offers an additional means to access the cloud computing patterns. It describes the phases of creating a new cloud application and lists patterns to be considered during each phase. These patterns provide an initial set of patterns to be selected for an architecture, which may then be refined further by IT architects.

### **8.2 Limitations of Cloud Computing Patterns**

The use of cloud computing patterns in two industry use cases (Section 7.1) has shown two limitations.

First, the cloud computing patterns do not address how to build a cloud environment and the contained offerings. This restriction was made during the identification of the cloud computing patterns, as cloud providers offered minimal information about the internal workings of the clouds. However, as many companies desire to use *private clouds* (66), patterns for the creation of the cloud itself are required.

Second, the abstraction of the cloud computing patterns and patterns in general, which makes them reusable for different technologies, also poses a limitation for the approach. The application of the abstract concept prescribed by a pattern to concrete technologies is a challenging task for which the required technical detail is not provided by the pattern document. Therefore, IT architects often have to specify additional guidelines and templates to be used by developers during the implementation of patterns.

### **8.3 Research Opportunities**

Possible extensions to the results presented in this work could be as follows. The abstract concepts of the cloud computing patterns themselves could be improved, leading to new solutions not captured by the patterns. The cloud computing patterns could be integrated with other existing patterns, and new patterns could be created. The navigation of the cloud computing pattern language would likely have to be improved after such an integration, for the purpose of guiding IT architects. Finally, cloud computing patterns could be interpreted as architectural decisions and could also be used in modeling tools for application architectures.

#### **8.3.1 Research-Driven Improvement of Patterns**

Patterns always capture knowledge gained from experience. Therefore, the cloud computing patterns abstract from cloud offerings, provider documentation and guidelines, as well as existing cloud applications. The patterns have introduced an abstraction from the best practices

currently in use. These covered abstract concepts may now be reviewed from a research perspective to develop new and superior solutions.

For example, the cloud computing patterns describe currently-applied means to enable elasticity in cloud applications. The concept of elasticity may now be researched further, to create superior means to those used in practice today; for example, as considered by Truong and Dustdar [TD15]. The cloud computing patterns could bridge the gap between industry and research, as they describe current concepts applied in industry to which research may now provide improvements. As these improvements are adapted by cloud applications, they extend the cloud computing patterns, as well.

Another extension of the cloud computing patterns that could be researched is their application in a different environment than originally intended. For example, the applicability of the cloud computing patterns in a non-cloud environment could be investigated to evaluate possible benefits of their use. Nowak et al. [NLS+11; NL13b] used the cloud computing patterns to reduce the ecological footprint of process-based applications. They describe respective variants of the cloud computing patterns to be applied for this purpose.

### **8.3.2 A Pattern Language for IT Applications**

The cloud computing patterns already reference some other patterns from the IT domain. The analysis of related cloud computing patterns in Chapter 2 already provided a mapping of the cloud computing patterns presented in this work to these other patterns. However, there is still a significant number of patterns related to computer science that have not been integrated. For example, the integration with data patterns for confidentiality [SBK+12] or data patterns for scientific



workflows [RSM14] could be used to relate to more patterns specifically targeting the domain of data handling.

In particular, as these other existing patterns could likely be added to the presented pattern repository (Section 6.3) in order to be connected with each other, a pattern language for IT applications would be formed. Such an effort would likely require the definition of new reference types in the pattern language metamodel (Section 4.2) to be used for references between pattern languages.

Also, new methods and tools are likely necessary to handle the large set of interrelated pattern documents, as the manual following of references among patterns becomes too time-consuming. New ways to handle pattern recommendation likely have to be investigated. The connected patterns already form a directed graph with typed edges. An addition to this graph could be values on the edges based on some rationale; for example, the impact on a property of the resulting application (cost, performance, etc.). Such an addition could enable better guidance of pattern users during the selection of patterns fitting their design goals, as well as an optimization of pattern selection based on specified goals.

When many patterns from the IT domain are integrated in such a manner, concepts not covered by these pattern languages may also become visible. In these areas, pattern research may then be undertaken according to the pattern engineering process (Section 1.3).

### 8.3.3 Cloud Computing Patterns as Architectural Decisions

Selection of a pattern represents a decision made by an IT architect. Each selection of a pattern can, therefore, be interpreted as an architectural decision [HAZ07]. The rationale behind each architectural decision could be captured using architectural decision methods and tools. Zimmermann [Zim09] has presented such an approach in the domain of service-oriented architectures (SOA), which is likely applicable in the domain of cloud computing, as well. Documenting the decisions for patterns would result in a record of a combination of patterns in a single application scenario. By comparing the patterns used in multiple applications, new composite patterns could then be identified.

### 8.3.4 Pattern-Based Architectural Modeling

Fleck et al. [FTLW14] and Bergmayr et al. [BTN+14] have already used UML<sup>60</sup> as a means to model applications based on the cloud computing patterns. These efforts could be extended to other architecture description languages (ADL), such as ACME<sup>61</sup> or AADL,<sup>62</sup> to use the cloud computing patterns as modeling elements.

The reference implementations of the cloud computing patterns (Section 6.4) could likely be used during the refinement of such architectural models for their implementation. If architectural models are used in this fashion, runtime information could likely be connected to the originating architectural model. By doing so, the architectural model could

---

<sup>60</sup><http://www.uml.org/>

<sup>61</sup><http://www.cs.cmu.edu/~acme/>

<sup>62</sup><http://www.sei.cmu.edu/architecture/research/model-based-engineering/aadl.cfm>

provide an abstracted view of a running application. Such architectural views could likely be approached similarly to views on business processes [Sch15].

## 8.4 Chapter Summary

The answers to the research questions raised in Chapter 1 have been presented. Limitations of the cloud computing patterns obtained from industry use cases were listed. It has been shown that additional pattern identification should be performed to create patterns describing how to build the cloud itself, as there is a need for guidelines addressing the creation of *private clouds* (66). Furthermore, the refinement to technologies of abstract concepts covered by patterns was found to be very challenging. This extension formed one of the discussed research opportunities. The cloud computing patterns may bridge between industry and research as research-driven improvements of the abstract concepts may be contributed back to the pattern. The formalization of the cloud computing pattern languages may provide a basis to integrate the large number of patterns existing in the IT domain to obtain a complete *pattern language for IT applications*. The cloud computing patterns could also be used to drive and document architectural decisions. Finally, the cloud computing patterns could also be used as modeling elements in architecture description languages, which could enable the automated refinement of such models toward technology-specific models.



## APPENDIX A

---

### Detailed Pattern Engineering Process

---

Fig. A.1 shows the detailed pattern engineering process. The results of the pattern identification phase are detailed in Chapter 3, while the results of the pattern authoring phase are described in Chapter 4. The results of the pattern application phase regarding the organization of cloud computing patterns, as well as the pattern-based design method for cloud applications, are detailed in Chapter 5. The selection of applicable patterns and their use, also covered by this phase, are supported by the toolchain presented in Chapter 6.

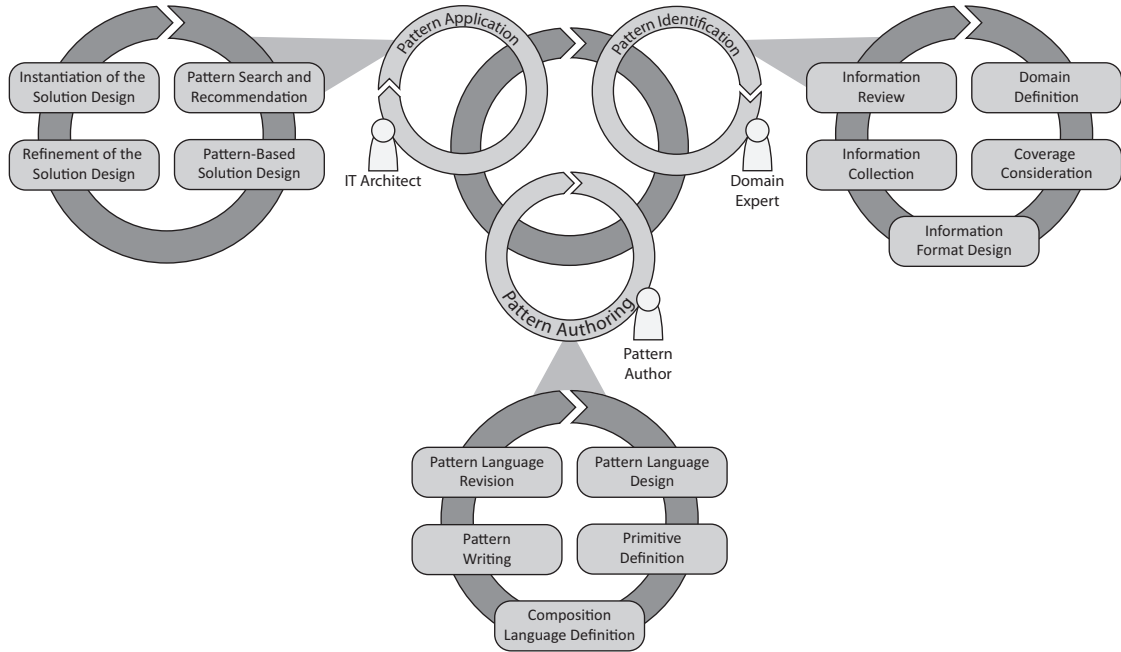


Figure A.1 – Detailed pattern engineering process

## APPENDIX B

---

### Detailed Mapping of Related Patterns

---

The following tables map related patterns to the cloud computing patterns presented in this work using the following relations:

**Refinement:** The related pattern describes the same concept as the mapped cloud computing pattern, but refines it to a specific set of technologies or provider-specific offerings.

**Aspect Refinement:** This is analogous to refinement, but only one aspect of the cloud computing pattern is discussed.

**Extension:** The related pattern describes a concept that is not addressed by the cloud computing patterns. This can be a technical or provider-specific detail. In this case, it should be investigated whether the same concept can be observed at other providers, as well, to identify new patterns.

**Generalization:** The related pattern summarizes the concepts of one or more cloud computing patterns.

**Similarity:** The related pattern is almost equivalent to a cloud computing pattern, as it discusses the same concept in a similar context and with a similar level of detail.

To differentiate between the cloud computing patterns mentioned in the following tables and the covered related patterns, only the names of cloud computing pattern are italicized.



**Table B.1** – Mapping of Cloud Design Patterns (AWS)

Cloud Design Patterns (AWS)				
Pattern	Summary	Mapped to	Relation	Explanation
Backnet	Use multiple virtual network cards connected to different networks to separate connections from users and management access.	Out of scope	Extension	Management of different virtual networks is not considered by the cloud computing patterns.
Bootstrap	Handle configuration files in a storage offering from where virtual servers acquire them at boot time.	<i>managed configuration</i> (247)	Aspect Refinement	The bootstrap pattern details polling a <i>managed configuration</i> upon boot time of a virtual server.
Cache Distribution	Host multiple replicas of content close to users.	<i>content distribution network</i> (300)	Refinement	Patterns describe the same strategy. The cache distribution pattern covers specific AWS technology.
Cache Proxy	Use virtual servers as cache servers to actively add caches to the scaled out application.	<i>elastic load balancer</i> (254)	Extension	The <i>elastic load balancer</i> treats all application components among which load is distributed equally. The cache proxy pattern enables a layering of these instances by introducing caches.
Clone Server	Clone a virtual server to handle scaling out.	<i>elastic load balancer</i> (254)	Aspect Refinement	The <i>elastic load balancer</i> only states that instances of virtual servers are added to the system. The Clone Server pattern describes one way of doing so.
Cloud DI	Pass configuration parameter for virtual machines by passing them in a start configuration handled by the runtime environment.	<i>hypervisor</i> (101), <i>elastic infrastructure</i> (87), <i>managed configuration</i> (247)	Extension	Using functionality of the <i>hypervisor</i> and <i>elastic infrastructure</i> to enable managed configurations is not yet considered.
CloudHub	Use the cloud provider as a hub to connect multiple remote sites. Each site establishes a VPN connection to the cloud provider.	<i>integration provider</i> (234)	Refinement	Using a cloud to integrate distributed hosting environments is discussed by the <i>integration provider</i> pattern.
DB Replication	Establish slaves to relational databases in multiple regions and keep them in sync to secure data.	<i>relational database</i> (115)	Extension	Establishing database replicas manually is not yet considered by the <i>relational database</i> pattern.
Deep Health Check	Use a custom PHP page that checks the availability of all components of an application. Check the availability of this page using the AWS load balancer health check function.	<i>watchdog</i> (260)	Aspect Refinement	The <i>watchdog</i> describes the use of provider-supplied health checks, i.e., for network availability, heartbeats are sent from application components, and application level checks are performed to verify proper operation of application functions. The deep health check pattern describes the last option.
Direct Hosting	Use storage offerings to host websites. Thus, rely on a platform offering instead of using servers to reduce costs.	<i>blob storage</i> (112)	Extension	The <i>blob storage</i> pattern does not consider the special use to host websites directly. It should be investigated if this type of use is only available in AWS.

Cloud Design Patterns (AWS)				
Pattern	Summary	Mapped to	Relation	Explanation
Direct Object Upload	If many users upload files to an application, for example, video and images, let them upload to storage offerings directly to avoid this load in the application.	<i>data access component</i> (188)	Extension	Bypassing the <i>data access component</i> temporarily is not yet considered.
Floating IP	Switch an IP address from one server to another in order to decrease downtime during an update.	<i>update transition process</i> (275)	Aspect Refinement	The <i>update transition process</i> describes the steps to update an application component. One step is to enable an instantaneous switch between application components of different versions.
Hybrid Backup	Back up on-premise data to the cloud to keep backups in a geographically separate location.	<i>hybrid backup</i> (314)	Refinement	Both patterns describe the same aspects.
Inmemory DB Cache	Use in memory caches to increase read access performance to databases.	<i>relational database</i> (115)	Extension	Technical database optimization and caching techniques are not considered by the cloud computing patterns, but are referenced.
Job Observer	Scale the number of virtual servers processing batch jobs from a queue based on the number of queued messages.	<i>elastic queue</i> (257)	Refinement	The <i>elastic queue</i> pattern describes the same aspect.
Monitoring Integration	Install monitoring software on the virtual servers in addition to monitoring information offered by the provider. Collect all of these monitoring data using the monitoring software to obtain a complete view.	Out of scope	Extension	Enabling application-level monitoring is not considered by the cloud computing patterns.
Multi-Load balancer	Different clients and devices may require different access configurations, such as sol encryption, session affinity, etc. This is handled by using different load balancers for the different types of clients.	Out of Scope	Extension	Management of different access credentials for multiple clients is not considered by the cloud computing patterns.
Multi-Datcenter	Distribute load across virtual servers running in geographically separate locations.	<i>elastic load balancer</i> (254), <i>elastic queue</i> (257), <i>elasticity manager</i> (250)	Extension	The load balancing patterns of the cloud computing patterns currently do not consider distribution among multiple data centers.
Multi-Server	Use a load balancer to distribute requests among multiple virtual servers.	<i>elastic load balancer</i> (254), <i>elastic queue</i> , (257) <i>elasticity manager</i> (250)	Aspect Refinement	Distributing load among resources is considered by all load balancing patterns of the cloud computing patterns. The multi-server pattern does not consider the characteristics of this load (e.g. synchronous, asynchronous, etc.).
NFS Replica	Replicate static data from an NFS service to the virtual hard drive of virtual servers.	<i>blob storage</i> (112)	Extension	The <i>blob storage</i> pattern only describes how data is accessed. The NFS replica pattern describes how such data can be replicated in order to increase access performance.

Cloud Design Patterns (AWS)				
Pattern	Summary	Mapped to	Relation	Explanation
NFS Sharing	Use a server offering a network file system (NFS) service to share data among multiple virtual server instances.	<i>blob storage</i> (112)	Refinement	The <i>blob storage</i> pattern describes a file system-like storage offering. The NFS Sharing pattern covers one possible implementation of this pattern for Amazon AWS.
Ondemand Disk	Add virtual hard drives to a running virtual server as needed.	<i>blob storage</i> (110)	Refinement	The Ondemand Disk pattern covers how the functionality of the <i>blob storage</i> pattern can be used on demand in AWS.
OnDemand Nat	Most virtual servers running at a cloud provider do not need outbound internet connectivity, which is therefore disabled for security. When virtual servers are updated, special virtual machines are provisioned that handle network address translation (NAT), thus enabling access to the Internet during update.	Out of scope	Extension	Using manually configured virtual servers as networking components is not considered by the cloud computing patterns.
Operational Firewall	Use virtual firewall configurations to grant access to each organizational unit of personnel performing a certain function for the application such as maintenance, monitoring, or development.	<i>virtual networking</i> (132)	Extension	Securing the different user and management groups of an application using network access configuration is not yet considered.
Priority Queue	Assign different priorities to messages and process those of higher priority first to ensure an adequate level of service.	<i>message-oriented middleware</i> (136)	Aspect Refinement	The <i>message-oriented middleware</i> (136) pattern covers the use of message prioritization, as well.
Private Distribution	Use access control, links valid only at certain times, and user-specific URLs to control access to files in a storage offering.	<i>blob storage</i> (112), <i>data access component</i> (188)	Refinement	The <i>blob storage</i> and <i>data access component</i> patterns currently do not consider the technical aspect of restricting access to certain times.
Queuing Chain	Use messaging to coordinate jobs handled by multiple systems and increase throughput using parallelization.	<i>batch processing component</i> (185), competing consumer [HW03]	Refinement	The <i>batch processing component</i> describes a similar approach to assign messages to processors. The competing consumer pattern of Hohpe and Woolf [HW03] describes this concept in general.
Read Replica	Create replicas of a relational database to increase read access performance. All replicas are updated from a master database where writes also occur.	<i>eventual consistency</i> (126)	Refinement	The eventual consistency pattern describes the scenario of having a master database that is replicated to multiple read replicas.
Rename Distribution	Rename new versions of files to avoid that caches serve obsolete versions.	<i>blob storage</i> (112), <i>eventual consistency</i> (126)	Extension	The eventual consistent behavior of Amazon's <i>blob storage</i> allows this strategy to enable immediate visibility of new file versions for all clients. It should be investigated whether this is also possible for other cloud providers.

Cloud Design Patterns (AWS)				
Pattern	Summary	Mapped to	Relation	Explanation
Rewrite Proxy	Use a proxy server in addition to a Web application to serve static content more quickly by assigning requests to content delivery networks and provider-supplied storage offerings using the proxy. This even works without adjusting a possibly existing Web application.	<i>hybrid multimedia web application</i> (323)	Aspect Refinement	The <i>hybrid multimedia web application</i> does not describe how an existing application can exploit external storage offerings and provider-services making content accessible.
Scale Out	Distribute load across virtual servers using a load balancer and adjust the number of virtual servers with respect to the workload.	<i>elastic load balancer</i> (254)	Refinement	Describes how the <i>elastic load balancer</i> pattern is implemented for Amazon AWS.
Scale up	Change the capabilities of a virtual server (CPU, memory, etc.) by restarting it with a different configuration.	<i>elastic load balancer</i> (254), <i>elastic queue</i> (257), <i>elasticity manager</i> (250)	Extension	The cloud computing patterns only consider elastic scaling, the scaling by addition of more resource numbers. An incorporation of a scaling up strategy – even though it may be limited in the scope of cloud computing – could be beneficial.
Scheduled Autoscaling	Process batch jobs contained in a queue when their processing is most feasible.	<i>batch processing component</i> (185)	Refinement	The <i>batch processing component</i> pattern describes the same aspect.
Scheduled Scale Out	Proactively add virtual servers to an application during times of the day, week, etc. when higher loads are common. This ensures improved responsiveness compared with scaling based on monitored load changes.	<i>once-in-a-lifetime workload</i> (33), <i>periodic workload</i> (29)	Refinement	The scheduled scale out pattern covers how the workload types described by the <i>once-in-a-lifetime workload</i> and <i>periodic workload</i> patterns can be handled in Amazon AWS.
Server Swapping	If a virtual server is faulty, start a replacement and reassign the virtual hard drive of the faulty server to the replacement.	<i>watchdog</i> (260)	Aspect Refinement	The server swapping pattern details the technical replacement of a faulty server.
Sharding Write	Distribute tables across instances of a database management system to increase write performance.	<i>relational database</i> (115)	Extension	Technical database optimization and caching techniques are not considered by the cloud computing patterns, but other sources are referenced.
Snapshot	Backup of EC2 servers based on virtual hard drive images.	<i>hybrid backup</i> (314)	Aspect Refinement	The snapshot pattern describes the backup based on virtual machine images, which is one aspect of the <i>hybrid backup</i> pattern.
Stack Deployment	Configure a group of virtual servers and start or stop them as a whole using this template.	Out of scope	Extension	The configuration of virtual servers as a deployment unit is not yet considered.
Stamp	Start multiple virtual servers by replicating a running server.	<i>elastic load balancer</i> (254), <i>elastic queue</i> (257), <i>elasticity manager</i> (250)	Extension	The stamp pattern discusses how Amazon AWS functionality can be used to handle the addition of virtual server instances required by the <i>elastic load balancer</i> , <i>elastic queue</i> , and <i>elasticity manager</i> patterns.

Cloud Design Patterns (AWS)				
Pattern	Summary	Mapped to	Relation	Explanation
State Sharing	Make applications hosted on virtual servers stateless. Keep application state in a provider-supplied storage offering.	<i>stateless component</i> (171)	Refinement	The <i>stateless component</i> pattern describes that application state and session state should not be contained in component instances. The state sharing pattern describes how application state can be kept external.
Storage Index	When storing large files in a storage offering, save metadata in a key-value storage that can be queried more efficiently.	<i>blob storage</i> (112)	Extension	The <i>blob storage</i> pattern assumes that the cloud provider can create an index of contained files.
URL Rewriting	Manage static content in a provider-supplied storage offering. This content is referenced from a Web application to be acquired from the provider storage offering directly by browsers accessing the Web application.	<i>hybrid multimedia web application</i> (323)	Refinement	The <i>hybrid multimedia web application</i> describes how large multi-media files can be managed in an elastic cloud to be accessed directly by clients. The URL rewriting pattern describes this concept using Amazon AWS technologies.
WAF Proxy	Web application firewall software is installed on proxy servers to determine the number of required licenses. Other virtual servers behind these proxies can be scaled out flexibly.	Out of Scope	Extension	Manual installation and management of firewall software is not considered by the cloud computing patterns.
Web Storage	Serve large files directly from storage offerings instead of webservers to decrease load on servers. Browsers accessing the webserver thus directly access the large files.	<i>hybrid multimedia web application</i> (323)	Refinement	The <i>hybrid multimedia web application</i> pattern discusses a similar setup to serve large files more efficiently to users. The web storage pattern describes how this can be realized using AWS.
Web Storage Archive	Use storage offerings to archive application log files.	Out of scope	Extension	Log file management is not yet considered by the cloud computing patterns.
Weighted Transition	When switching between two instances of a Web application, use a weighted DNS-based round robin algorithm to transition between both running instances.	<i>update transition process</i> (275)	Aspect Refinement	The <i>update transition process</i> describes the steps to update an application component. This may include a transitional switch between two component versions.
Write Proxy	When transferring large files or many small files to a storage offering, archive them or split them first. Then send them to a virtual server at the cloud provider where the alterations are reversed. This makes the file transfer more durable.	<i>data access component</i> (188)	Extension	The <i>data access component</i> does not yet cover special techniques for upload of large files or many small files.

Table B.2 – Mapping of Cloud Design Patterns (Windows Azure)

Cloud Design Patterns (Azure)				
Pattern	Summary	Mapped to	Relation	Explanation
Cache-Aside	Store accessed data in a cache to improve performance. Check freshness of data upon later accesses.	Out of scope	Extension	The details of cache management are not covered by the cloud computing patterns. This pattern is, however, related to eventual consistency, as it covers a strategy to keep data replicas consistent.
Asynchronous Messaging	Messaging is used by many distributed applications to enable scalability and resiliency.	<i>message-oriented middleware</i> (136)	Extension	The asynchronous messaging pattern covers messaging in greater detail, where the <i>message-oriented middleware</i> (136) pattern references patterns of Hohpe and Woolf [HW03].
Autoscaling	Manually adjusting resource number can be inefficient; thus, this process should be automated.	<i>elasticity manager</i> (250)	Refinement	The autoscaling guidance describes how to scale an application based on resource utilization just as the <i>elasticity manager</i> pattern suggests. Other strategies based on monitoring synchronous requests or asynchronous requests as described by the <i>elastic load balancer</i> (254) and <i>elastic queue</i> (257) pattern, respectively, are not covered.
Caching	Often-accessed data should be handled by in-memory caches or shared caches to increase performance.	Out of scope	Extension	Caching strategies are not covered by the cloud computing patterns. The eventual consistency and strict consistency pattern seem related, as they cover general management of data replicas.
Circuit Breaker	The repeated execution of an operation that is likely to fail is avoided to reduce the load generated by retries.	Out of scope	Extension	The circuit breaker pattern describes how application components should behave if they cannot interact with remote functionality. In the scope of the cloud computing patterns, this interaction is considered to be enabled through messaging. The circuit breaker pattern should be considered if synchronous communication is used.
Command and Query Responsibility Segregation (CQRS)	Operations that read data are segregated from operations that update data by using separate interfaces.	<i>data access component</i> (188)	Extension	The <i>data access component</i> pattern describes how access functionality to retrieve data can be offered by an application component. The CQRS pattern extends the covered concept toward a use of several data access components for different purposes (read and write).

Cloud Design Patterns (Azure)				
Pattern	Summary	Mapped to	Relation	Explanation
Compensating Transaction	When using eventual consistent storage offerings that do not support distributed transactions, define compensation actions for each of the changes made by a process in order to possibly revert all of them if a step fails.	Out of scope	Extension	The transactional processor pattern seems to cover a related topic, as it describes the transaction-based interaction with storage offerings.
Competing Consumers	Multiple message processors access the same message channel in order to balance load among them.	<i>message-oriented middleware</i> (136)	Refinement	The competing consumers pattern has originally been described by Hohpe and Woolf [HW03]. It has been summarized by the <i>message-oriented middleware</i> pattern.
Compute Partitioning	Application functionality should be decomposed into logical components that are then summarized to physical partitions that can be deployed in an efficient manner.	<i>distributed application</i> (160), <i>multi-component image</i> (206)	Refinement	The <i>distributed application</i> pattern describes the same decomposition of application functionality into logical components that are then summarized into tiers that can be deployed as a unit. The <i>multi-component image</i> also seems related, as it describes how multiple application components should be summarized into a virtual machine image.
Compute Resource Consolidation	Summarize different application functionality into a computational unit that can be deployed in order to increase the utilization of the resulting unit and reduce hosting costs.	<i>distributed application</i> (160), <i>two-tier cloud application</i> (290), <i>three-tier cloud application</i> (294)	Aspect Refinement	The <i>distributed application</i> pattern describes the decomposition of application functionality into multiple logical application components that are then summarized into tiers to be deployed as units. The <i>two-tier cloud application</i> and <i>three-tier cloud application</i> patterns are often used in distributed applications.
Data Consistency	Consistency of data has to be weighted against availability in large distributed applications.	<i>eventual consistency</i> (126), <i>strict consistency</i> (123), <i>data access component</i> (188)	Refinement	Data consistency is described as strict or eventual. The <i>data access component</i> covers strategies about how to alter this behavior displayed by storage offerings.
Data Partitioning	Large volumes of data should be partitioned to improve scalability and optimize performance.	Out of scope	Extension	Strategies for how to distribute data across multiple storage offerings are not yet considered by cloud computing patterns.
Data Replication and Synchronization	Data stored in multiple locations has to be synchronized, which has to consider the updates to data.	<i>content distribution network</i> (300), <i>hybrid data</i> (311), <i>hybrid backup</i> (314)	Generalization	The cloud computing patterns describe the behavior of data replication and synchronization always within the scope of patterns that use these concepts. A general overview of strategies for replication and synchronization has not been covered.
Event Sourcing	An append-only storage is used to collect all alterations of application data instead of managing only the current state to avoid update conflicts, auditing, responsiveness, etc.	Out of scope	Extension	The event sourcing pattern seems to describe a type of storage offering that has not yet been generalized by the cloud computing patterns.

Cloud Design Patterns (Azure)				
Pattern	Summary	Mapped to	Relation	Explanation
External Configuration Store	Configuration data is not kept in deployment packages of an application but kept at a centralized location.	<i>managed configuration</i> (247)	Aspect Refinement	The external configuration store describes the pull model to manage configurations.
Federated Identity	Authentication of accesses is delegated to an external identity provider.	Out of scope	Extension	The cloud computing patterns currently do not consider identity management.
Gatekeeper	A dedicated application component acts as a broker between clients and an application to validate and sanitize requests to increase security.	Out of scope	Extension	The cloud computing patterns currently do not consider the validation and security checks for requests issued toward an application. The gatekeeper pattern could extend the <i>elastic load balancer</i> (254) and <i>elastic queue</i> (257) patterns, which also process requests to an application.
Health Endpoint Monitoring	Functional checks within an application are performed by specialized components, which make the results of such checks available via an interface.	<i>watchdog</i> (260)	Aspect Refinement	The health endpoint monitoring pattern describes how to implement the aspect of application-level checks described by the <i>watchdog</i> pattern.
Index Table	Create an index of data that is often accessed by queries to a database in order to increase performance.	Out of scope	Extension	Indexes are a common concept for database optimization. The index table pattern, therefore, is related to the <i>relational database</i> (115) pattern. However, the <i>relational database</i> references other sources covering index tables and does not describe this as part of the pattern.
Instrumentation and Telemetry	Custom diagnostic functions (instrumentation) and the collection of provided information (telemetry) should be added to an application in order to monitor its status.	<i>watchdog</i> (260)	Refinement	The <i>watchdog</i> pattern covers monitoring application health using provider-supplied data, heartbeat messages of components, and application-level checks. General strategies for how to collect monitoring information about a cloud application are not covered by the cloud computing patterns.
Leader Election	Multiple actions performed by concurrently running instances of tasks handled by an application are coordinated by electing one instance as a leader that coordinates other instances.	Out of scope	Extension	The cloud computing patterns consider applications to process requests in parallel. A need for synchronization of parallel tasks has not been considered.
Materialized View	Views of data are stored separately to increase query performance.	Out of scope	Extension	Materialized views are a concept for database optimization. The materialized view pattern, therefore, seems to be related to the <i>relational database</i> (115) pattern. However, the <i>relational database</i> pattern references other sources covering materialized views and does not describe this as part of the pattern.



Cloud Design Patterns (Azure)				
Pattern	Summary	Mapped to	Relation	Explanation
Multiple Datacenter Deployment	Applications are deployed into multiple data centers to increase availability and access performance.	<i>content distribution network</i> (300)	Extension	The multiple datacenter deployment guidance focuses on requirements and regulations that have to be considered when hosting an application in multiple data centers.
Pipes and Filters	A task of complex processing is decomposed into a series of discrete elements that can be reused.	<i>distributed application</i> (160)	Aspect Refinement	The pipes and filters decomposition of application functionality is one of the decomposition strategies covered by the <i>distributed application</i> pattern.
Priority Queue	Messages are prioritized based on priority information associated with messages.	<i>message-oriented middleware</i> (136)	Extension	The prioritization of messages is summarized in the <i>message-oriented middleware</i> pattern.
Queue-Based Load Leveling	A messaging queue is used as a buffer for a message processor, so that load peaks will not overload the message processor.	<i>batch processing component</i> (185), <i>elastic queue</i> (257)	Refinement	The <i>batch processing component</i> describes how messages can be stored until their processing is feasible. The <i>elastic queue</i> pattern scales an application based on messages in a queue; thus, it can implement similar behavior.
Retry	An operation that accesses functionality over a network is re-executed in order to cope with temporary failures.	Out of scope	Extension	The cloud computing patterns consider message-based applications, thus relying on asynchronous communication. Unavailability of synchronously enacted interfaces could be a promising extension.
Runtime Re-configuration	Changes to an application at runtime do not require redeployment, but are updated during runtime of application components.	<i>managed configuration</i> (247)	Aspect Refinement	The runtime reconfiguration pattern describes the push model used to update configurations of application component instances as described by the <i>managed configuration</i> pattern.
Scheduler Agent Supervisor	A set of actions performed by distributed resources is coordinated by a central agent that enacts functionality offered by the distributed resources.	<i>distributed application</i> (160)	Extension	The scheduler agent supervisor considers a centralized notion of a workflow executed by the distributed application. This concept is covered by the <i>distributed application</i> pattern as a process-based decomposition strategy.
Service Metering	Use of the application is metered in order to bill it to users, plan for future requirements, etc.	Out of scope	Extension	Metering of application use is not explicitly covered by the cloud computing patterns. These topics are valid extension points, especially if the cloud application should be offered as a service.
Sharding	Data handled by a storage offerings is partitions into subsets to be distributed among multiple storage offerings in order to increase performance.	<i>relational database</i> (115)	Extension	Sharding of database tables is a common concept of database optimization of relational databases. Related sources have been referenced by the <i>relational database</i> pattern.
Static Content Hosting	Static Web content is directly provided by storage offerings to client browsers in order to avoid the necessity of compute resources.	<i>hybrid multimedia web application</i> (323)	Extension	The <i>hybrid multimedia web application</i> pattern describes a similar concept to manage large files that are components of websites.

Cloud Design Patterns (Azure)				
Pattern	Summary	Mapped to	Relation	Explanation
Throttling	Resource consumption of individual application instances, tenants, etc. are only scaled up until a certain threshold. If the threshold is reached, consumption of additional resources is disallowed to control resource consumption.	<i>feature flag management process (271)</i>	Extension	The <i>feature flag management process</i> pattern describes how an application can cope with the unavailability of resources. Controlling the maximum of allowed resources is a separate concept that could be integrated or added.
Valet Key	Access tokens are issued to clients of an application so that they may upload data directly to a storage offering and avoid relaying this data through an application component.	<i>data access component (188)</i>	Extension	The valet key pattern optimizes data upload, which is one aspect handled by the <i>data access component</i> pattern.

**Table B.3 – Mapping of Cloud Architecture Patterns**

Cloud Architecture Patterns				
Pattern	Summary	Mapped to	Relation	Explanation
Auto-Scaling	Addition and removal of resources to and from an application should be automated in order to reduce costs and avoid human errors.	<i>elasticity manager</i> (250), <i>elastic load balancer</i> (254), <i>elastic queue</i> (257), <i>elasticity management process</i> (267)	Extension	The auto-scaling pattern briefly summarizes the cloud computing patterns handling elasticity. It covers different monitoring information to identify increasing and decreasing workload. An extension of the concept is a brief discussion of scaling strategies, i.e., how quickly to react to workload increase and decrease, how many resources to “overprovision”, etc.
Busy Signal	If a request to a service is answered by a busy signal, i.e., a notification that the service is temporarily unavailable, it is retried a couple of times prior to treating the called service as unavailable.	Out of scope	Extension	The cloud computing patterns mostly consider message-based applications to ensure <i>loose coupling</i> (156) among interacting application components. The busy signal pattern would also be an adequate extension to discuss synchronous interaction.
CDN	Content served by an application is distributed globally in order to increase performance for the globally distributed user group.	<i>content distribution network</i> (300)	Similarity	Both patterns describe the same aspect.
Colocate	Deploy nodes that interact often close to each other, i.e., in the same data center.	Out of scope	Extension	A deployment optimization of application components among multiple data centers is not yet considered by the cloud computing patterns.
Database-Sharding	A database is distributed among multiple databases (shards) having the same schema in order to horizontally scale data.	<i>relational database</i> (115)	Extension	Database sharing is a long-established optimization for databases. It has not been covered explicitly by the cloud computing patterns.
Horizontal Scaling Compute	Compute nodes are added and removed dynamically from the application. Nodes are stateless to simplify this process.	<i>stateless component</i> (171), <i>standby pooling process</i> (279), <i>loose coupling</i> (156)	Extension	The horizontal scaling compute pattern summarizes aspects of the <i>stateless component</i> pattern (how can resources be added easily) and discusses the availability of new resources (related to the <i>standby pooling process</i> pattern). Elasticity as a concept is introduced by the IDEAL cloud computing properties rather than a separate pattern. A discussion of different rental models forms an extension introduced by the horizontal scaling compute pattern.
MapReduce	Processing of parallelizable datasets by different nodes.	<i>map reduce</i> (106)	Refinement	This pattern focuses on the Azure Hadoop Service, which has been available as a preview at the time of writing. Some map reduce programming languages are discussed briefly.

Cloud Architecture Patterns				
Pattern	Summary	Mapped to	Relation	Explanation
Multisite Deployment	Instances of an application are globally distributed in order to better serve globally distributed users. Data of instances is synchronized.	<i>hybrid application functions</i> (320)	Refinement	The <i>hybrid application functions</i> pattern distributes application functions among multiple geographic locations, as the functions have different requirements. The multisite deployment pattern hosts all application functions in multiple geographic locations.
Node Failure	Application functionality running on a compute node, for example, a virtual machine, reacts to signals of the cloud provider, enabling a graceful shutdown of the resource.	Out of scope	Extension	The cloud computing patterns consider application components to possibly fail at any time without notice and enable the application to cope with this behavior. Considering a graceful shutdown poses an extension that is not currently covered.
Queue-Centric Workflow	Interactive requests are handled by a user interface and put into a queue where they are asynchronously picked up by another node for processing. This is mostly done for requests that are time-consuming, resource-intensive, or depend on remote services.	<i>processing component</i> (180), <i>user interface component</i> (175), <i>two-tier cloud application</i> (290), <i>idempotent processor</i> (197)	Generalization	The queue-centric workflow pattern summarizes concepts for the <i>processing component</i> and <i>user interface component</i> patterns. The composition of these two patterns seems similar to the <i>two-tier cloud application</i> pattern without discussing data handling in detail. The problem of message duplicates is also covered similarly to the <i>idempotent processor</i> pattern.
Valet Key	Issue tokens to clients of an application that allow them to temporarily interact directly with storage offerings to upload files. This avoids load on application components through which the upload would have to take place.	<i>data access component</i> (188)	Extension	The <i>data access component</i> pattern does not yet consider mechanisms to temporarily allow direct interaction with the storage offering it provides access to.

**Table B.4** – Mapping of Patterns from CloudPatterns.org

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Audit Monitor	Access information to services is monitored to support regulatory and contractual obligations.	<i>elastic infrastructure</i> (87), <i>elastic platform</i> (91), <i>Infrastructure as a Service (IaaS)</i> (45)	Extension	The audit monitor describes the monitoring of services in detail, i.e., how monitoring is enabled at the provider.
Automated Administration	Management tasks are automated as scripts and handled by a central automation engine.	All patterns of the management process category	Generalization	The cloud computing patterns detail multiple management process patterns that are handled by dedicated management components.
Automated Scaling Listener	Workload experienced by a cloud service is monitored in order to scale resources accordingly.	<i>elastic load balancer</i> (254)	Aspect Refinement	The automated scaling listener summarizes the concept of scaling rules in order to add and remove resources based on certain conditions.
Bare-Metal Provisioning	Remote management support of hardware is used to dynamically install operating systems to servers.	Out of scope	Extension	The cloud computing patterns do not cover the organization of physical hardware in order to build a cloud.
Billing Management System	Cloud providers monitor the use of cloud resources in order to enable pay-per-use pricing models.	<i>Infrastructure as a Service (IaaS)</i> (45), <i>Platform as a Service (PaaS)</i> (49), <i>Software as a Service (SaaS)</i> (55)	Extension	The cloud computing patterns only describe how the cloud service models behave. The billing management system pattern also details how cloud providers build such a service internally.
Broad Access	Cloud services support multiple clients.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Burst In	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Burst Out to Private Cloud	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Burst Out to Public Cloud	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Centralized Remote Administration	Cloud providers consolidate multiple management features in one user interface.	Out of scope	N/A	The consolidation of management interfaces would propose a valid extension to the behavior description of cloud providers.

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Cloud Balancing	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Cloud Bursting	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Cloud Storage Device	Cloud providers offer different storage devices, such as files, blocks, datasets and objects.	<i>block storage</i> (110), <i>relational database</i> (115), <i>key-value storage</i> (119)	Generalization	The cloud storage device pattern gives a brief overview of available cloud storage offerings. The introduced "objects" type is an extension, but only described very briefly.
Cloud Usage Monitor	Usage of cloud resources can be monitored using agents (monitor accesses), resource agents (monitor state of resources), or polling agents (periodically request resource state).	<i>Infrastructure as a Service, (IaaS)</i> (45) <i>Platform as a Service (PaaS)</i> (49), <i>Software as a Service (SaaS)</i> (55)	Extension	The cloud usage monitor details the monitoring of resource use implemented by a cloud provider to enable pay-per-use billing.
Cross-Hypervisor Workload Mobility	Standardized virtual machine formats, such as OVF, <sup>63</sup> are used to transfer virtual machines between hypervisors.	Migration Patterns introduced in [FLR+13]	Extension	The management patterns started work on the migration of existing applications. Hypervisor compatibility was a major issue. The final version of the cross-hypervisor workload mobility pattern may provide solutions for these challenges.
Cross-Storage Device Vertical Tiering	A LUN (logical unit identifier in a storage area network) is used to move LUN disks.	<i>elastic infrastructure</i> (87)	Extension	Management of storage area networks is not covered by the cloud computing patterns.
Direct I/O Access	A virtual server directly accesses hardware in the physical host.	<i>hypervisor</i> (101)	Extension	The technical specifics of hypervisors are not covered by the cloud computing patterns.
Direct LUN Access	A virtual server may directly access a LUN disc.	<i>block storage</i> (110)	Aspect Refinement	The <i>block storage</i> pattern discusses, in general, how virtual disks may be used over a network. LUN access is one technical implementation of this aspect.
Dynamic Data Normalization	Data handled in a cloud storage service is normalized to avoid duplicate storage of the same data.	<i>block storage</i> (110)	Extension	The optimization of storage performed by a cloud provider is not covered by the cloud computing patterns.
Dynamic Failure Detection and Recovery	A watchdog supervises cloud resources and replaces failing ones.	<i>watchdog</i> (260)	Similarity	Both patterns describe similar failure detection and recovery concepts.

<sup>63</sup><http://www.dmtf.org/standards/ovf>

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Dynamic Scalability	The capabilities of cloud resources are adjusted with respect to experienced workload.	<i>elasticity manager</i> (250), <i>elastic load balancer</i> (254), <i>elastic queue</i> (257)	Extension	The dynamic scalability pattern focuses on how to incorporate scaling up with scaling out and relocation of resources. The cloud computing patterns currently focus on scaling out.
Elastic Disk Provisioning	Virtual disks are billed according to the contained data, not based on the size of the disk.	Out of scope	Extension	The cloud computing patterns do not cover the organization of physical hardware in order to build a cloud.
Elastic Environment	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Elastic Network Capacity	The capabilities of the connection network are dynamically adjusted to the experienced workload	<i>virtual networking</i> (132)	Extension	The virtual network pattern currently does not specifically describe how bandwidth can be configured dynamically.
Elastic Resource Capacity	CPUs and RAM are dynamically added to virtual servers in order to adjust their capabilities to experienced workload.	<i>elastic infrastructure</i> (87)	Extension	The elastic resource capacity pattern discusses the technical detail of virtual server scaling.
External Virtual Server Accessibility	Virtual servers are connected using virtual switches.	<i>virtual networking</i> (132)	Extension	The final version of the pattern should be compared with the mapped pattern(s) to find possible extensions / establish relations.
Failover System	For IT resources, a redundant or standby resource instance is provided that instantly replaces a failing resource. This failover can be active-active (both resources handle workload) or active-passive (one resource handles workload, the other is only active if the first resource becomes unavailable).	<i>distributed application</i> (160)	Generalization	The failover system generally describes why cloud applications and cloud services use multiple resources to enable higher performance and resiliency.
Hypervisor	Physical hardware is abstracted into virtual hardware in order to host multiple virtual servers on a physical service.	<i>hypervisor</i> (101)	Similarity	Both hypervisor patterns describe similar concepts.
Hypervisor Clustering	Hypervisors are installed on multiple physical servers for resiliency.	<i>hypervisor</i> (101)	Extension	The <i>hypervisor</i> pattern does not cover how to build a hypervisor, but when and how to use one.
Infrastructure-as-a-Service (IaaS)	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Intra-Storage Device Vertical Data Tiering	A cloud storage device integrates multiple physical disks.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Load Balanced Virtual Server Instances	Virtual servers are dynamically moved among physical servers to balance workload.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Load Balanced Virtual Switches	Traffic is balanced across multiple virtual and physical switches.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Load Balancer	Workload is distributed among multiple resources according to several distribution functions.	<i>elastic load balancer (254), elastic queue (257), elasticity manager (250)</i>	Generalization	The cloud computing patterns only describe load balancing in addition to elastic scaling of resources. The load balancer pattern covers strategies for load balancing in general. This functionality is often offered by the cloud provider.
Logical Network Perimeter	An isolation of a network environment from the rest of a communications network to establish a virtual network boundary between IT resources.	<i>virtual networking (132)</i>	Similarity	The logical network perimeter covers virtual firewalls and virtual private networks between a cloud customer and a cloud provider, similar to the <i>virtual networking</i> pattern.
Memory Over-Committing	Virtual servers hosted on one physical server are assigned more virtual memory than what is physically available on the host.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Multi-Device Broker	A cloud service is encapsulated by a broker in order to validate accesses, transform protocols, provide access for different devices, etc.	<i>provider adapter (243), data access component (188)</i>	Generalization	The multi-device broker summarizes concepts to access different cloud offerings that have been captured as multiple patterns in the cloud computing patterns. A possible extension is to route accessing devices to different user interfaces based on the type of device used; for example, mobile phones. This concept is currently not considered by the cloud computing patterns.
Multipath Resource Access	IT resources are connected through redundant network paths.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Multitenant Environment	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
NIC Teaming	Networking cards are configured for concurrent use.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Non-Disruptive Service Relocation	Virtualization enables the migration of cloud resources without downtime.	Migration patterns introduced in [FLR+13]	Extension	The migration patterns consider the move of applications. The non-disruptive service relocation pattern may impact the described migration processes.



CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Pay-as-You-Go	Resource use is monitored to enable pay-per-use billing.	Out of scope	N/A	The cloud computing patterns do not cover how pay-per-use billing can be created for developed applications. This may be a valid extension point. The current version of the pay-as-you-go pattern, however, is too brief for an evaluation.
Pay-Per-Use Monitor	Based on monitored resource usage, pay-per-use billing models are enabled by the cloud provider.	<i>Infrastructure as a Service (IaaS)</i> (45), <i>Platform as a Service (PaaS)</i> (49), <i>Software as a Service (SaaS)</i> (55)	Extension	Details the monitoring of resource use implemented by a cloud provider to enable pay-per-use billing.
Persistent Virtual Network Configuration	Network configuration of virtual servers is stored centrally in order to be equivalent after a virtual server has been moved between physical servers.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Platform Provisioning	Virtual server images are used to provide ready-to-use runtime platforms.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud. The platform provisioning pattern seems to discuss the creation of an elastic platform.
Platform-as-a-Service (PaaS)	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Power Consumption Reduction	Hypervisors that are not in use are powered off.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Private Cloud	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Public Cloud	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Rapid Provisioning	A provisioning engine automates the setup of cloud resources.	<i>elastic infrastructure</i> (87), <i>elastic platform</i> (91)	Extension	The rapid provisioning engine details how application components and virtual servers may be provisioned by the cloud provider. This is related to the <i>elastic infrastructure</i> and <i>elastic platform</i> patterns. How this provisioning functionality works in detail is not covered by the cloud computing patterns.

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Ready-Made Environment	The cloud provider offers a predefined cloud-based platform to remotely develop and deploy applications.	<i>Platform as a Service (PaaS)</i> (49), <i>elastic platform</i> (91), <i>execution environment</i> (104), <i>hybrid development environment</i> (326)	Generalization	The ready-made environment details the resources offered by <i>PaaS</i> . The cloud computing patterns split these concepts among the patterns <i>elastic platform</i> , <i>execution environment</i> and <i>hybrid development environment</i> , as each of these services could be used individually.
Realtime Resource Availability	The state of a cloud resource is reported in real time.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Redundant Physical Connection for Virtual Servers	Physical server used a second networking card for resiliency.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Redundant Storage	Physical storage devices are made redundant so that a failure of the primary device can be handled by the redundant device.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Remote Administration System	Remote administration of cloud resources is enabled through a management interface.	<i>Infrastructure as a Service (IaaS)</i> (45), <i>Platform as a Service (PaaS)</i> (49), <i>Software as a Service (SaaS)</i> (55)	Aspect Refinement	All cloud service models support a self-service interface, which is not separately discussed by the cloud computing patterns.
Resilient Environment	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Resource Cluster	Multiple cloud resources are combined to a cluster in order to offer functionality as a single resources.	Out of scope	Extension	The clustering of physical resources to provide virtual machine hosting, synchronized databases, and other services that rely on multiple physical resources, is a concept that has been common in data centers and is not covered as a cloud-specific concept by the cloud computing patterns.
Resource Management	Cloud providers use tools and controls to isolate the management activities of customers.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Resource Management System	IT resources are coordinated to handle management actions, such as managing of virtual IT resource templates, allocation of resources, or load balancing replications.	<i>elastic infrastructure</i> (87), <i>elastic platform</i> (91)	Aspect Refinement	The management functionality of cloud environments is summarized by the cloud computing patterns for each of the considered hosting environments. The resource management system describes this functionality as a separate pattern.

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Resource Pooling	Identical IT resources are grouped into a pool to maintain their synchronicity.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Resource Replication	IT resources are instantiated multiple times based on templates of these resources (virtual images).	<i>elastic infrastructure</i> (87)	Aspect Refinement	The resource replication pattern details a functionality of the elastic infrastructure to start multiple instances of virtual servers based on a virtual server image. This is often used for elastic scaling operations.
Resource Reservation	Use of IT resources by customers is made exclusively to customers.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Self-Provisioning	A self-service interface is used to enable automated IT resource provisioning by customers.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Service Load Balancing	A cloud service is deployed multiple times, and requests of customers are load balanced among these service instances.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Service State Management	A cloud service is designed to use a state management system instead of holding data in memory.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Shared Resources	Physical resources are shared among multiple cloud customers.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Single Root I/O Virtualization	A physical I/O device is abstracted into multiple virtualized ones in order to be shared among cloud customers.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
SLA Management System	Monitored information about cloud service state is matched against service level agreements to ensure conformity.	Out of scope	Extension	Monitoring of service level agreements is not currently considered by the cloud computing patterns.
SLA Monitor	Conformity of cloud resources to defined service level agreements can be monitored using agents (monitor accesses), resource agents (monitor state of resources), or polling agents (periodically request resource state).	<i>Infrastructure as a Service (IaaS)</i> (45), <i>Platform as a Service (PaaS)</i> (49), <i>Software as a Service (SaaS)</i> (55)	Extension	The SLA monitor pattern details the monitoring of resource use implemented by a cloud provider to enable service level agreements.
Software-as-a-Service (SaaS)	The current version of this pattern only lists other of the abovementioned patterns that have to be combined by this pattern. Currently, no additional explanatory text is provided.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
State Management Database	The state of an application is handled in a separate database to be more scalable.	<i>stateless component</i> (171)	Aspect Refinement	The <i>stateless component</i> pattern covers how state can be provided with each request to the resource and how it can be handled in an external storage offering. The state management database details the former aspect.
Stateless Hypervisor	A hypervisor is booted using a boot image that is accessed over a network.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Storage Maintenance Window	LUN migration is used during planned maintenance in order to avoid downtime.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Storage Workload Management	Storage workload is load balanced among multiple storage systems managing LUNs.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Synchronized Operating State	Instead of using clustering techniques, heartbeat messages are used to synchronize virtual servers.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Usage Monitoring	Multiple cloud usage monitors are used to track and measure the IT resource usage.	Unknown	N/A	The current version of the pattern does not provide enough detail in order to be compared with the cloud computing patterns.
Virtual Server	A physical server is emulated to share the same physical server among multiple virtual ones.	<i>hypervisor</i> (101)	Aspect Refinement	The concept of a virtual server is described by the <i>hypervisor</i> pattern and not captured as a separate pattern.
Virtual Server Auto Crash Recovery	Virtual servers are monitored and recovered in case of operating system failure.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Server Connectivity Isolation	A virtual server is isolated from others with respect to networking using virtual switches.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Server Folder Migration	Virtual Server images are handled by LUNs.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Server NAT Connectivity	A virtual server connects to a network via an intermediary.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Server-to-Host Affinity	A virtual server is hosted on one target host and never moved.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Server-to-Host Anti-Affinity Pattern	A virtual server will never be hosted on a specified host.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.

CloudPatterns.org				
Pattern	Summary	Mapped to	Relation	Explanation
Virtual Server-to-Host Connectivity	A virtual switch is used to enable a secure network-based channel between virtual server and hypervisor.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Server-to-Virtual Server Affinity	A group of virtual servers will always be hosted on the same physical server.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Virtual Switch Isolation	Virtual switches are used to reduce network contention and bandwidth competition.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.
Workload Distribution	An IT resource is horizontally scaled, and a load balancing system assigns workload to the individual instances.	<i>elastic load balancer (254), elastic queue (257), elasticity manager (250)</i>	Aspect Refinement	The cloud computing patterns only describe load balancing in addition to elastic scaling of resources.
Zero Downtime	A virtual server is migrated to a different physical server if the physical server hosting it fails.	Out of scope	N/A	The cloud computing patterns do not cover the organization of hardware in order to build a cloud.



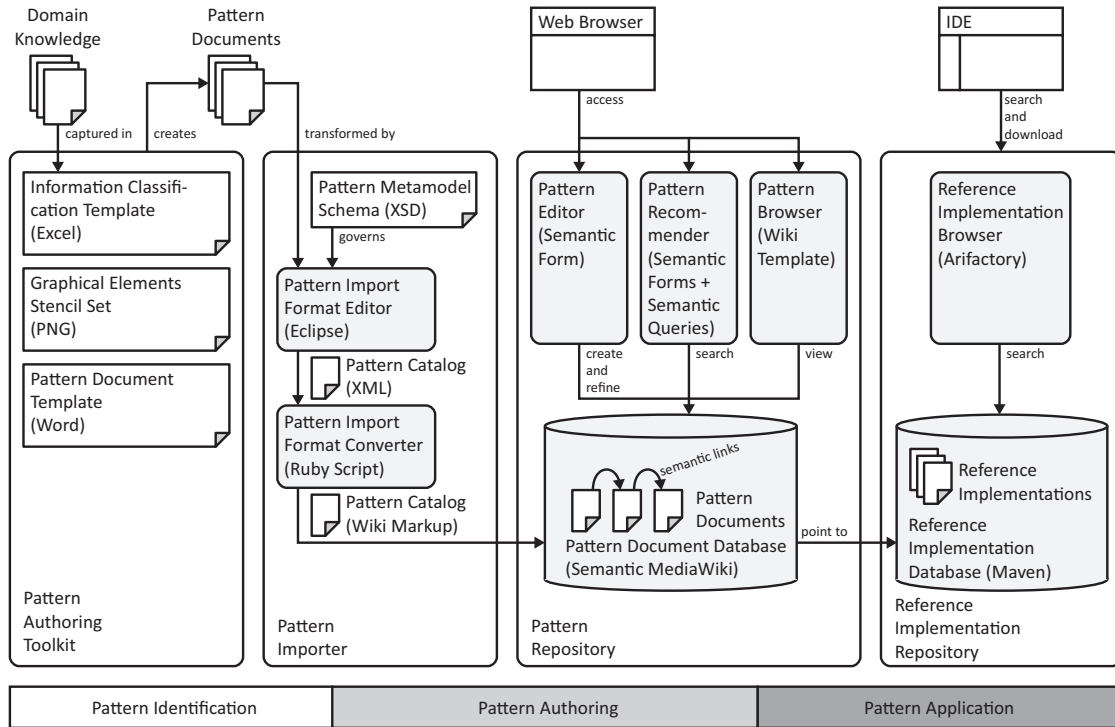
## APPENDIX C

---

### Toolchain for Cloud Computing Patterns

---

A complete view of the toolchain covered in Chapter 6 is given in Fig. C.1. This toolchain supports the phases of the pattern engineering process introduced in Section 1.3, which are pattern identification, pattern authoring, and pattern application.



**Figure C.1** – Detailed components of the toolchain supporting the cloud computing patterns



---

## Bibliography

---

- [ADKL14] V. Andrikopoulos, A. Darsow, D. Karastoyanova, and F. Leymann. “CloudDSF–The Cloud Decision Support Framework for Application Migration.” In: *Proceedings of the European Conference on Service-Oriented and Cloud Computing (ESOCC)*. 2014 (cit. on p. 87).
- [AF09] M. L. Abbott and M. T. Fisher. *The Art of Scalability: Scalable Web Architecture, Processes and Organizations for the Modern Enterprise*. Addison-Wesley, 2009 (cit. on p. 170).
- [AH11] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist*. Morgan Kaufmann, 2011 (cit. on p. 185).
- [Ale78] C. Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1978 (cit. on p. 20).

## Bibliography

---

- [All08] J. Allspaw. *The Art of Capacity Planning: Scaling Web Resources*. O'Reilly, 2008 (cit. on p. 160).
- [Ara13] J. Araújo. "Semantic Mashups of Linked-USDL Services." MA thesis. University of Coimbra, 2013 (cit. on p. 204).
- [ASL13] V. Andrikopoulos, S. Strauch, and F. Leymann. "Decision Support for Application Migration to the Cloud." In: *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER)*. Citeseer, 2013, pp. 149–155 (cit. on p. 87).
- [BBKL14] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann. "Automating Cloud Application Management Using Management Idioms." In: *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications*. Xpert Publishing Services (XPS), 2014, pp. 60–69 (cit. on p. 170).
- [BBL14] U. Breitenbücher, T. Binz, and F. Leymann. "A Method to Automate Cloud Application Management Patterns." In: *Proceedings of the Eighth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2014)*. Xpert Publishing Services (XPS), 2014, pp. 140–145 (cit. on p. 170).
- [BDA+11] I. Brandic, S. Dustdar, T. Anstett, D. Schumm, F. Leymann, and R. Konrad. "Compliant Cloud Computing (C3)." In: *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*. 2011 (cit. on p. 89).
- [BLS11] T. Binz, F. Leymann, and D. Schumm. "CMotion: A Framework for Migration of Applications into and between Clouds." In: *International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE. 2011, pp. 1–4 (cit. on p. 88).

- [BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture*. Wiley, 1996 (cit. on pp. 20, 40, 43, 96, 101, 118, 166).
- [BN09] P. A. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann, 2009 (cit. on p. 62).
- [Bor01] J. O. Borchers. “A Pattern Approach to Interaction Design.” In: *Ai & Society* 15.4 (2001), pp. 359–376 (cit. on p. 20).
- [Bre12] E. Brewer. “CAP Twelve Years Later: How the “Rules” have Changed.” In: *IEEE Computer Magazine* 45 (2012), pp. 23–28 (cit. on pp. 62, 90, 134).
- [BT14] N. H. Bien and T. D. Thu. “Hierarchical Multi-Tenant Pattern.” In: *Proceedings of the International Conference on Computing, Management and Telecommunications (ComManTel)*. 2014 (cit. on p. 205).
- [BTN+14] A. Bergmayr, J. Troya, P. Neubauer, M. Wimmer, and G. Kappel. “UML-based Cloud Application Modeling with Libraries, Profiles, and Templates.” In: *Proceedings of the International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE) co-located with the International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. 2014 (cit. on pp. 206, 218).
- [CC06] F. Chong and G. Carraro. *Architecture Strategies for Catching the Long Tail*. Technical Report. Microsoft, 2006. URL: <http://msdn.microsoft.com/library/aa479069.aspx> (cit. on p. 92).
- [CD01] J. Cheesman and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001 (cit. on p. 157).

## Bibliography

---

- [CDK05] G. F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Pearson Education, 2005 (cit. on pp. 66, 67, 70, 72).
- [Cha14] L. Charissis. “Design, Implementation and Evaluation of a Rapid-Prototyping Framework for Telematics Services.” Master Thesis. University of Stuttgart, Tilburg University, University of Crete, 2014 (cit. on p. 199).
- [Cod70] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks.” In: *Communications of the ACM* 13 (1970), pp. 377–387 (cit. on pp. 47, 164).
- [Cop06] J. O. Coplien. “Organizational patterns.” In: *Enterprise Information Systems VI*. Springer, 2006, pp. 43–52 (cit. on p. 94).
- [Cop14] J. O. Coplien. *Software Patterns*. 2014. URL: <http://hillside.net/patterns/50-patterns-library/patterns/222-design-pattern-definition> (cit. on pp. 19, 20).
- [Dai11] R. Daigneau. *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. Addison-Wesley, 2011 (cit. on p. 157).
- [Dar14] S. Dara. “Privacy Patterns in Public Clouds.” In: *Proceedings of the Indian Conference on Pattern Languages of Programs (GuruPLoP)*. 2014 (cit. on p. 44).
- [DCE13] B. Di Martino, G. Cretella, and A. Esposito. “Semantic and Agnostic Representation of Cloud Patterns for Cloud Interoperability and Portability.” In: *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*. 2013 (cit. on p. 206).

- [DCES14] B. Di Martino, G. Cretella, A. Esposito, and Sperandeo. “Semantic Representation of Cloud Services: A Case Study for Microsoft Windows Azure.” In: *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems (INCoS)*. 2014 (cit. on p. 206).
- [DiM14] B. Di Martino. “Applications Portability and Services Interoperability among Multiple Clouds.” In: *IEEE Cloud Computing* 1.1 (May 2014), pp. 74–77 (cit. on pp. 20, 206).
- [DKPM07] Y. Diao, A. Keller, S. Parekh, and V. V. Marinov. “Predicting Labor Cost through IT Management Complexity Metrics.” In: *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2007, pp. 274–283 (cit. on p. 190).
- [DMTF14] Distributed Management Task Force, Inc. (DMTF). *Open Virtualization Format Specification*. 2014 (cit. on p. 25).
- [EC09] P. Eeles and P. Cripps. *The Process of Software Architecting*. Addison-Wesley, 2009 (cit. on pp. 156, 195).
- [EFM14] O. Encina, E. B. Fernandez, and R. Monge. “A Misuse Pattern for Denial-of-Service in Federated Inter-Clouds.” In: *Proceedings of the Asian Conference on Pattern Languages of Programs (AsianPLOP)*. 2014 (cit. on pp. 44, 94).
- [EKLR14] V.-P. Eloranta, J. Koskinen, M. Leppnen, and V. Reijonen. *Designing Distributed Control Systems: A Pattern Language Approach*. Wiley Publishing, 2014 (cit. on p. 43).
- [EN10] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 2010 (cit. on pp. 47, 164).

- [EPM13] T. Erl, R. Puttini, and Z. Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 2013 (cit. on pp. 49, 50).
- [FB14] M. M. Falatah and O. A. Batarfi. “Cloud Scalability Considerations.” In: *International Journal of Computer Science & Engineering Survey (IJCES)* 5.4 (2014) (cit. on p. 204).
- [FBBL14] C. Fehling, J. Barzen, U. Breitenbücher, and F. Leymann. “A Process of Pattern Identification, Extraction, and Application.” In: *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLOP)*. 2014 (cit. on pp. 33, 38, 54, 100, 150, 211).
- [FBFL14] C. Fehling, J. Barzen, M. Falkenthal, and F. Leymann. “PatternPedia – Collaborative Pattern Identification and Authoring.” In: *Proceedings of Pursuit of Pattern Languages for Societal Change (PURPLSOC) – Preparatory Workshop*. 2014 (cit. on p. 38).
- [FEL+12] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rutschlin, and D. Schumm. “Capturing Cloud Computing Knowledge and Experience in Patterns.” In: *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*. 2012 (cit. on pp. 38, 43, 178, 203).
- [Fer06] D. Ferrante. “Software Licensing Models: What’s Out There?” In: *IT Professional* (2006) (cit. on p. 91).
- [FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. W3C RFC 2616. 1999. URL: <http://www.w3.org/Protocols/rfc2616/rfc2616.txt> (cit. on p. 76).

- [Fie00] R. T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures.” PhD thesis. University of California, 2000 (cit. on pp. 66, 75).
- [FL14] C. Fehling and F. Leymann. *PatternPedia: A Wiki for Patterns*. Technical Report 2014/03. University of Stuttgart, 2014 (cit. on p. 181).
- [FLM10] C. Fehling, F. Leymann, and R. Mietzner. “A Framework for Optimized Distribution of Tenants in Cloud Applications.” In: *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*. 2010 (cit. on pp. 38, 91).
- [FLMS11] C. Fehling, F. Leymann, R. Mietzner, and W. Schupeck. *A Collection of Patterns for Cloud Types, Cloud Service Models, and Cloud-based Application Architectures*. Technical Report. University of Stuttgart, 2011 (cit. on pp. 43, 102, 104, 121, 123, 126, 127, 130, 132, 133, 135, 137, 139–143, 145, 148, 178, 203).
- [FLR+11] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck. “An Architectural Pattern Language of Cloud-based Applications.” In: *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*. 2011 (cit. on pp. 38, 43, 101, 104, 178, 203).
- [FLR+13] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, and S. Verclas. “Service Migration Patterns.” In: *IEEE International Conference on Service Oriented Computing and Application (SOCA)*. 2013 (cit. on pp. 38, 66, 87, 126, 203, 204, 238, 240).
- [FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud Computing Patterns*. Springer, 2014 (cit. on pp. 18, 21, 23, 27, 38, 40, 43–45, 56, 63, 64, 71, 80, 84, 88, 90,

## Bibliography

---

- 93, 95, 101, 121–123, 127–129, 140, 148, 151, 153, 158, 168, 179, 181, 194, 203).
- [FLR14] C. Fehling, F. Leymann, and R. Retter. “Your Coffee Shop Uses Cloud Computing.” In: *Internet Computing, IEEE* 18.5 (2014), pp. 52–59 (cit. on pp. 23, 38, 154, 203).
- [FLRS12] C. Fehling, F. Leymann, J. Rüttschlin, and D. Schumm. “Pattern-Based Development and Management of Cloud Applications.” In: *Future Internet* 4 (2012), pp. 110–141 (cit. on pp. 38, 203).
- [FLS+11] C. Fehling, F. Leymann, D. Schumm, R. Konrad, R. Mietzner, and M. Pauly. “Flexible Process-based Applications in Hybrid Clouds.” In: *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*. 2011 (cit. on p. 38).
- [FM11] C. Fehling and R. Mietzner. “Composite as a Service: Cloud Application Structures, Provisioning, and Management.” In: *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik* 53.4 (2011), pp. 188–194 (cit. on p. 38).
- [Fow02] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002 (cit. on pp. 20, 97, 157, 162).
- [Fow12] M. Fowler. *NosqlDefinition*. Jan. 2012. URL: <http://martinfowler.com/bliki/NosqlDefinition.html> (cit. on p. 133).
- [FT02] R. T. Fielding and R. N. Taylor. “Principled Design of the Modern Web Architecture.” In: *ACM Transactions on Internet Technology* 2.2 (May 2002), pp. 115–150 (cit. on pp. 66, 75).



- [FTLW14] M. Fleck, J. Troya, P. Langer, and M. Wimmer. “Towards Pattern-Based Optimization of Cloud Applications.” In: *Proceedings of the International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE) co-located with the International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. 2014 (cit. on pp. 206, 218).
- [FYW14] E. B. Fernandez, N. Yoshioka, and H. Washizaki. “Patterns for Cloud Firewalls.” In: *Proceedings of the Asian Conference on Pattern Languages of Programs (AsianPLoP)*. 2014 (cit. on pp. 44, 94).
- [Gar10] P. Garvin. *Carbon Accounting: Beyond The Calculation and Looking To The Future*. Green Economy Post. 2010. URL: <http://greeneconomypost.com/carbon-accounting-7439.htm> (cit. on p. 91).
- [Ger90] German Federal Law. *Federal Data Protection Act (Bundesdatenschutzgesetz, BDSG)*. Dec. 1990. URL: <http://www.iuscomp.org/gla/statutes/BDSG.htm> (cit. on p. 89).
- [GHJ94] E. Gamma, R. Helm, and R. E. Johnson. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994 (cit. on pp. 20, 40, 43, 96, 101, 118, 166, 205).
- [GL02] S. Gilbert and N. Lynch. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services.” In: *SIGACT News* 33.2 (June 2002), pp. 51–59 (cit. on pp. 62, 90, 134, 161).
- [Gol71] R. P. Goldberg. “Virtual Machines: Semantics and Examples.” In: *Proceedings of the IEEE International Computer Society Conference*. 1971 (cit. on p. 60).

## Bibliography

---

- [Gol72] R. P. Goldberg. “Architectural Principles for Virtual Computer Systems.” PhD thesis. Harvard University, 1972 (cit. on p. 60).
- [Gol73] R. P. Goldberg. “Architecture of Virtual Machines.” In: *Proceedings of the Workshop on Virtual Computer Systems*. 1973 (cit. on p. 60).
- [GP13] A. Gambi and C. Pautasso. “RESTful Business Process Management in the Cloud.” In: *ICSE Workshop on Principles of Engineering Service-Oriented Systems (PESOS)*. IEEE, 2013, pp. 1–10 (cit. on p. 204).
- [GR14] I. Gangwar and P. Rana. “Cloud Computing Overview: Services and Features.” In: *International Journal of Innovations & Advancement in Computer Science (IJIAACS)* 3.1 (2014) (cit. on p. 204).
- [GR93] J. Gray and A. Reuter. *Transaction Processing - Concepts and Techniques*. Morgan Kaufmann, 1993 (cit. on pp. 62, 164).
- [Gro12] D. C. M. W. Group. *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol*. Oct. 2012. URL: [http://dmtf.org/sites/default/files/standards/documents/DSP0263\\_1.0.1.pdf](http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf) (cit. on p. 25).
- [GSH+07] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. “A Framework for Native Multi-Tenancy Application Development and Management.” In: *Proceedings of the IEEE International Conference on E-Commerce Technology and the IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*. 2007 (cit. on p. 92).

- [Han07] R. Hanmer. *Patterns for Fault Tolerant Software*. Wiley, 2007 (cit. on pp. 43, 45, 97, 170, 204).
- [Han12] R. Hanmer. “Pattern Mining Patterns.” In: *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*. 2012 (cit. on pp. 41, 56).
- [Han13] R. Hanmer. *Pattern-Oriented Software Architecture For Dummies*. John Wiley & Sons, 2013 (cit. on p. 20).
- [Han14] R. Hanmer. “Patterns for Fault Tolerant Cloud Software.” In: *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*. 2014 (cit. on pp. 45, 170, 204).
- [Har99] N. B. Harrison. “The Language of Shepherding.” In: *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*. 1999 (cit. on pp. 34, 42).
- [HAZ07] N. B. Harrison, P. Avgeriou, and U. Zdun. “Using Patterns to Capture Architectural Decisions.” In: *IEEE Software* 24.4 (2007), pp. 38–45 (cit. on p. 218).
- [HFL12] K. Hashizume, E. B. Fernandez, and M. M. Larrondo-Petrie. “Cloud Service Model Patterns.” In: *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*. 2012 (cit. on p. 44).
- [HHA11] S. M. Hezavehi, U. van Heesch, and P. Avgeriou. “A Pattern Language for Architecture Patterns and Software Technologies.” In: *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP)*. 2011 (cit. on p. 183).
- [Hil14] M. Hilbert. “Architecture for a Cloud-Based Vehicle Telematics Platform.” Diploma Thesis No. 3575. University of Stuttgart, 2014 (cit. on pp. 195, 196).

- [HKLS14] F. Haupt, D. Karastoyanova, F. Leymann, and B. Schroth. “A Model-Driven Approach for REST Compliant Services.” In: *Proceedings of the IEEE International Conference on Web Services (ICWS)*. 2014 (cit. on p. 75).
- [HW03] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003. URL: <http://www.eaipatterns.com/> (cit. on pp. 40, 43, 47, 66, 72–74, 96, 102, 135, 166, 206, 227, 230, 231).
- [HYF11] K. Hashizume, N. Yoshioka, and E. B. Fernandez. “Misuse Patterns for Cloud Computing.” In: *Proceedings of the Asian Conference on Pattern Languages of Programs (AsianPLOP)*. ACM. 2011, p. 12 (cit. on pp. 44, 95).
- [IEE15a] IEEE. *Cloud Profiles Working Group (CPWG)*. 2015. URL: [http://standards.ieee.org/develop/wg/CPWG-2301\\_WG.html](http://standards.ieee.org/develop/wg/CPWG-2301_WG.html) (cit. on p. 25).
- [IEE15b] IEEE. *Intercloud Working Group (ICWG)*. 2015. URL: [http://standards.ieee.org/develop/wg/ICWG-2302\\_WG.html](http://standards.ieee.org/develop/wg/ICWG-2302_WG.html) (cit. on p. 25).
- [JP14] P. Jamshidi and C. Pahl. “Orthogonal Variability Modeling to Support Multi-Cloud Application Configuration.” In: *Proceedings of the SeaClouds Workshop held in conjunction with the European Conference on Service-Oriented and Cloud Computing (ESOCC)*. 2014 (cit. on p. 205).
- [KAP14] D. Kourtesis, J. M. Alvarez-Rodríguez, and I. Paraskakis. “Semantic-Based QoS Management in Cloud Systems: Current Status and Future Challenges.” In: *Future Generation Computer Systems* 32 (2014) (cit. on p. 205).

- [Kas05] M. Kasunic. *Designing an Effective Survey*. Technical Report. Carnegie Mellon Software Engineering Institute, 2005 (cit. on p. 193).
- [KBS05] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA*. Prentice Hall, 2005 (cit. on pp. 66, 78).
- [KSS10] A. Khajeh-Hosseini, I. Sommerville, and I. Sriram. *Research Challenges for Enterprise Cloud Computing*. Technical Report. Cornell University Library, 2010 (cit. on p. 93).
- [LAA+04] D. Lucrédio, E. S. de Almeida, A. Alvaro, V. Cardoso, and E. K. P. Garcia. “Student’s PLoP Guide: A Pattern Family to Guide Computer Science Students during PLoP Conferences.” In: *Proceedings of the SugarLoafPLoP*. 2004 (cit. on pp. 34, 42).
- [LC01] B. Leuf and W. Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional, 2001 (cit. on p. 183).
- [Ley09] F. Leymann. “Cloud Computing: The Next Revolution in IT.” In: *Proceedings of the 52th Photogrammetric Week*. 2009, pp. 3–12 (cit. on pp. 59, 66).
- [LFM+11] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, and S. Dustdar. “Moving Applications to the Cloud: An Approach based on Application Model Enrichment.” In: *International Journal of Cooperative Information Systems* 20.3 (2011), pp. 307–356. DOI: 10.1142/S0218843011002250 (cit. on p. 88).
- [MA01] D. A. Menasce and V. A. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2001 (cit. on pp. 160, 170).

## Bibliography

---

- [MD98] G. Meszaros and J. Doble. “A Pattern Language for Pattern Writing.” In: *Pattern Languages of Program Design 3* (1998), pp. 529–574 (cit. on pp. 20, 41, 102).
- [MG11] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology (NIST), Sept. 2011. URL: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf> (cit. on pp. 22, 57, 69, 121, 123, 125).
- [Mie10] R. Mietzner. “A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing.” PhD thesis. University of Stuttgart, 2010 (cit. on p. 199).
- [MLU09] R. Mietzner, F. Leymann, and T. Unger. “Horizontal and Vertical Combination of Multi-Tenancy Patterns in Service-Oriented Applications.” In: *International IEEE EDOC Enterprise Computing Conference*. 2009 (cit. on p. 141).
- [MM15] V. Mizonov and S. Manheim. *Windows Azure Queues and Windows Azure Service Bus Queues - Compared and Contrasted*. Mar. 2015. URL: <http://msdn.microsoft.com/en-us/library/windowsazure/hh767287.aspx> (cit. on p. 166).
- [MR04] M. L. Manns and L. Rising. *Fearless Change: Patterns for Introducing New Ideas*. Addison-Wesley, 2004 (cit. on p. 94).
- [Neu94] B. C. Neuman. “Scale in Distributed Systems.” In: *Readings in Distributed Computing Systems*. IEEE Computer Society Press, 1994, pp. 463–489 (cit. on pp. 66, 71).

- [NKL+12] A. Nowak, D. Karastoyanova, F. Leymann, A. Rapoport, and D. Schumm. “Flexible Information Design for Business Process Visualizations.” In: *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. 2012 (cit. on p. 92).
- [NL13a] A. Nowak and F. Leymann. *An Overview on Implicit Green Business Process Patterns*. Technical Report 2013/05. University of Stuttgart, 2013 (cit. on p. 91).
- [NL13b] A. Nowak and F. Leymann. “Green Business Process Patterns - Part II.” In: *Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA)*. 2013 (cit. on pp. 92, 216).
- [NLS+11] A. Nowak, F. Leymann, D. Schleicher, D. Schumm, and S. Wagner. “Green Business Process Patterns.” In: *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP)*. 2011 (cit. on pp. 92, 216).
- [Now14] A. Nowak. “Green Business Process Management : Methode und Realisierung.” PhD thesis. University of Stuttgart, 2014 (cit. on pp. 173, 183, 187).
- [OAS12] OASIS. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. Aug. 2012. URL: <http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd04/TOSCA-v1.0-csd04.html> (cit. on p. 25).
- [Pet95] M. Petre. “Why Looking Isn’t Always Seeing: Readership Skills and Graphical Programming.” In: *Communications of the ACM* 38 (1995), pp. 33–44 (cit. on pp. 26, 103, 111, 178, 213).
- [Pri08] D. Pritchett. “BASE: An Acid Alternative.” In: *ACM Queue* 6 (2008), pp. 48–55 (cit. on p. 62).

- [Ram12] R. Ramakrishnan. “CAP and Cloud Data Management.” In: *IEEE Computer Magazine* 45 (2012), pp. 23–28 (cit. on pp. 90, 134).
- [RJKG11] B. P. Rimal, A. Jukan, D. Katsaros, and Y. Goeleven. “Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach.” In: *Journal of Grid Computing* 9.1 (2011), pp. 3–26 (cit. on p. 17).
- [RSM14] P. Reimann, H. Schwarz, and B. Mitschang. “Data Patterns to Alleviate the Design of Scientific Workflows Exemplified by a Bone Simulation.” In: *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*. ACM. 2014, p. 43 (cit. on p. 217).
- [SAB+12] S. Strauch, V. Andrikopoulos, U. Breitenbücher, O. Kopp, and L. Frank. “Non-Functional Data Layer Patterns for Cloud Applications.” In: *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2012 (cit. on p. 90).
- [SAB+13a] S. Strauch, V. Andrikopoulos, T. Bachmann, D. Karastoyanova, S. Passow, and K. Vukojevic-Haupt. “Decision Support for the Migration of the Application Database Layer to the Cloud.” In: *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2013 (cit. on pp. 87, 187).
- [SAB+13b] S. Strauch, V. Andrikopoulos, U. Breitenbücher, S. Gómez Sáez, O. Kopp, and F. Leymann. “Using Patterns to Move the Application Data Layer to the Cloud.” In: *Proceedings of the 5th International Conference on Pervasive Patterns and Applications (PATTERNS)*. 2013 (cit. on p. 90).



- [SAB13] S. Strauch, V. Andrikopoulos, and T. Bachmann. “Migrating Application Data to the Cloud Using Cloud Data Patterns.” In: *Proceedings of the International Conference on Cloud Computing and Service Science (CLOSER)*. 2013 (cit. on p. 162).
- [SAGL13] S. Strauch, V. Andrikopoulos, S. Gómez Sáez, and F. Leymann. “ESB<sup>MT</sup>: A Multi-tenant Aware Enterprise Service Bus.” In: *International Journal of Next-Generation Computing* 3.4 (2013) (cit. on p. 93).
- [SBK+12] S. Strauch, U. Breitenbuecher, O. Kopp, F. Leymann, and T. Unger. “Cloud Data Patterns for Confidentiality.” In: *Proceedings of the International Conference on Cloud Computing and Service Science (CLOSER)*. 2012 (cit. on pp. 162, 216).
- [Sch13] A. Schraitle. “Provisioning of Customizable Pattern-Based Software Artifacts into Cloud Environments.” Diploma Thesis No. 3468. University of Stuttgart, 2013 (cit. on p. 199).
- [Sch15] D. Schumm. “Sichten auf Geschäftsprozesse mit besonderer Betrachtung von Compliance.” PhD thesis. University of Stuttgart, 2015 (cit. on p. 219).
- [SF12] P. J. Sadalage and M. Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, 2012 (cit. on pp. 133, 164).
- [SFH+06] M. Schumacher, E. B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. Wiley, 2006 (cit. on pp. 44, 94, 95, 97).

## Bibliography

---

- [SKS10] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Professional, 2010 (cit. on pp. 47, 164).
- [SPH+12] J. Suzuki, D. H. Phan, M. Higuchi, Y. Yamano, and K. Oba. “Model-Driven Integration for a Service Placement Optimizer in a Sustainable Cloud of Clouds.” In: *Joint International Conference on Soft Computing and Intelligent Systems (SCIS) and International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, 2012, pp. 301–306 (cit. on p. 205).
- [SRD14] G. Sousa, W. Rudametkin, and L. Duchien. “Challenges for Automatic Multi-Cloud Configuration.” In: *JLDP 14-Journée Lignes de Produits* (Dec. 2014) (cit. on p. 205).
- [SSBM11] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. “Native Support of Multi-tenancy in RDBMS for Software as a Service.” In: *EDBT*. ACM, Jan. 2011, pp. 117–128 (cit. on p. 93).
- [TD15] H.-L. Truong and S. Dustdar. “Programming Elasticity in the Cloud.” In: *Computer* 48.3 (2015), pp. 87–90 (cit. on p. 216).
- [Tiw11] S. Tiwari. *Professional NoSQL*. Wrox, 2011 (cit. on p. 164).
- [TS06] A. S. Tanenbaum and M. van Steen. *Distributed Systems Principles and Paradigms Second Edition*. Prentice Hall, 2006 (cit. on pp. 66, 67, 69, 70, 90, 163, 164).
- [Vai14] M. Vainikka. “Migrating Legacy Applications to Cloud: Case TOAS.” MA thesis. Lappeenranta University of Technology, 2014 (cit. on p. 204).

- [Var08] J. Varia. *Cloud Architectures*. Technical Report. Amazon Web Services, 2008 (cit. on pp. 161, 166).
- [Var10a] J. Varia. *Architecting for the Cloud: Best Practices*. Technical Report. Amazon Web Services, 2010 (cit. on pp. 161, 166).
- [Var10b] J. Varia. *Migrating Your Existing Applications to the Cloud – A Phase-Driven Approach to Cloud Migration*. Technical Report. Amazon Web Services, 2010 (cit. on p. 166).
- [Vog09] W. Vogels. “Eventually Consistent.” In: *Communications of the ACM* 52 (2009), pp. 40–44 (cit. on p. 62).
- [WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, Apr. 2005 (cit. on pp. 21, 78, 156).
- [WF11] T. Wellhausen and A. Fießer. “How to Write a Pattern.” In: *European Conference on Pattern Languages of Programs (EuroPLOP)*. Vol. 11. 2011 (cit. on pp. 41, 102).
- [Wie14] R. J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014 (cit. on p. 194).
- [ZA08] U. Zdun and P. Avgeriou. “A Catalog of Architectural Primitives for Modeling Architectural Patterns.” In: *Information and Software Technology* 50.9 (2008), pp. 1003–1034 (cit. on p. 113).
- [Zdu07] U. Zdun. “Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis.” In: *Software: Practice and Experience* 37.9 (2007), pp. 983–1016 (cit. on p. 20).

## Bibliography

---

- [Zim09] O. Zimmermann. “An Architectural Decision Modeling Framework for Service-Oriented Architecture Design.” PhD thesis. University of Stuttgart, 2009 (cit. on pp. 78, 218).

All links were last followed on May 31, 2015.

---

## Acknowledgments (Danksagungen)

---

Abschließend möchte ich bei den Menschen bedanken, die mich während der Erstellung dieser Arbeit begleitet haben. Im Besonderen gilt mein Dank meiner Familie und Freunden, die mich in dieser Zeit stets unterstützt haben.

Meinem Doktorvater Prof. Dr. Dr. h. c. Frank Leymann danke ich sehr für die außerordentliche fachliche und persönliche Betreuung während meiner Promotion. Auch das vorangegangene Buchprojekt wäre ohne seinen Einfluss und fortwährende Motivation nicht zustande gekommen. Meinem Zweitgutachter Univ.Prof. Dr. Schahram Dustdar danke ich sehr für die freundliche Unterstützung meiner Promotion und den anregenden Gedankenaustausch bei meinen Besuchen in Wien.

## Acknowledgments (Danksagungen)

---

Bei meinen Kollegen vom Institut für Architektur von Anwendungssystemen möchte ich mich für die Zusammenarbeit bedanken. Für die vielen Diskussionen und Ratschläge danke ich insbesondere Ralph Retter, David Schumm, Olaf Zimmermann, Daniel Schleicher und Oliver Kopp. Weiterhin möchte ich Ulrike Ritzmann für die organisatorische Unterstützung danken.

Für den fachlichen Austausch und die Zusammenarbeit möchte ich mich auch bei Partnern aus der Industrie bedanken. Insbesondere gilt dieser Dank Walter Schupeck, Stefan T. Ruehl, Peter Arbitter, Jochen Rütschlin, Jens Nahm, Marc Rudek und Uli Held.

Zu guter Letzt danke ich den Studenten, die meine Forschung im Rahmen Ihrer Diplom-, Bachelor- und Masterarbeiten unterstützt haben.