Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

Pattern-based solution for architecting cloud-enabled software

Jalawi Sulaiman Alshudukhi *

College of Computer Science and Engineering, University of Ha'il, Ha'il, Saudi Arabia

ARTICLE INFO

Article history:
Received 2 February 2021
Received in revised form
16 April 2021
Accepted 27 April 2021

Keywords:
Cloud architectures
Patterns and tactics
Pattern catalog

ABSTRACT

Cloud computing exploits the software as a service model with distributed and interoperable services for the composition of software systems. Cloud-enabled systems that demand elasticity, scalability, and composition of services, etc., there is a need to capitalize on reusable solutions exploiting patterns and styles to architect cloud-based software. The objective of this research is to build and exploit a catalog of patterns that support reusable design knowledge to develop cloud-based architectures. We propose a three-step process with (i) pattern discovery, (ii) pattern documentation (building the catalog), and finally, (iii) pattern application (exploiting the catalog) to enable pattern-based architecting of cloud systems. We discovered seven patterns as generic and reusable solutions and demonstrate the pattern-driven architecture of the ECMC case study. Results suggest that pattern-based architecting enables the reuse of generic design decisions but lacks fine-grained architectural design. The solution is the first attempt towards establishing the catalog as a repository of patterns for architecture-based development of cloud systems.

1. Introduction

Cloud computing has gained widespread adoption in all sorts of business entities, public as well as private. Apart from the potential business benefits of a pay-per-use model as opposed to upfront investment and set up for IT infrastructures (Jamshidi et al., 2013a). There are clearly observable technical advantages of cloud computing compared with many other models of IT provisionings such as scalability, multi-tenancy, resource virtualization, and runtime acquisition of computing resources (Herbst et al., 2013). Rapid demand in designing and/or evolving applications for cloud-based infrastructures requires a number of highly knowledgeable and experienced architects who may not be widely available as cloud computing is an innovative paradigm (Wilder, 2012; Ahmad et al., 2012a).

Architectural styles and patterns proved a successful mechanism for providing packaged knowledge about well-known design solutions to both experienced and novice architects (Buschmann et al., 2007; Ahmad et al., 2014a). Patterns document frequent solutions to recurring problems in a given domain (Harrison et al., 2007; Cámara et al., 2013; Côté et al., 2007). In recent years, pattern-based approaches resulted in (i) promoting reuse and (ii) enhancing the efficiency of the architectural design and evolution processes (Ahmad et al., 2014a; Cámara et al., 2013). In addition, pattern-oriented solutions (Buschmann et al., 2007; Rischbeck and Erl, 2009) enhance quality (extensibility, coupling, etc.) by applying the best practices and knowledge to resolve recurring problems of architectural design.

Ahmad et al. (2014b) asserted that providing architectural patterns for cloud-based software can accelerate the process of gaining knowledge and experience in successfully modeling and evolving the system's structure and behavior at higher abstractions. We focus on providing a collection of architectural patterns that promote the reuse of design knowledge for architecting cloud-based systems. We represent patterns as a collection of reusable design (service components and connectors) to determine (i) what is the architectural composition and a set of constraints (configuration of elements) to answer how architectural composition is achieved (Buschmann et al., 2007; Jamshidi et al., 2013b). While architecting cloud-based systems one exploits dynamically composed services (software-as-a-service: SaaS) for systems that can dynamically reconfigure them as per changes in systems operational environments

* Corresponding Author.
Email Address: j.alshudukhi@uoh.edu.sa
https://doi.org/10.21833/ijaas.2021.08.002
Corresponding author's ORCID profile:
https://orcid.org/0000-0003-0619-0020
2313-626X/© 2021 The Authors. Published by IASE.

(Herbst et al., 2013; Wilder, 2012). As opposed to traditional IT systems (Côté et al., 2007), patterns for cloud architectures enforce specific requirements including; composition, elasticity, scalability, and multi-tenancy of soft-ware services (Arcitura, 2014; Ahmad et al., 2012a).

One of the key challenges in providing pattern-based architectural knowledge is systematic discovery and detailed documentation of patterns as a generic, yet reusable solution to most frequently occurring architectural solutions (Ahmad et al., 2014a; Harrison et al., 2007). Whilst a pattern language of cloud-based application (Jamshidi et al., 2013b) and online catalog of patterns for cloud application (Arcitura, 2014; CDP, 2014) have been reported, there has been no attempt to systematically discover and document architectural patterns for cloud-based applications. We enable a systematic pattern discovery by investigating the recurring problems and their generic, repeatable solutions in existing architectures for cloud systems (Ahmad et al., 2014b). In this paper, we report our empirical effort for building a catalog of architectural patterns for cloud-based applications; and demonstrate how the discovered architectural patterns can be applied to design a cloud-based application. Our approach consisted of three simple steps including Pattern discovery, pattern documentation, and pattern application. With regards to the existing research in Wilder (2012), Arcitura (2014), Jamshidi et al. (2013b), and CDP (2014), our contributions are:

- Investigating sources to empirically discover patterns that address cloud architecture requirements such as scalability, elasticity, multi-tenancy, etc. in the SaaS model.
- Exploit the discovered patterns as elements of reuse knowledge that guides a step-wise process of pattern-based architecting for cloud systems.

2. Related work on architecture patterns for cloud computing

We aim to discuss the rationale for the proposed solution in the context of existing research on (i) patterns for cloud and SOA systems, and (ii) pattern discovery and pattern application.

2.1. Architecture patterns for cloud systems

One of the thorough works on cloud architecture patterns (Wilder, 2012) reports best practices for scalability, big data, fault handling, and distributed services on Windows Azure (Platform as a Service: PaaS). Wilder (2012) provided guidelines and practical solutions to address the scalability and elasticity in cloud-native applications for the Windows Azure platform. However, this research has two limitations:

1. Lack of empirical discovery patterns has been reported based on the expertise from a single source (pattern author) without investigating multiple sources or systems. This means the patterns may not be generally applicable to different solutions (Harrison et al., 2007).
2. Lack of generic solution patterns are specific to problems in a specific domain (i.e., Windows Azure) and their reuse across different domains is not explicit (Buschmann et al., 2007).

The work reported in Jamshidi et al. (2013b) provides a documented repository of patterns for the development of cloud computing services (SaaS, PaaS, IaaS), and their deployment using cloud deployment models (private, public, hybrid, community). The patterns are organized in a framework to guide developers to systematically select and apply these patterns.

In contrast to Wilder (2012) and Jamshidi et al. (2013b), the patterns in our catalog are focused on deployment, execution, and management aspects of cloud services (focusing SaaS model). With regards to a fixed number of patterns in Wilder (2012), our work aims to establish a pattern catalog that would evolve with the incremental discovery of new patterns guided by Ahmad et al. (2012b).

2.2. Repositories of cloud computing patterns

The work in Arcitura (2014) and CDP (2014) reported a community-driven development of pattern including:

1. Arcitura (2014) presented a collection of 39 patterns to address the scalability, reliability, security, and monitoring issues of cloud applications.
2. CDP (2014) presented a collection of patterns created by various (cloud) architects based on the type of problems, and their generic design patterns.

The patterns in Arcitura (2014), and CDP (2014) are not specific to cloud-based architectures, instead, they focus on issues like security and monitoring of cloud computing applications. Also, the repository of patterns in Ahmad et al. (2019) focused on addressing the scalability, reliability, security, and monitoring of the cloud computing infrastructures (IaaS). In contrast to Ahmad et al. (2019) (IaaS: addressing cloud infrastructure), and Wilder (2012) (PaaS: addressing cloud platform) the patterns presented in this research primarily focus on (i) SaaS (addressing cloud services) and (ii) exploit architectural abstraction to develop cloud systems deployed on PaaS.

2.3. Architecture pattern mining and pattern application

The existing research to empirically discover patterns can be broadly categorized into History Analysis vs Design Review methods. In contrast to history-based pattern mining (Ahmad et al., 2012b), we review the architectural design (Ahmad et al., 2014b) to discover patterns. Patterns for software architecture in Buschmann et al. (2007) represented one of the earlier solutions aimed at proposing a system-of-patterns to design software architectures. Since then, the research on patterns for architectural development and evolution has progressed over more than a decade (Ahmad et al., 2014a). Based on the research overview above and the findings of our systematic reviews (Jamshidi et al., 2013a; Ahmad et al., 2014a), we claim that the proposed solution is the first attempt to establish a catalog for pattern-driven, architecture-based development of cloud systems.

2.4. Patterns for cloud-based architectures

We clarify (i) how cloud architectures are distinguished from the rest and (ii) what are the (quality) characteristics to be addressed by these patterns with an example.

2.5. Characteristics of cloud-based architectures

By utilizing the SaaS model, cloud architectures (Ahmad et al., 2012a) can exploit the principle of service orientation (specifically SOAs (Rischbeck and Erl, 2009) that enables service composition as a foundation to develop cloud-based applications (Ahmad and Babar, 2014a). Also central to cloud architectures are the quality of service (QoS) requirements that ensures composable services must satisfy the desired quality characteristics. These characteristics include but are not limited to scalability, elasticity, multi-tenancy, and virtualization of software services that distinguish the cloud architectures (Ahmad et al., 2012a) from the traditional (object and component) ones (Ahmad et al., 2014a; Côté et al., 2007). Therefore, the patterns for traditional software development (Côté et al., 2007) cannot easily be applied to cloud-based systems unless they support the above-mentioned characteristics specific to cloud architectures. A single pattern may not ensure all these characteristics; however, the pattern collection must try to address them all.

For example, unlike traditional architectures, cloud-based architectures are supposed to serve multiple tenants with each tenant having its own specific QoS requirement that can vary from performance and reliability to security aspects. Multi-tenant capabilities of SaaS need to be considered not only at service (Arcitura, 2014) but also at the platform (Wilder, 2012) and infrastructure (Ahmad et al., 2019) level not addressed in existing SOA patterns (Rischbeck and Erl, 2009).

2.6. Pattern abstraction and pattern instantiation

We use an example of one of the discovered patterns named Service Interoperability to distinguish abstraction and instantiation in Fig. 1. We have also reported details about an individual pattern (Tools as a Service (TaaS)) and our experiences in Ahmad and Babar (2014a).

A. Abstraction for Pattern Modeling: Abstraction is vital to promote a pattern as a generic solution by abstracting the complex implementation-specific details as in Fig. 1a. As illustrated in Fig. 1a, the abstraction of the Service Interoperability pattern is vital to help a pattern user (de-signer/architect) to analyze the high-level solution for binding services to available interfaces. Pattern abstraction is vital to analyze its impact on the architecture model before pattern application (preconditions), the architectural view when the pattern is applied (post-conditions)–promoting pattern as a generic and incremental design process.
B. Instantiation for Pattern Application: Instantiation provides the necessary details in terms of concrete architectural elements to instantiate a pattern. This is also referred to as a pattern application by means of adding refinements–extending the abstract box and arrows with architectural components and connectors from Fig. 1a to pattern abstraction in Fig. 1b (Zdun, 2007). In Fig. 1b, the instance of the Service Interoperability pattern utilizes the Service Bus to bind services in the Service Pool to interfaces in the Interface Compatibility component.

3. Research methodology and proposed solution

We now present an overview of the methodology and then discuss a three-step approach to discover documents, and apply architecture patterns.

3.1. Methodology for pattern discovery and documentation

Pattern discovery is based on the design review method (Chen, 1998) reviewing recurring de-sign solutions to frequent problems of architecting cloud systems (Ahmad et al., 2012a) (in 4.1.A-3.1.C). The review team comprises of 3 members with experience of (a) conducting the SLRs (Jamshidi et al., 2013a; Ahmad et al., 2014a), (b) pattern mining (Ahmad et al., 2012b), and (c) development of cloud systems (Babar and Chauhan, 2011; Ahmad and Babar, 2014a).

A. Systematic Review of Architecture Solutions for Cloud Systems: The review was conducted to investigate the recurring challenges, design problems, and existing solutions to develop cloud architectures. A systematic review (Jamshidi et al., 2013a) is expected to minimize the potential bias in the review and has a protocol that guides the process. Based on the research questions (RQs) below and the protocol in Ahmad et al. (2014b), we selected 86 studies (problem-solution mapping) as sources of pattern discovery.

RQ1–What methods/ techniques/ frameworks/ solutions are provided in existing (research and practices) to model/develop/evolve cloud system architectures?

RQ2–What are the existing patterns/ styles/ frameworks to support reusable design knowledge for architecting cloud-based systems?

Service Interoperability Pattern

Pattern Intent: "interoperability between a collection of services (S_1,... , S_N) in a service pool with compatible interfaces (C_1, ..., C_N) acquired at run time".
Design Problem: How to achieve interoperability among services in a service pool?
Solution: Provide a mediator to bind services to their compatible interfaces (Fig. 2)
Architecture Elements: Service Pool and Interface Compatibility components interconnected using a Service Bus connector.
Constraints: specify (i) a one-to-one correspondence between services and interfaces, and (ii) a connector (service bus) that to mediate the binding. Reuse design knowledge is expressed as loosely coupled components bound by integrating mediators.
Quality Characteristics–e.g., Service Composition and Elasticity
-Elasticity achieved by providing a pool of services independent of their interfaces. This allows a dynamic acquisition of new services or releasing unutilized ones by delegating the interface binding and unbinding to a mediator.
-Composition is supported through a mediator to enable service orchestration. The pattern assumes that the required interfaces are provided in Compatible Interface component to ensure service availability.

Reference Diagram

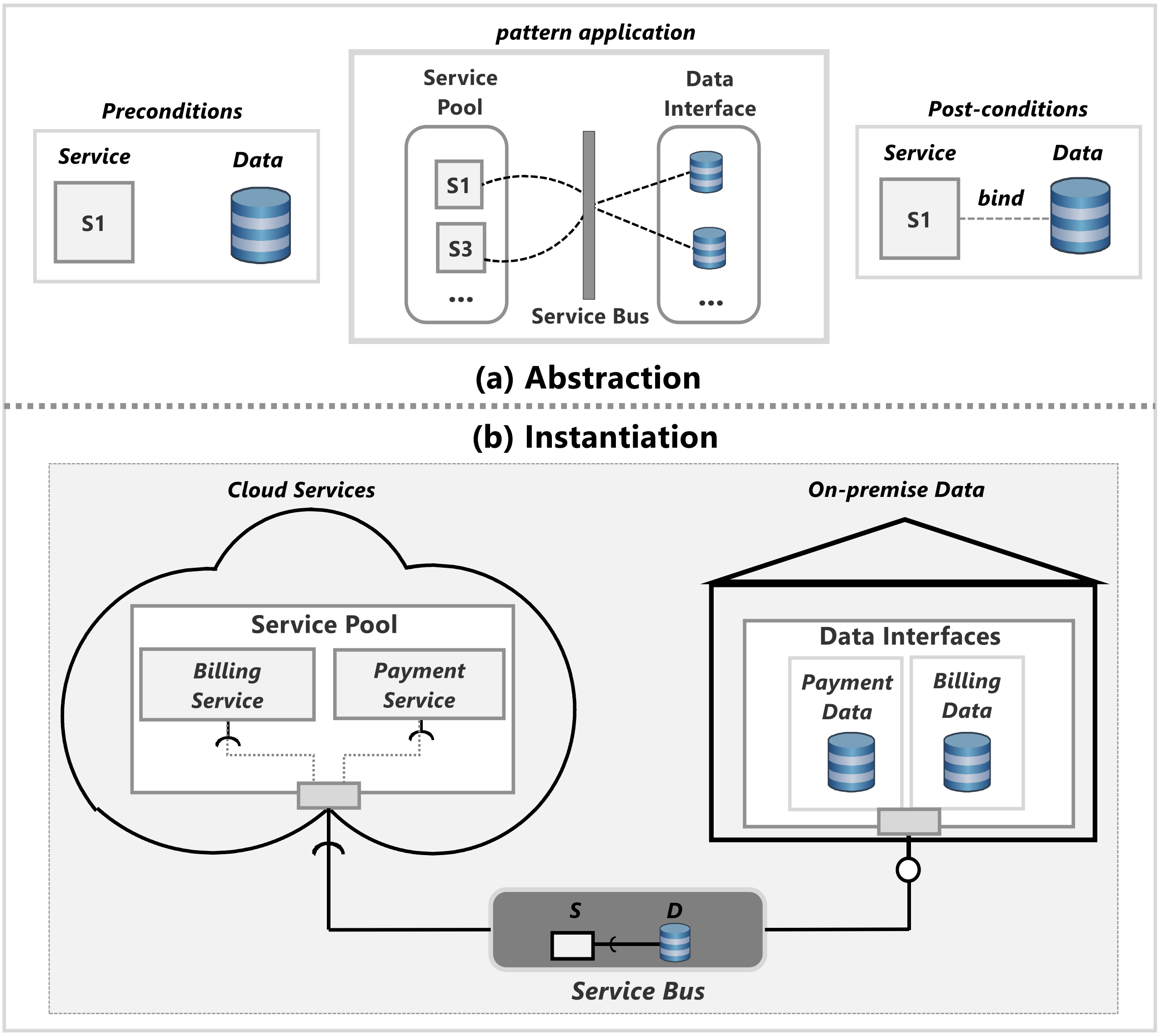


Fig. 1: Service interoperability pattern (abstraction vs instantiation)

B. Identification of Pattern Data Sets–Once the studies were identified, we extracted the data sets in Table 1 from selected studies. Datasets refer to mapping the existing architectural design problems and their solutions. For example, in the case of Service Interoperability pattern (Fig. 1), design problem underlines service interoperability, while architectural solution proposes interface binding by using a mediator.

For an objective evaluation, we derived 7 items (I1 to I7–item collection is referred to as datasets) based on the recommendations and guidelines in Buschmann et al. (2007) and Harrison et al. (2007) and classification of architectural styles and patterns (Ahmad et al., 2014a). Items in Table 1 guided the pattern mining team to objectively review the problem (P) and solution (S) mapping, the attributes (A) that affect the solution and the occurrence frequency (T) of the repeatable solution by analyzing the pattern datasets. Once a decision (D) is reached, the results are documented as pattern elements (E) for a peer-review before finalization. We have also published individual patterns (Ahmad and Babar, 2014a) to seeking preliminary feedback on our ongoing work from software architecture community.

Table 1: Dataset items for pattern discovery process

ID	Items	Description
I1	Design Space (C)	All the available architectural designs (C=86 studies).
I2	Recurring Problem (P)	Repeatable problems existing in the design space (P∈ C).
I3	Frequent Solution (S)	Solutions to repeatable problems in the design space (S∈ C).
I4	Frequency Threshold (T)	Threshold for occurrence of S to be discovered as a pattern.
I5	Design Attributes (A)	Attributes affecting S (A=15) Service Level Agreements, Quality of Service, Service Elasticity, Service Composition, Context Awareness, Service Versioning, Service Deployment, Service Execution, Service Management, Service Security, Service Reliability, Service Availability, Service Coupling, Service Interoperability, Data Storage
I6	Discovered Pattern (N)	2=Yes, 1=Not sure (consensus required), 0=No
I7	Pattern Elements (E)	Elements of Pattern Description (E=9) Name, Intent, Problem (P), Solution (S), Impact, Origin, Uses, Reference Diagram, Architecture Elements, Constraints

C. Thematic Analysis to Investigate Pattern Datasets: After identification of the datasets, thematic analysis as the final step helps to 'identify, analyze and report' patterns from datasets by following three steps. A theme is a possible solution, or method, or mechanism to resolve problems (Ahmad et al., 2019).

1. Data Analysis process comprises (a) analyzing datasets, (b) extracting the design attributes from problem-solution mapping (I5 in Table 1).
2. Pattern Discovery process involves (a) searching of the recurring themes based on data analysis, and (b) reviewing the identified themes. To discover patterns, we reviewed studies and aimed at discovering design problems (I2) and they relate solutions (I3). We consider a recurring theme as a discovered pattern (I6).
3. Pattern Documentation is the last process that includes (a) classification of related themes based on design attributes (I5) and documented them in a template (I7).

3.2. Solution overview for pattern-based architecting

We propose pattern-based architecting as a 3-step process with underlying activities and repositories in Table 2. Pattern discovery involves pattern mining and pattern modeling. Pattern documentation involves pattern classification. Pattern application involves selection and instantiation. If a designer finds suitable patterns from the catalog, then the first two steps are skipped.

Table 2: Processes, activities, repositories for pattern-based architecting

Processes	Activities	Repositories
Pattern Discovery	Pattern Mining Pattern Modeling	Pattern Source Artifacts of data sets that contain patterns in them.
Pattern Documentation	Pattern Classification Pattern Specification	
Pattern Application	Pattern Selection Pattern Instantiation	Pattern Catalogue Repository to store and retrieve patterns to enable reuse.

3.3. A metamodel of the discovered patterns

Based on the methodological description (Table 1) and applied solution (Table 2), we discovered a total of seven patterns. However, before presenting the discovered patterns, we provide a metamodel as a formalized foundation to model architecture patterns. The metamodel express pattern-based architecting as a 5-tuple PatArch:=<ARCH, OPR, CNS, PAT, CAT> in Fig. 2. For space reasons we only present a minimal model with extended metamodel provided in Ahmad et al. (2014b).

- Metamodel represents a structural model and its required elements and the relationships between them (e.g.; Pattern is Composed of one or many Operations).
- A formalization of pattern model to ensure all pattern instances conform to the same abstraction (meta-model).
- Also, a formal representation of the metamodel elements facilitates (semi-) automation and tool-based manipulation of the pattern descriptions.

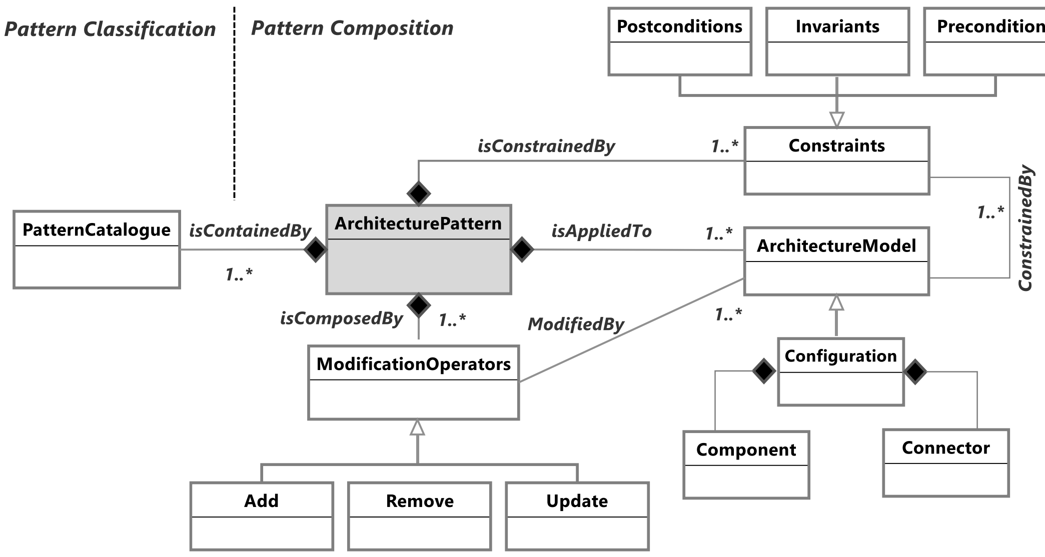


Fig 2: A metamodel representation of architecture pattern

A. Specification of the Cloud Architecture Model (ARCH): The architectural description is driven by the Service Component Architecture (SCA) specifications (Ahmad et al., 2018) with components that expose services in the SaaS model.

Example: In the Service Interoperability pattern (Fig. 1), the Service Pool is a component composed of atomic services (Billing Service, Payment Service). The connector is represented as a Service Bus to enable component-level interconnection.

B. Specifying the Constraints on Architecture (CNS): The constraints on the architecture model are Preconditions (conditions before pattern is being applied) and Postconditions (conditions after pattern application) to ensure consistency of pattern-based architecture composition. Invariants restrict the number of components/connectors in a pattern.

Example: The preconditions for Service Interoperability pattern specify the existence of services and their interfaces, while the post-conditions represent a binding between services and interfaces. Invariants ensure each service is bound to one interface only.

C. Specifying the Modification Operators (OPR): Operators parameterize the addition or removal of architecture elements to modify the architecture model (design by modification (Boyatzis, 1998)). These consist of Add, Remove, and Update operators on architecture elements.

Example: For example, the operators in the Service Interoperability pattern enable the addition or removal of elements (service components and their interfaces). The addition operator enables the introduction of Service Bus connectors (modifications from preconditions to postconditions) in order to bind components.

D. Specifying the Architecture Pattern (PAT): Architectural patterns represent the abstraction for reusable architecting of the system. Pattern is represented via its name and has an intent, constrained composition of operationalization on architecture model, as pattern composition in Fig. 2.

Example: The Service Interoperability pattern helps to develop interoperable services. It specifies the constraints on architecture such that services must be isolated from interfaces and service binding added at runtime with service bus (Fig. 1).

E. A Catalogue of Architecture Patterns (CAT): A pattern catalog is a collection of classified patterns ready for selection and reuse (catalog can be supported by a repository management system or a manual system). The guidelines in Harrison et al. (2007) can be followed to develop and a pattern template. With the UML metamodel in Fig. 2, we express the concrete syntax of patterns in the catalog with eXtensible Markup Language (XML). XML-based specification of patterns allows us to (i) customize the pattern elements and preserve the pattern hierarchy, (ii) extensibility with addition or removal of pattern (metamodel) elements (when required), and (iii) machine readability for sharing pattern specifications.

4. Architecture pattern catalog

Once the patterns are discovered and modeled, the next step involves classifying the patterns and documenting them in a catalog in Table 3.

4.1. A taxonomical classification of discovered patterns

The pattern taxonomy defines a systematic mapping, naming, and organization of related patterns. The attributes of classification are extracted by analyzing the design attributes (I5, cf. Table 1). For example, analyzing the known uses of the Service Interoperability pattern has helped us to identify the "Service Coupling" and "Service Elasticity" characteristics associated with this pattern. The benefits of classification include:

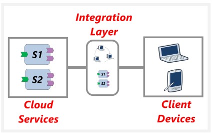
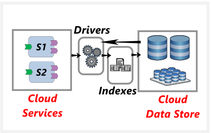
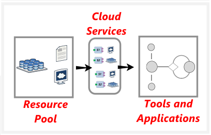
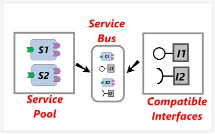
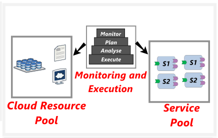
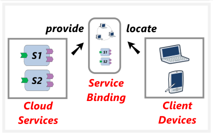
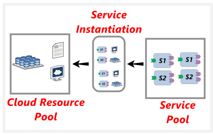
- Pattern Selection from existing repositories a critical challenge (Zdun, 2007). A pattern classification helps in reducing the search space where patterns that share the same attributes can be located easily. For example, based on the item (I5, cf. Table 1) the Service Interoperability and Service Watchdog patterns (Table 3) ensure the QoS requirements (reliability, scalability, etc.) are classified under "service execution".
- Efficient Searching of patterns can be achieved (using classification as indexing) to reduce the time taken to search pattern catalog. A (semi-) automated searching and retrieval of patterns from the catalog represents part of the planned future work.

We have classified patterns into three distinct types including Service Deployment, Execution, and Management. The name of classification type is subjective as classification is a manual process-based on design reviews by the team.

4.2. A template-based specification of architecture patterns

Table 3 shows a precise view of the pattern catalog that is derived based on the guidelines from Ahmad et al. (2014b).

Table 3: A catalog of architecture patterns for cloud-based software

Pattern Name	Intent	Reference Diagram
End-to-end Service Binding	Design Problem: How to provide an end-to-end binding between requests of client devices and available services in a service pool? Solution: Provide a service integration layer as a mediator that maps the available services in the pool to the requester client devices.	 Integration Layer S1 S2 Cloud Services Client Devices
Multi-tenant Access to Databases	Design Problem: How to enable multi-tenant access to databases deployed in the cloud? Solution: Allocate a dedicated service to each tenant request, responsible for the data access (handling db drivers and indexes) specific to each tenant.	 Drivers S1 S2 Indexes Cloud Services Cloud Data Store
Tool as a Service (TaaS)	Design Problem: How to enable the provisioning of software tools and applications as a cloud service? Solution: Provide a layered architecture with cloud services (Layer 2) mediating between the available tools (layer 1) and client requests (layer 3).	
Services Interoperability	Design Problem: How to achieve interoperability among services in a service pool? Solution: Provide a mediator that binds services in the pool to their java compatible interfaces at runtime (related pattern: *Service Request Handling*).	 Service Bus S1 S2 I1 I2 Service Pool Compatible Interfaces
Service Watchdog	Design Problem: How to continuously monitor the quality of services in a pool? Solution: Provide a service monitor using MAPE model when a number of service providers use a pool – monitoring of SLAs and QoS requirements before service execution.	 Monitor Plan Analyse Execute Monitoring and Execution S1 S2 Cloud Resource Pool Service Pool
Service Request Handling	Design Problem: How to provide a shared pool to providers (producing services) and the requesters (consuming services)? Solution: Provide service binding layers to handling clients´ requests by selecting the provider services (related pattern: *Service Interoperability*).	 provide locate S1 S2 Service Binding Cloud Services Client Devices
Services Unit	Design Problem: How to service composition in a service pool to satisfy multi- client requests? Solution: Provide a service unit (composition of atomic services into composites) - the cooperating services act as a single unit to external clients (based on Service Orchestration).	 Service Instantiation S1 S2 Cloud Resource Pool Service Pool

4.3. Architecture pattern application

Finally, we demonstrate how to utilize a specific pattern from a catalog and clarify a few concepts relevant to patterns and pattern application.

A. Application Domain of the Patterns: We specify patterns as a mapping between generic solution(s) to recurring problem(s) in a specific domain. The application domain of the proposed patterns is the SaaS model for cloud-based architectures (Ahmad et al., 2014b).

B. Runtime Change Support: As a consequence of ever-changing requirements, architectural design undergoes frequent modifications (a.k.a design by modification (Côté et al., 2007)). In cloud architectures, modifications must be implemented as runtime adaptations to support dynamic: (1) service composition, and (2) architectural elasticity for acquisition and release of services. We abstract the time implications, as details about pattern-based runtime modifications are provided in Ahmad and Babar (2014b) guided by the IBM MAPE loop (Ganek and Corbi, 2003).

C. Primitive vs Pattern-based Modifications: A primitive in an architectural modification refers to the most fundamental operation applied to the architecture elements (add, remove or update architectural elements) in the metamodel (Fig. 2). In contrast, patterns abstract the primitives and provide higher-level reusable operations such as integration, composition, replacement of architectural elements (Ahmad et al., 2014a; 2012b). The pattern represents a process-driven approach in terms of analyzing the architecture (1) before (preconditions: Source), (2) during (invariants: intermediate), and (3) after pattern application (preconditions: Target).

D. Scenario-based Analysis of Architecture Modification: In order to systematically identify and analyze the design scenarios and required architectural modifications, we utilize the Architecture Level Modifiability Analysis (ALMA) (Bengtsson et al., 2004) as a 5-step process to Step 1–Analyze design and deployment scenario(s), Step 2–Model software architecture, Step 3–Select scenario(s), Step 4–Evaluate scenario(s), and finally Step 5–Interpret the results of architectural modification.

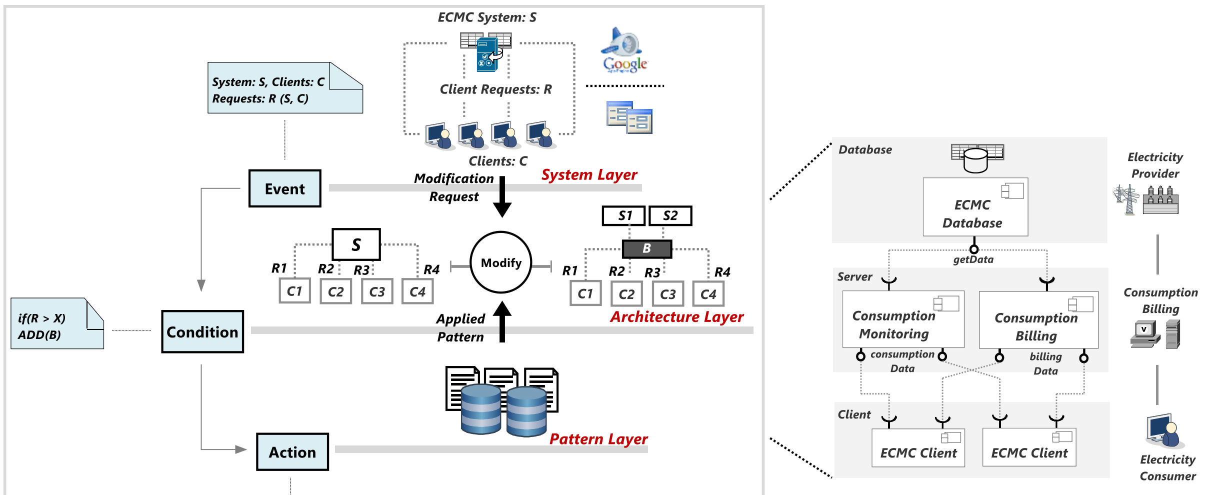
5. Analyze architecture scenario and cloud environment

A. Cloud Architecture Case Studies: The evaluation of pattern-based architecture development is performed using case studies of (i) Electricity Consumption Monitoring and Contributing (ECMC) system, and an online (ii) Auction Management System (AMS) system. For illustrative purposes, we utilize the architectural view of ECMC System (Ahmad and Babar, 2014b; Bengtsson et al., 2004) as illustrated in Fig. 3a. The ECMC system is offered by a Danish electricity provider as an online portal to its customers (electricity consumers) with two purposes: (i) view electricity consumption, and (ii) pay electricity bills. Based on increasing client requests to ECMC architecture, an elastic load balancer (B) needs to be integrated between the clients (C1, C2, C3, C4, ..., CN) and the ECMC server (S) that can acquire or release services based on the number of client requests (R1, R2, R3, R4, ..., RN). In Fig. 3a, ECMC client requests (system layer) trigger the modifications in the architecture model (architecture layers), patterns provide reuse of modifications (pattern layer).

B. Cloud Computing Platform for Service Execution: We have chosen the Google App Engine (GAE) to deploy and execute the services (SaaS model) of ECMC. For space reasons, we abstract technical details of cloud platform that can be found in Babar and Chauhan (2011) and Ahmad and Babar (2014a). The selection of GAE was primarily influenced by the needs for auto-scaling, resource allocation, and service implementation framework (Java) for ECMC.

6. Architectural description of ECMC system

ECMC has a layered architecture with components and a database, where the client can monitor or pay the bill for their electricity consumption. For example, a typical usage scenario is view consumption monitoring: After authentication, the electricity consumer (ECMCClient1) sends a request (consumption Data) to the Consumption Monitoring component in order to view the consumption details. The monitoring component queries the ECMC Database to retrieve consumption data (getData) in Fig. 3b. The architectural view represents the source architecture (SCA model (Ahmad et al., 2018).



b) Overview of the ECMC Architecture Model

a) Overview of the ECMC System with Architecture and Pattern Layer

Fig. 3: (a) Overview of pattern application, (b) Architectural view of ECMC

6.1. Select scenario–load balancing of requests

In Fig. 3b, ECMC Clients are integrated with Consumption Monitoring and Consumption Billing components resulting in a tight coupling between the clients and server that affects load balancing of the server. The optimal performances (P) of server response time in relation to an increased client requests (R) is given as $\boldsymbol{P} = \frac{\boldsymbol{R}}{\boldsymbol{N}}$. N (elasticity threshold) is the maximum number of a client request that do not hinder P. If client requests at a given time exceed a certain threshold (X), such that if: $R > X$, then server response time (T) deteriorates. The notion of this scenario is given in Fig. 3 (architecture layer) as exceeding clients affects server performance.

6.2. Evaluate scenario–integrating the load balancer

The scenario in Fig. 4 illustrates that ECMC architecture in Fig. 3 requires an elastic load balancer to mediate between the clients and server components that handle client requests. More specifically:

- if the client requests (R) exceeds a certain frequency threshold (X), then a new RequestBroker (B) is added, or alternatively.
- if the client requests a return to the below threshold, then the B must be removed.

The scenario is evaluated based on the Event Condition Action (ECA) formalism in Fig. 3a and Table 4. Details about the ECA-based formalism to trigger architectural modifications are discussed in Ahmad and Babar (2014b). The addition or removal of architectural elements is achieved with modification operators (Fig. 2).

Table 4: Event condition action

ECA	Component Addition	Component Removal
Event	Client: C sends Requests: R to the Server: S	
Condition	*If:* $R \geq X$	*if* $R < X$
Action	*then:* ADD(B ∈ CMP) in S	*then:* REM(B ∈ CMP) from S

6.3. Interpret results–pattern-based architecting

As the final step, we have selected and applied the End-to-End Service Binding pattern to enable the integration of a Request Broker component between the ECMC Client and Consumption Monitoring components. At this stage, the pattern selection from the catalog is a manual process. We automate the selection of the most appropriate pattern using the Question Option Criteria (QOC) methodology as detailed in Zdun (2007). Pattern application results in architectural modification such that the application of a pattern on the source architecture (Src) in Fig. 4a results in the modified or target architecture (Trg) in Fig. 4c by preserving the intermediate architecture (Inv) in Fig. 4b. Architectural modification in Fig. 4 is based on the Double-Push-Out (DPO with extended details in (Ahmad et al., 2019) approach for architecture modification. The DPO approach allows the source architecture to be modified into the target architecture by using an intermediate architecture. The intermediate architecture represents architectural structures or properties that must be preserved during modification. For example, in Fig. 4b, the pattern aims at integrating a mediator (Request Breaker) while preserving the Consumption Monitoring and ECMC Client components in the architecture. End-to-End Service Binding pattern in Fig. 4 is selected from the catalog (Table 3) for reusable (abstract primitives) and process-driven modification (pre/post-conditions).

7. Evaluations, lessons learned, and conclusions

We report preliminary results from an evaluation of the pattern-based architecting process as in Table 5.

7.1. Assertion evaluation

We assert that pattern-based architecting helps to support the reusable architecture of the cloud-based software. We formulate:

$$1 - \left(\frac{Pattern_{TMO}}{Primitive_{TMO}}\right) \text{ x } 100$$

8. Conclusions and future research

Cloud architectures rely on software services that entail a recurring need for dynamic composition, elasticity and scalability, and multi-tenancy, etc. that can be best supported by applying reusable practices and solutions. We proposed a 3-step; a pattern-driven architecting process that exploits empirically discovered patterns to guide architectural modifications. A collection of patterns enhances, reusability by abstracting (design primitives). Pattern discovery is a continuous process and we support a systemic approach to investigate emerging design problems and their recurring solutions. We conclude the primary contributions as:

- Investigating sources to empirically discover patterns that address cloud architecture requirements such as scalability, elasticity, multi-tenancy, etc. in the SaaS model.
- Exploit the discovered patterns as elements of reuse knowledge that guides a step-wise process of pattern-based architecting for cloud systems.

8.1. Dimensions for future research

We envisage potential future research in the following dimensions:

- Enabling Tool Support: We aim to provide tool support for pattern-based architecting that can automate the manual and laborious tasks that can be error-prone and time-consuming. Tool-based and pattern-supported architecting enables reusability and automation in the architectural design process.

- Establishing Pattern Language: We focus on establishing a pattern language as an interconnected collection of reusable patterns that can be applied in an incremental manner to support phase-wise architecting of the cloud-enabled software.

Table 5: Overview of % reuse for primitive vs patterns

Patterns		Primitives		% Reuse
Pattern Name	TMO	Primitive	TMO	R
End-to-End Service Binding	3	Integration	5	40
Multi-tenant Data Access	3	Data Access	6	50
Tool as a Service	3	Composition	8	62.5
Service Interoperability	3	Integration	5	40
Service Watchdog	3	Monitoring	7	57.1
Service Request Handling	3	Composition	6	50
Service Unity	3	Integration	7	57.1
	21/7=3		44/7=6.2	51 approx.

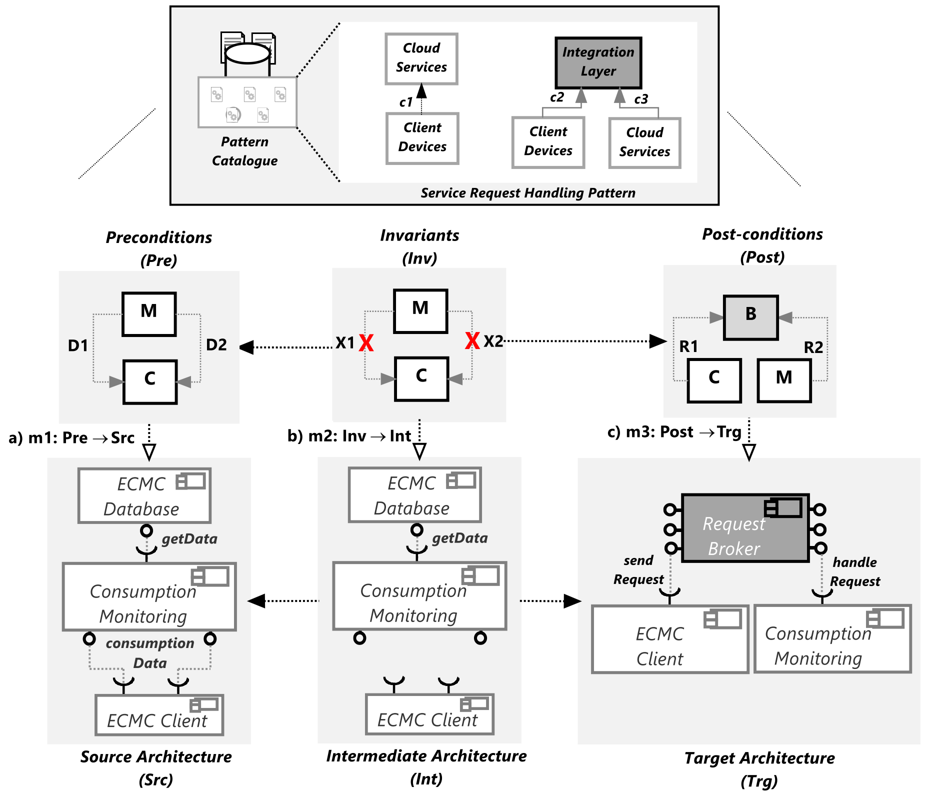


Fig. 4: Pattern-based modification (end-to-end service binding with DPO)

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Ahmad A and Babar MA (2014a). A framework for architecture-driven migration of legacy systems to cloud-enabled software. In the WICSA 2014 Companion, Association for Computing Machinery, Sydney, Australia: 1-8. https://doi.org/10.1145/2578128.2578232

Ahmad A and Babar MA (2014b). Towards a pattern language for self-adaptation of cloud-based architectures. In the Western Indo-Canadian Students' Association 2014 Companion, Association for Computing Machinery, Sydney, Australia: 1-6. https://doi.org/10.1145/2578128.2578227

Ahmad A, Alseadoon I, Alkhalil A, and Sultan K (2019). A framework for the evolution of legacy software towards context-aware and portable mobile computing applications. In the International Conference on Software Engineering Research and Practice, CSREA Press, Las Vegas, USA: 3-9.

Ahmad A, Chauhan MA, and Babar MA (2014b). Cloud styles-towards establishing a catalogue of styles for architecting cloud-based software. Technical Report, TR-2014-171, Software and Systems Section, IT University of Copenhagen, Copenhagen, Denmark.

Ahmad A, Jamshidi P, and Pahl C (2012a). Pattern-driven reuse in architecture-centric evolution for service software. In the 7th International Conference on Software Paradigm Trends ICSOFT, Rome, Italy.

Ahmad A, Jamshidi P, and Pahl C (2012b). Graph-based pattern identification from architecture change logs. In the International Conference on Advanced Information Systems Engineering, Springer, Gdansk, Poland: 200-213. https://doi.org/10.1007/978-3-642-31069-0_18

Ahmad A, Jamshidi P, and Pahl C (2014a). Classification and comparison of architecture evolution reuse knowledge-A systematic review. Journal of Software: Evolution and Process, 26(7): 654-691. https://doi.org/10.1002/smr.1643

Ahmad A, Pahl C, Altamimi AB, and Alreshidi A (2018). Mining patterns from change logs to support reuse-driven evolution of software architectures. Journal of Computer Science and Technology, 33(6): 1278-1306. https://doi.org/10.1007/s11390-018-1887-3

Arcitura (2014). Cloud computing design patterns. Available online at: http://www.cloudpatterns.org

Babar MA and Chauhan MA (2011). A tale of migration to cloud computing for sharing experiences and observations. In the 2nd International Workshop on Software Engineering for Cloud Computing, Association for Computing Machinery, Waikiki, USA: 50-56. https://doi.org/10.1145/1985500.1985509

Bengtsson P, Lassing N, Bosch J, and van Vliet H (2004). Architecture-level modifiability analysis (ALMA). Journal of Systems and Software, 69(1-2): 129-147. https://doi.org/10.1016/S0164-1212(03)00080-3

Boyatzis RE (1998). Transforming qualitative information: Thematic analysis and code development. Sage, Thousand Oaks, USA.

Buschmann F, Henney K, and Schmidt DC (2007). Pattern-oriented software architecture, on patterns and pattern languages. Volume 5, John Wiley and Sons, Hoboken, USA.

Cámara J, Correia P, De Lemos R, Garlan D, Gomes P, Schmerl B, and Ventura R (2013). Evolving an adaptive industrial software system to use architecture-based self-adaptation. In the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE, San Francisco, USA: 13-22. https://doi.org/10.1109/SEAMS.2013.6595488

CDP (2014). Cloud design patterns. Available online at: http://en.clouddesignpattern.org/index.php/Main_Page

Chen KZ (1998). Integration of design method software for concurrent engineering using axiomatic design. Integrated Manufacturing Systems, 9(4): 242-252. https://doi.org/10.1108/09576069810217847

Côté I, Heisel M, and Wentzlaff I (2007). Pattern-based evolution of software architectures. In the European Conference on Software Architecture, Springer, Aranjuez, Spain: 29-43. https://doi.org/10.1007/978-3-540-75132-8_4

Ganek AG and Corbi TA (2003). The dawning of the autonomic computing era. IBM Systems Journal, 42(1): 5-18. https://doi.org/10.1147/sj.421.0005

Harrison NB, Avgeriou P, and Zdun U (2007). Using patterns to capture architectural decisions. IEEE Software, 24(4): 38-45. https://doi.org/10.1109/MS.2007.124

Herbst NR, Kounev S, and Reussner R (2013). Elasticity in cloud computing: What it is, and what it is not. In the 10th International Conference on Autonomic Computing, San Jose, USA: 23-27.

Jamshidi P, Ahmad A, and Pahl C (2013a). Cloud migration research: A systematic review. IEEE Transactions on Cloud Computing, 1(2): 142-157. https://doi.org/10.1109/TCC.2013.10

Jamshidi P, Ghafari M, Ahmad A, and Pahl C (2013b). A framework for classifying and comparing architecture-centric software evolution research. In the 17th European Conference on Software Maintenance and Reengineering, IEEE, Genova, Italy: 305-314. https://doi.org/10.1109/CSMR.2013.39

Rischbeck T and Erl T (2009). SOA design patterns. The Prentice Hall, Hoboken, USA.

Wilder B (2012). Cloud architecture patterns: Using Microsoft azure. O'Reilly Media Inc., Newton, USA.

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