JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
*J. Softw. Evol. and Proc.* 2016; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

# Cloud Architecture Continuity: Change Models and Change Rules for Sustainable Cloud Software Architectures

Claus Pahl[1]*, Pooyan Jamshidi [2], Danny Weyns[3]

[1] *Free University of Bozen-Bolzano, Italy*
[2] *Imperial College London, UK*
[3] *KU Leuven, Belgium & Linnaeus University, Sweden*

## SUMMARY

Cloud systems provide elastic execution environments of resources that link application and infrastructure/platform components, which are both exposed to uncertainties and change. Change appears in two forms: the evolution of architectural components under changing requirements and the adaptation of the infrastructure running applications. Cloud architecture continuity refers to the ability of a cloud system to change its architecture and maintain the validity of the goals that determine the architecture. Goal validity implies the satisfaction of goals in adapting or evolving systems. Architecture continuity aids technical sustainability, that is, the longevity of information, systems, and infrastructure and their adequate evolution with changing conditions. In a cloud setting that requires both steady alignment with technological evolution and availability, architecture continuity directly impacts economic sustainability. We investigate change models and change rules for managing change to support cloud architecture continuity. These models and rules define transformations of architectures to maintain system goals: evolution is about unanticipated change of structural aspects of architectures and adaptation is about anticipated change of architecture configurations. Both are driven by quality and cost, and both represent multi-dimensional decision problems under uncertainty. We have applied the models and rules for adaptation and evolution in research and industry consultancy projects. Copyright © 2016 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Software Architecture; Cloud Systems; Change; Adaptation; Evolution; Change Models.

## 1. INTRODUCTION

Cloud computing enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [25]. A cloud platform provides an elastic execution environment of resources involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality (of service) [49].

Cloud systems are exposed to uncertainties and change. Change can appear in two forms manifested at the architecturally significant level. First, the requirements (goals) can change, which imply the need for the evolution of components to deal with goal changes. Second, the operational aspects of the infrastructure of running application components can change, which imply the need for runtime adaptation of the components to keep satisfying the system-level goals.

We introduce the notion of *cloud architecture continuity* as the ability of a cloud system to change its architecture and maintain the validity of the goals that determine the architecture. Goal validity

*Correspondence to: Free University of Bozen-Bolzano, Italy implies the satisfaction of goals and goal changes in adapting or evolving systems respectively. Our aim is to address the continuity of cloud software architectures and provide goal validity under uncertainty and change, adding to technical and economical sustainability [19, 46, 4].

In particular, cloud architecture continuity aid *technical sustainability* of cloud systems, that is, the longevity of information, systems, and infrastructure and their adequate evolution with changing surrounding conditions. Furthermore, as cloud systems require both steady alignment with technological evolution and 24/7 availability, architecture continuity directly impacts the *economic sustainability* of cloud systems, i.e. maintaining capital and added value. Realising sustainability through cloud architecture continuity requires an integrated solution for evolution and adaptation, which however have often been investigated separately [34, 4]. Architecture continuity for cloud systems needs a solution that considers uncertainties at different levels: uncertainty related to infrastructure and resources, but also uncertainty arising from the presence of multiple actors, distribution and heterogeneity [47, 7]. Specific to the cloud as a virtualised environment is the opportunity to consider resources, i.e., the platform on which a software application runs, in decisions about its design architecture and changes to its initial design over time.

This paper contributes with a set of models and rules for managing change to support cloud software architecture continuity. These change models and change rules define transformations that can be applied to software architectures to maintain the goals of the system and accommodate goal changes. In particular, first, we contribute with change models and rules to support evolution of a cloud, i.e. to deal with unanticipated change of structural aspects of architectures. The transformations defined by these models and rules support cloud migration, characterised by multiple variability dimensions and different types of uncertainty. Second, we contribute with change models and rules to support adaptation of cloud applications, i.e., deal with anticipated change of software architecture configurations during operation. The transformations defined by these models and rules support cloud configuration and adaptation within layers. Both types of models and rules to handle change are driven by quality and cost, and both represent multi-dimensional decision problems under uncertainty that needs to be reflected in the models and rules.

Our research contribution builds upon and extends earlier work [4, 5, 31, 26] into a unifying framework for sustainable cloud architectures that integrates evolution and adaptation under various types of uncertainties. In particular, our contribution are the following: (i) a set of change models and change rules that enable specification of architecture transformations for evolution and adaptation of cloud architectures, (ii) an application of the framework to a concrete case on document management, and (iii) an evaluation of the effectiveness of the framework based on a retrospective evaluation of different cloud applications we have studied over the past years.

The evaluation of unifying framework for sustainable cloud architectures is based on research and industry consultancy projects. In these projects, we applied empirical and experimental techniques, taking practical work of several case studies in real-world projects and technology implementations for platforms such as Azure and OpenStack into account.

The paper is structured as follows. We start with discussing related work in Section 2 and then introduce our change framework in Section 3. A cloud reference architecture is presented in Section 4. We then discuss evolution in Section 5, before addressing adaptation in Section 6. In Section 7, we evaluate the framework before concluding the paper.

## 2. RELATED WORK

We distinguish the two change incarnations: (i) evolution and (ii) adaptation. Lehman has already captured software evolution and adaptation in the form of laws [42]: Continuing change means a system must be continually adapted or it becomes progressively less satisfactory (e.g., losing the ability to maintain business sustainability) Increasing complexity, arising as a system evolves, causes uncertainty unless work is done to maintain or reduce it. Self-regulation or self adaptation is therefore a necessity. A feedback system for intertwined adaptation and evolution processes therefore needs to be multi-level, multi-loop and multi-agent. Adaptive systems have only recently received attention from a software architecture perspective. These are managed by a control loop

[13, 8, 30, 3, 9, 14]. The MAPE-K loop is an example of a feedback loop that implements an adaptive system. We take the respective principles of self-management on board for both adaptation and evolution settings by using the MAPE-K model, but combine these also with models to deal with uncertainty in a service-based cloud context. Our aim is somewhat similar to [8, 30] by aiming at determining model requirements, but we single out uncertainty as a prevalent phenomenon in the cloud environment and integrate this with evolution and change management techniques.

An important software engineering concern, specifically in the maintenance and evolution community, is reuse through variability management [37, 10, 11, 38]. Stahl et al. [37] and Ghezzi et al. [11] look at variability management through model-driven solutions. We adopt this by having an explicit model for the decision process. The models here act as an abstract, pattern-based representation of quality situations that can be used to determine change rules for change enactment. For instance, in the evolution model, the different feature dimensions are covered. Models in pattern-form are here used to select or be transformed into a change rule. Many evolution-oriented migration frameworks have been developed as re-engineering/refactoring solutions [22, 26, 24]. Patterns have been proposed by van Hoorn et al. [36] for adaptive systems. Decision support solutions are discussed by Zimmermann [20] for general software architectures, but also more specifically for cloud-based systems by Andrikopoulos et al. [15]. We here propose a set of cloud-specific migration patterns [40, 50], as reusable solution template following a similar representation as the well-known design patterns. Decision support system as favoured by other authors embed the same knowledge into a recommender system. Our focus is on models, but an extension towards more automation in the evolution cycle could be also considered.

In the context of software architecture evolution, of which migration is an example, sustainability is defined in different ways. Koziolek [41] defines software systems as sustainable if they can be cost-efficiently maintained and evolved over their entire lifecycle, assuming the traditional software architecture definition of ISO/IEC 42010-2007 as a fundamental organization of a system embodied in its components and their relationships to each other. We focus specifically on cloud architectures with their layered organisation of services. We have defined sustainability in this context through resiliency to uncertainty, where resiliency is the ability to withstand or recover from difficult situations Sustainability is the ability to maintain a system at a certain level or rate. We have already pointed out that uncertainty continuously affects systems in its various forms. Thus, resilience to uncertainty is a sustainability concern that is prevalent in cloud environments, i.e., to maintain a system operational with these negative impacts.

A concern to be taken into account is the integration of development and operations of software, often captured under the term continuous development [16] or continuous architecting of software systems [51]. Fitzgerald et al. [16] cover maintenance and reuse by having a broader look. While development methodologies, such as agile development, encourage cross-functional collaboration between analysis, design, development and quality assurance QA in traditional, functionally separated organisations, a cross-departmental integration of these functions with IT operations is lacking. DevOps [18] promotes processes and methods for communication and collaboration between development, QA and IT operations. It is a method that emphasises communication, integration, automation and cooperation between software developers and other IT professionals. It sits at the intersection of development, quality assurance and technology operations. Its adoption is being driven by cloud-related factors such as the wide availability of virtualised and cloud infrastructure from internal and external providers and the increased usage of data center automation and configuration management tools.

Self-adaptive cloud management is the subject of current research [13, 2]. Control theory is utilised in adaptive systems management such as the cloud. Control-theoretic foundations are being explored here to manage uncertainty concerns whereby different models from statistics machine learning and fuzzy control are used [32, 33]. Our own work includes the prediction and the use of fuzzy sets [31, 5]. Fuzzy sets are a natural and effective solution and provides the opportunity to ease the specification of rules using intuitive linguistic concepts. A performance-oriented adaptation of software systems is proposed in [45]. The importance of models in distributed self-adaptive systems is stressed in [48, 44]. Abstractions are proposed as higher-level styles or more integrated formal models. We use here incarnations of the patterns concept as discrete, abstracted solutions.

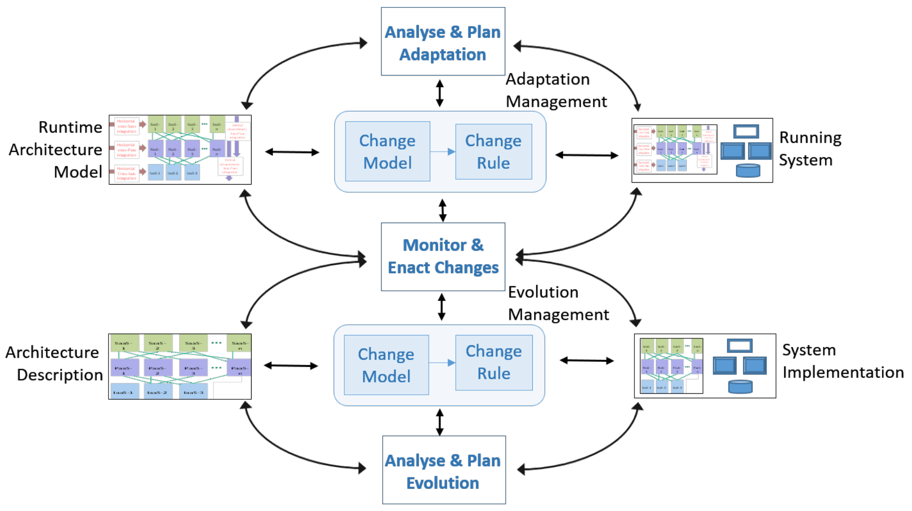


Figure 1. Change Model with Evolution and Adaptation Loops, based on [1].

## 3. SUSTAINABILITY AND CHANGE THROUGH EVOLUTION AND ADAPTATION

### *3.1. Sustainability in Uncertain Environments*

In an ecological context, sustainability refers to the ability of biological systems to remain diverse and productive. More technically, sustainability refers to the endurance of systems and processes. Sustainability is a systemic concept that applies in various contexts from social systems to economy to the environment [46]. Economic sustainability involves using the assets of a company efficiently to allow it to continue functioning profitability over time. While economic sustainability is well-established, there is also technical sustainability. Technical sustainability refers to the longevity of information, systems, and infrastructure and their adequate evolution with changing surrounding conditions, mainly related to the continuous, often fast evolution of technologies.

There is an obvious link between technical and economic sustainability for companies with a focus on software. Through use cases we will link for example needed expansion and modernisation to maintain the customer base as economic driver with a clear technical solution. Business longevity and sustainability are also linked to technical sustainability. There is a need to sustain system architectures through migration (evolution) into the cloud (and changing the platform), resulting from business drivers (market pressure to adapt to cloud, flexible access, expansion strategies). This requires to sustain business validity while adapting a cloud costing model.

Software systems such as cloud applications and service-oriented systems are dynamically composed from autonomous and heterogeneous resources that interact with each other to provide users with often complex functionalities. These systems are architected from small services and lightweight interconnections (cf. microservice architecture [52]) that both can enhance sustainability by applying specialized patterns. These systems operate under highly dynamic conditions where both the components and their interconnections are subject to continuous change, rendering traditional stability assumptions invalid. These dynamic operating conditions introduce uncertainty, which can negatively impact the technical and, as a consequence, the business sustainability of the system [47]. Uncertainty can lead to incomplete, blurred, inaccurate, unreliable or inconclusive results. Thus, uncertainty has an impact on context, goals, models, functional and quality properties. When uncertainty is the rule rather than the exception, managing it becomes a critical problem for the sustainability of software systems. We define a software system to be sustainable if it is *resilient to uncertainty* [54, 53].

Developing sustainable software systems is a challenge, as the ubiquitous uncertainty affects all stages of systems development, from goals elicitation to design, validation and in particular runtime. This raises a set of core challenges that call for changing the way software systems are developed, validated and operated. Handling the continuous change of software and realising sustainable systems requires putting adaptation and evolution as driving principles in both design and operations. Whereas adaptation refers to the ability of mitigating uncertainty in order to keep satisfying the goals, evolution refers to the ability of accommodating uncertainty in order to handle goal changes. Goal continuity then refers to the maintenance of goals in adapting or evolving systems. Goals continuity means therefore (i) goals to be valid and (ii) valid goals are enforced. If goals are continuously maintained, the system is resilient to (not effected by) uncertainty and technical sustainability is maintained. So far, adaptation and evolution have mainly been tackled independently by focusing on development and runtime issues separately. However, the increasing need for business continuity requires modern software systems to be continuously available and continuously meeting the business and technical goals, which blurs the traditional separation between runtime and development time: changes to the system should be applied when the information becomes available while the system is running, this extends the lifetime of any evolution and adaptation to runtime as well.

### *3.2. Sustainable Change in Uncertain Environments*

Change can occur in two incarnations – evolution and adaptation, see Figure 1 based on [4]. There is a need for systems to be sustainable under change, but change causes uncertainties making goal continuity a challenging task. In environments like the cloud that are subject to dynamic composition of autonomous and heterogeneous resources, dynamic operating conditions cause further uncertainties. Thus, the two change incarnations need to be addressed:

- Adaptation to mitigate uncertainty in order to keep satisfying goals,
- Evolution to accommodate uncertainty in order to deal with goal changes.

A change model can capture the adaptation and evolution process to assure sustainability (Figure 1). Models are core artifacts in a process that links system execution, monitoring, analysis and feedback. This is commonly known as the MAPE-K loop [8], with Monitoring, Analysis, Planning and Execution components operating based on retrieval or update of information from/to Knowledge component, which we adopt and adapt to address the uncertainties. For the cloud, a change model needs to consider an explicit representation of the resources that are operated on. The existing change models like the one proposed in [4] or its predecessor in [1] are not cloud-specific. Here an explicit notion of cloud resources and their inherent dynamics would be beneficial. The change model is structured around two loops for short-termed adaptation (between application, resource and the environment) and long-termed evolution (system and application architecture evolution). Evolution management is usually performed by humans, supported by tools. Evolution can be triggered in two ways. First, adaptation management may trigger the need for evolution, i.e., when a problem with no mitigation plan is discovered. When no adaptation plan is available, evolution management analyses the request in order to update the software architecture and the system implementation. The update is then enacted to the running system and the runtime architecture model. Second, system goals may evolve due to changing user requirements or other changes in the environment. For a goal change, the architecture description and the system implementation will be updated. Evolution triggers an update of the running system and the runtime architecture model, e.g., changing or adding new components or integrating platform updates. Obviously, evolution requires synchronization between the adaptation and evolution processes. Evolution may be subject to different uncertainties. In addition to uncertainties with respect to monitoring data (noise [54]) and change enaction time (latency [55]), a key uncertainty of evolution management is goals uncertainty – for example an unanticipated user requirement.

## 4. A CLOUD ARCHITECTURE MODEL

### 4.1. *Cloud Architecture Model*

Cloud services are often categorised into software, platform or infrastructure services. We define a reference architecture using the service types to align them with the architecture of any computer system with hardware infrastructure at the bottom, facilitated by the operating system, then platform components such as databases, middleware and programming environments, and finally the actual applications at the top, see Fig. 2. The architecture of a cloud system can thus be seen as a layered, distributed architecture that has different services linked across layers, down and up-stream:

- Downstream X-YaaS integration: the cloud is a tiered set of service layers, covering infrastructure (IaaS), platform (PaaS) and software (SaaS) services as central service models.
- Cross XaaS integration: Typically different subsystems across possibly distributed and heterogeneous individual services are integrated in a multi-cloud system.

The cloud is a heterogeneous distributed architecture where uncertainty challenges change management. This necessitates architecture management in a feedback loop involving the models. This feedback loop is an intrinsic element of a cloud system, supporting to configure, monitor and adjust resources used to deploy application software. We also distinguish the model term, which aids decision making, and the rules, which enact decisions.

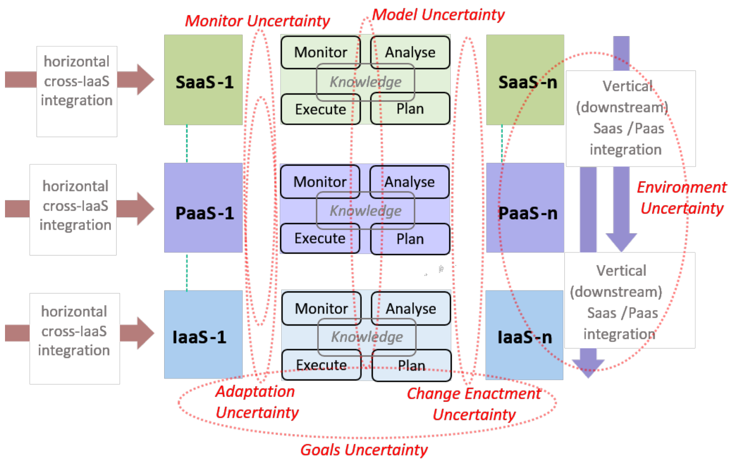

Monitor Uncertainty
Model Uncertainty
horizontal cross-IaaS integration
SaaS-1
Monitor
Analyse
Knowledge
Execute
Plan
SaaS-n
Vertical (downstream) Saas /Paas integration
horizontal cross-IaaS integration
PaaS-1
PaaS-n
Environment Uncertainty
horizontal cross-IaaS integration
IaaS-1
IaaS-n
Adaptation Uncertainty
Change Enactment Uncertainty
Goals Uncertainty

Figure 2. Cloud Architecture Model with Layers SaaS, PaaS, and IaaS.

Apart from the structure of a cloud architecture, another key cloud aspect is the uncertainty arising from its architecture, see Fig. 2 where the uncertainty types identified in Section 3 are linked to architecture components. Different cloud service providers can provide services such as IaaS-1 or IaaS-n for a cloud-based software system. A controller manages the key elasticity concerns of applications that cloud solutions exhibit in a MAPE-K style. Uncertainty arises from the different actors and stakeholders, the different platforms and their monitoring systems and the representation of knowledge – which can all be inconsistent or incomplete. The uncertainty types from Fig. 1 are associated here to the cloud architecture components.

New in the cloud is vertical and horizontal change in and across the layers. SOA is in this respect generic, by treating all services equal. In the cloud, we make the differences between services and their management explicit. We have singled out infrastructure, platform and application layers as these reflect computer systems architectures. As already explained, we see cloud systems as applications running on infrastructure and/or platform services which can jointly be adapted to main goal continuity. The distinguishing factor is cross-layer integration. Key limitations are interoperability concerns. We have carried out work on cross-service availability and recovery services in this context that exemplify the limitations. The workload patterns serve as an example here. In addition to SOA principles, here virtualisation, adaptation, uncertainty and also continuous development and continuous architecting (cf. DevOps) need to be make explicit in the model. The recent microservices architectural style [52] have a very similar principles as SOA style at the logical level, but adds resilience to failure. New architectures like Apache Storm or Spark allow individual services to fail while the whole application is resilient to failure. This resiliency is not possible without full stack cloud hosting of individual services.

## 4.2. *Software Architecture Change Rules and Models*

Models and rules govern the change process in the presence of uncertainty. The cloud architecture (Section 4) includes resources, platform and application concerns, i.e., it helps with architecting, configuring and deploying software, but also managing quality aspects. The change model (Section 3) is a process model covering adaptation and evolution activities. All change techniques can be categorized in terms of MAPE-K model [4]. The focus here are the decision-making activities, i.e., analysis and planning. Model and rules for adaptation and evolution form the knowledge, i.e., K part of the MAPE-K process. Models and change rules are involved in this process as follows:

- models are abstractions of concerns: functional and non-functional;
- these models determine the application of change rules (adaptation/evolution rules)

We need change rules to enact change in evolution or adaptation form and we need models to construct and/or select change rules driven by monitoring or other external information.

Associating specific models to the change model results in the cloud-specific change model in Figure 3 that focuses on the central connection point between evolution and adaptation. It has the cloud architecture at the core of both the evolution and adaptation concerns. This model instantiates the earlier generic change model from Figure 1. In Figure 3, for both forms of change we associate a model to decide on a change plan and change rules to enact change. Evolution is driven by a variability model as a decision support tool that helps in selecting actual transformation rules from a transformation pattern catalog. These transformation patterns exploit solution templates that can be reused across application areas. In a process involving humans, the variability model clarifies concerns, which in turn aids the selection of the transformation patterns. Adaption uses a prediction model and a set of fuzzy rules to address uncertainty and latency in the resource provisioning. In an autonomic setting, the model allows the specification and computation of a response in the form of a quality adaptation rule.

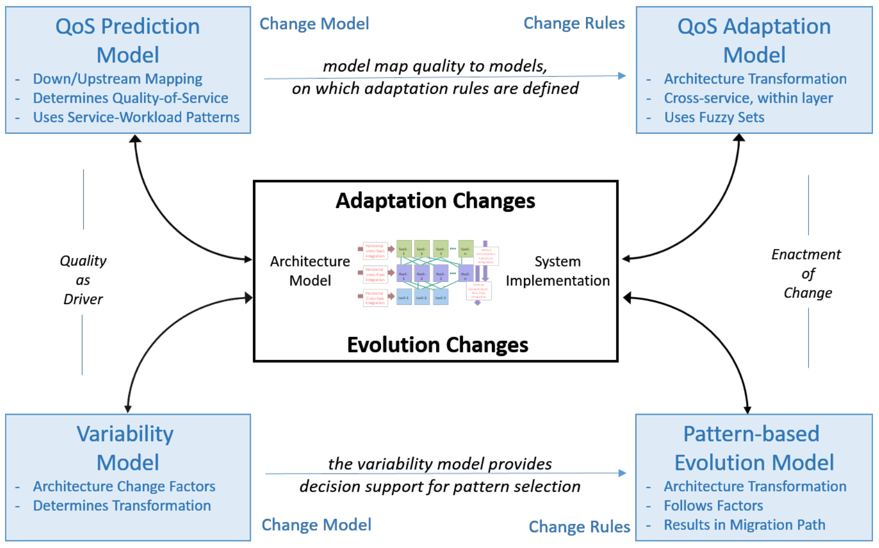

QoS Prediction Model
- Down/Upstream Mapping
- Determines Quality-of-Service
- Uses Service-Workload Patterns
Change Model
Change Rules
model map quality to models, on which adaptation rules are defined
QoS Adaptation Model
- Architecture Transformation
- Cross-service, within layer
- Uses Fuzzy Sets
Adaptation Changes
Architecture Model
System Implementation
Evolution Changes
Quality as Driver
Enactment of Change
Variability Model
- Architecture Change Factors
- Determines Transformation
the variability model provides decision support for pattern selection
Change Model
Change Rules
Pattern-based Evolution Model
- Architecture Transformation
- Follows Factors
- Results in Migration Path

Figure 3. Change Models for Cloud Architecture.

Key terms are models and rules. We use patterns to instantiate solutions for both. Patterns are abstractions that help to map uncertain situations on common recurring situations (models) to the required solutions templates. Despite being used for different aspects (models and rules), the pattern concept is central in describing the way evolution and adaptation is tackled. We differentiate:

- Migration architecture pattern as rules: these are specific instantiation of architecture transformation patterns.
- Resource adaptation pattern as models for situations: (a) when a prediction is needed based on a statistics-based mapping between quality concerns across cloud architecture layers, or (b) to specify a situation concerning a quality aspect within a cloud architecture layer.

The pattern concept is thus a key principle. In both evolution and adaptation, we map a given problem situation to a pattern as a discrete abstraction that helps to solve it. Migration patterns define in abstract terms an architecture transformation Workload patterns determine an adaptation rule. Patterns originate from the users side or can be used to communicate solutions with them.

## *4.3. A Use Case*

Here we discuss a use case that motivates this model architecture before looking at models in detail. This use case describes a real migration project that we worked on with a solution provider in that space. Document image storage and processing is a common concern for many organizations. Documents such as sales receipts or travel documents are scanned, processed and archived. Often, external companies provide the high-speed scanning devices. This solution benefits from a cloud architecture where scanned documents are directly transferred to and stored in the cloud. This enables access by all actors and also third-party processing such as OCR (optical character reading).

*4.3.1. Evolution.* The initial motivation was defined through the following goals: (i) flexible access channels to allow the company, its clients and also possibly third-parties to access the documents from various locations, and (ii) elasticity of large-scale document image processing (high data volumes transferred/stored). We implemented the migration as a two-stage change process:

- first an on-premise to cloud migration (into an IaaS solution) driven by the need to complete the transition quickly – an important time-to-market business continuity constraint as the product was meant to be provided into new markets.
- second a cloud-to-cloud migration (from an IaaS to a PaaS solution) driven by the technical sustainability need to modernise/re-architect the software to gain more flexibility in configuring the software and tailor it to specific clients.

In [23], the architectural options in the modernisation (evolution) process are analysed. As part of a migration project, often several architectural options arise. These are assessed as part of the migration in terms of their performance/cost trade-off. Technical sustainability of the cloud solution needs to meet the business sustainability goals.

*4.3.2. Adaptation.* The solution in this document processing use case is an image processing application, which is characterized by a high volume of data resulting from the images. While storage is an obvious need, there are also situations in which in particular traffic in and out of the cloud is significant. Documents are scanned by high-speed scanners with up to 200 pages p.m. in high quality ($>$ 1MB each) and uploaded to an storage system. Also bulk-downloading does happen. Both situations need to be facilitated for many customers concurrently, resulting a varying demands and consequently elasticity needs to match demand and resources, specifically to maintain acceptable response times.

A first problem is to match service-level performance requirements, e.g., response time for a customer-facing application service such as document download or OCR-recognition, with the configurable IaaS platform settings for CPU, storage and network resources. For instance, given a response-time goal, how can infrastructure resources be configured to match that goal? A second problem is to autonomously manage scalability needs for varying demands as described. Additional resources may be needed for processing and network access demands. Based on some input by stakeholders (*e.g.*, to signal limits of additional resources if costs are too excessive for these), the actual adaptation should be carried out without further human intervention.

*4.3.3. Sustainability.* In our use case, economical sustainability and the need to adapt to an ongoing internationalization process were the drivers of change. Other use cases exist where organizations had to evolve and adapt their architectures to meet environmental and social changes. The charity Oxfam (www.oxfam.org) is an example of an organization that fully migrated to the cloud in order to meet the needs of natural catastrophies and also migrant and refugee crises. Evolving by migrating to the cloud and constantly adapting their IT systems operations to fluctuating demand help in making their engagement more sustainable (http://www.bcs.org/content/conWebDoc/47663). Business sustainability for Oxfam includes a continuous operation of the business even in high-demand times, which in an online situation directly links to technical sustainability in the form of availability. High-demand times such as humanitarian crises are part of the core business and require guaranteed technical and business continuity. Oxfam operates IT infrastructures in over 60 countries, supporting shops and online presences as well operations and logistics centres in regions served by their aid programme. Availability of the whole IT infrastructure needs to be sustained over prolonged periods, in particular in crisis situations. Availability is one of the properties that needs to be maintained throughout.

## 5. EVOLUTION – STATIC ARCHITECTURE

The evolution is a long-term process, involving human intervention. Models consequently need to act as decision support tools for architects and managers and provide implementation guidance for developers. Uncertainty occurs here from an environmental perspective. Many companies consider cloud adoption as a response to environmental uncertainty about innovation, internationalisation or customer base extension. This causes more technical goals such as cost control to be uncertain, too.

### *5.1. Cloud Architecture Migration (into and in-between) and Modernisation*

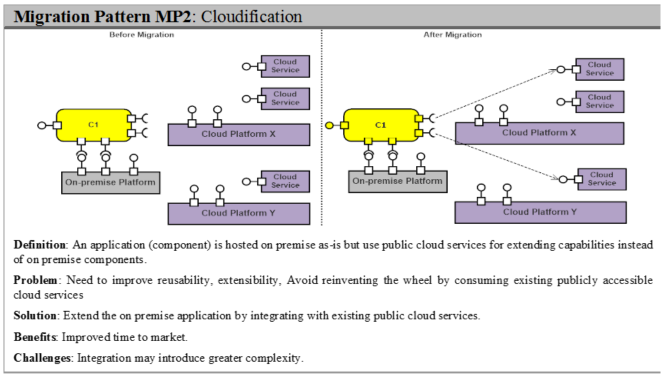
A sample use case of architecture change as an adaptation, modernisation and migration of existing on-premise IT systems to the cloud was introduced earlier. Generally, use cases involve migration into or between clouds. Our suggested process follows the principle activities of the knowledge-driven MAPE-K loop, which we, despite its original focus on autonomic computing, apply here to architecture migration changes as well:

- Monitor: two types of inputs are considered – external, i.e., strategic decisions (into cloud, move on) driven by quality/cost/.. concerns, and internal such as performance or cost.
- Analyse: use a variability model capturing different software features and descriptors to select an architectural transformation pattern.
- Plan: apply transformation patterns selected based on the variability feature model.
- Enact: re-architect or re-engineer in some form the cloud system under consideration based on architectural guidance by the transformation patterns.

Two models defining a rule system are at the core here. Firstly, a variability model defines faceted dimensions of software features. These dimensions are access, application and deployment, covering access channels, application types and technical settings, respectively. Secondly, a pattern model consisting of individual transformation steps (each defined by a change operation). Thus, the model allows a transformation graph to be constructed that based on core change operations defines a migration path. In other words, patterns act as evolution rules for architecture adaptation by lower-level change transformations.

The software variability model is the first change model. It enhances the component view by functional and non-functional classification that helps to deal with uncertainty arising from different access channels and different deployment configurations in addition to the complexity of a software system itself. Orthogonal variability modeling to support multi-cloud application configuration is the concern – accessibility-driven, application-driven and deployment-driven variability. The three variability models at different levels address different concerns of multi-cloud application deployments. The variability model captures three dimensions (access, application and deployment). The access model covers the options that exist to connect to and operate a service in the cloud. The application model covers the functionality of the application in terms of domain concepts. The deployment model captures technical settings, specifically those relevant for quality management. The application model is a fully fledged variability model that represents both functional application commonalities and variabilities of the cloud-based software services. The accessibility and deployment models represent only variabilities that determine non-functional aspects of the cloud services. Different variation points – external (availability, bandwidth, storage, DB), internal (platform, compute, elasticity, pattern) – can be identified. The variability model maps goals into more technical concerns, allowing the identification of the architecture components affected by change and what (new) quality concerns might apply to them. Based on the elaboration of goals in terms of functional and non-functional concerns, migration patterns as solution components for the actual migration can be selected.

Patterns as migration models for software architecture adaptation are the second evolution model [43], which allow the modeling of on-boarding [22] and also in-between clouds [26]. The migration patterns define common architecture transformation paths, driven by functional and non-functional concerns as determined in the variability model. The aim is to also combine architecture transformation with performance and costs analysis [23] as annotations in Fig. 4 like time-to-market. These patterns are similar to design patterns in terms of their presentation involving a structural architecture part, combined with a description of the pattern purpose, the problem description, the key idea of the solution and expected benefits as well as challenges, Fig. 4. Overall, they represent a solution template that can be reused across domain and facilitate the migration process. The migration of larger software systems is a complex process that needs to be broken into smaller steps that address specific components and their needs one-by-one. This results in a composition of patterns as transformation steps into a migration path.



**Migration Pattern MP2**: Cloudification

**Definition**: An application (component) is hosted on premise as-is but use public cloud services for extending capabilities instead of on premise components.

**Problem**: Need to improve reusability, extensibility, Avoid reinventing the wheel by consuming existing publicly accessible cloud services

**Solution**: Extend the on premise application by integrating with existing public cloud services.

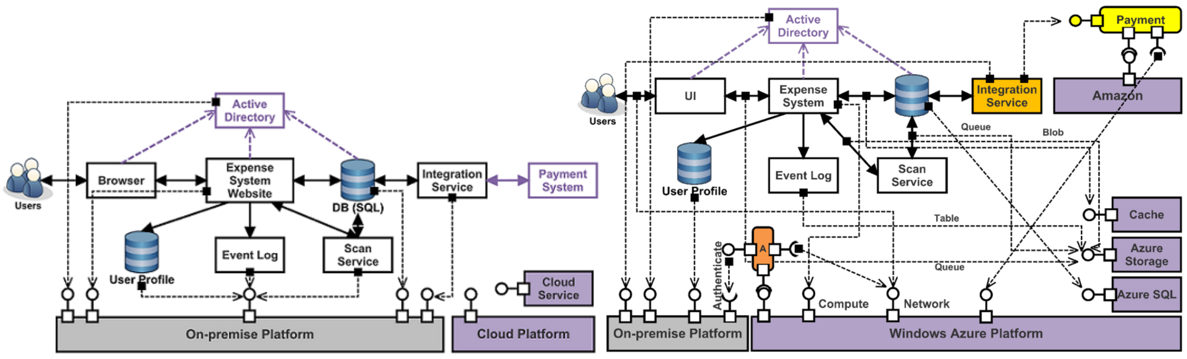
**Benefits**: Improved time to market.

**Challenges**: Integration may introduce greater complexity.

Figure 4. Sample Migration Pattern as an Evolution Rule for Architecture Transformation.

The migration patterns use the variability model for rule selection. Figure 5 presents a sample end-to-end migration, the result of a multi-step migration from an on-premise to a hybrid cloud scenario. In addition to re-hosting core services like the expense system or the event log in the cloud, other more substantial re-architecting steps were carried out. The payment system was replaced by an external third-party service. Additional cloud specific services for caching and storage were included to improve technical properties such as storage capability and access speed. Additional security components were integrated such as a login features for remote authentication between on-premise and cloud components. For more comprehensive set of patterns refer to [50].

This migration process can be seen as an incarnation of a wider architecture modifiability method. We need a scenario-based approach to frame this, in particular one supporting the evaluaton of modifiability. ATAM (Architecture Tradeoff Analysis Method) and ALMA (Architecture-Level



(a) Before Migration (b) After Migration

Figure 5. Sample migration with source and target architecture.

Modifiability Analysis) are the two currently supported and widely used methods [41]. ATAM is not specifically designed for sustainability evaluation, but more generally targets trade-offs between quality concerns. However, a scenario-based approach is useful and can be complemented by architecture-level metrics that we use specifically for the adaptation evaluation. ALMA is he other scenario-based method, but specific to modifiability [17]. ALMA is suitable for software architecture modifiability assessment by employing a set of indicators: maintenance cost prediction, risk assessment. In case of assessing and comparing different system, the modifiability analysis performed with ALMA supports software architecture selection as well. ALMA suggests to follow five steps, which we actually implement as part of the migration pattern method:

- Step 1: Goal definition using the variability model and pattern properties.
- Step 2: Target architecture description using a pattern-based migration path.
- Step 3: Define (elicit) change scenarios. This means to define migration plans, possibly involving different architectural alternatives (represented as alternative paths in a transition graph). The use of patterns allows to assemble alternatives from basic building blocks.
- Step 4: Evaluate scenarios, analyse expected / unexpected changes on a number of qualities. Examples of assessment criteria are cost/workload or performance. This is supported by properties attached to patterns and the selection matrix.
- Step 5: Interpret results (pattern-based migration paths, annotated with quality properties).

The ALMA methods allows to look at maintenance cost and risk assessment, which are relevant concerns at the interaction of technical and business sustainability.

## 6. ADAPTATION – DYNAMIC ARCHITECTURE

Adaptation refers to the adjustment of architectural settings to changing environmental factors relating to the continuity of quality goals of the software in question. Models for architecture (quality) management need to take the uncertainty of the environment into account. This kind of model should allow the architecture to sustain environmental changes while maintaining goal continuity. Models here can in particular link technical concerns such as performance and workload with business concerns such as cost.

Cloud applications can be seen as self-adaptive systems governed by specific software models and methods for development and deployment [48]. Principles such as service-orientation, virtualisation, adaptation and uncertainty apply in the cloud. Adaptation models are needed to manage adaptation. We introduce two models that are suitable to address both the horizontal as well as vertical integration needs expressed through the cloud architecture model introduced earlier. The models are constructed and configured from input data and help in analysing a situation and determining a reaction to possible problems with the situation:

- monitoring: both models are based on monitored configuration and quality data as input.

- analysis: either mapped to a higher architecture layer using pattern-style mappings or are mapped to a mathematical model (fuzzy sets).
- plan: a defuzzification step allows the actual adaptation plan (a rule) to be selected.
- enactment: both can be used to enact adaptation rules based on the mapping results.

We introduce the two models in more detail. Both follow architecturally directly the MAPE-K loop.

## *6.1. Multi Tier/Vertical QoS Mapping and Prediction Model*

The concern of adaptive systems is quality management – which can be implemented through quality prediction and configuration for dynamic service architectures. Short-term sustainability can be achieved through architecture quality adaptation, mapped to the MAPE-K loop.

- Monitor: data from infrastructure monitoring on compute, storage and network resources.
- Analyse: use a workload pattern-quality mapping $\langle a, b, c \rangle \rightarrow x$ to map, determine and predict QoS between lower infrastructure ($a, b, c$ for CPU, network and storage) and higher platform/software layers ($x$ for response time).
- Plan: apply the workload pattern-quality matrix that predicts higher-level QoS based on lower-level monitored data to select resource configuration for the software.
- Enact: enforce the configuration at infrastructure level.

The adaptation technique is based on quality models for quality assurance involving determination, prediction, analysis/mapping and enforcement: QoS determination, analysis and cross-layer mappings can be based on resource consumption patterns that map lower-level consumption to higher-level quality values. This QoS prediction for the higher layers is based on using statistical methods, collaborative filtering and pattern-based mapping. We can demonstrate that the accuracy and performance of the prediction method is adequate. The model is based on the observation that higher-layer quality-of-service parameters such as availability or performance are relatively stable for certain variations of lower-layer infrastructure parameters such as network bandwidth or CPU utilisation. An analysis of monitoring logs for three resource parameters (CPU utilisation rate, storage utilisation, network utilisation) confirmed that smaller variations of about 15-25% around a median value keeps a QoS value steady within roughly the same variation.

These observations can be used to capture the situations for services $s_i$ as mappings between the layers, here called service workload patterns that map the three infrastructure parameters to application-level performance parameters. These service workload patterns manage uncertainty through the multi-faceted configuration and quality patterns. We have used specifically collaborative filtering in the determination process to take observations on similar services into account if the observed data for a single service is not sufficient to provide reliable and accurate predictions.

Two application options emerge for service workload patterns – prediction or configuration. An upwards mapping allows accurate predictions of QoS aspects based on historical resource usage. A downwards mapping allows dynamic VM/storage/network resource configuration to maintain expected quality based on the predictions reflected in the patterns.

## *6.2. Single Tier/Horizontal QoS Analysis Model*

In adaptive systems, the implementation of the control loop enables for instance self-adaptive resource scalability for elastic service provisioning in cloud architectures. Sustainability is then achieved through autonomous adaptive software management to deal with software in uncertain situations (resulting from different sources) through self-adaptiveness:

- Monitor: monitored data is collected, here from the infrastructure resource management.
- Analyse: we fuzzify input to determine adaptation rules. First, we used a combination of expert-defined input and machine learning to adaptation rule definition. Second, we map monitored data into fuzzy sets where each fuzzy set is represented by a membership function.
- Plan: apply rule determination (defuzzify) to select an adapted configuration
- Enact: enforce reconfiguration within the cloud infrastructure

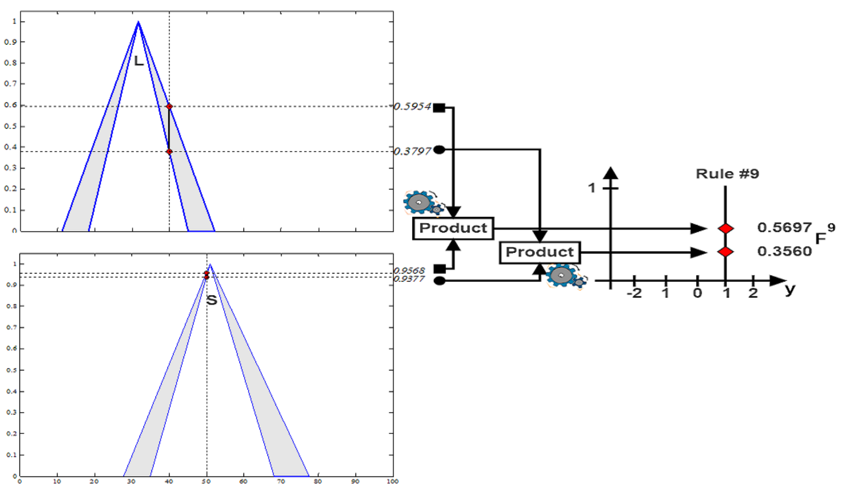


Figure 6. Defuzzification of fuzzy sets – normalised workload/performance (x-axis) and certainty (y-axis).

At the core of the model are the fuzzy sets. Each of these defines a single quality situation (or quality pattern). In Figure 6, the two triangular shaped fuzzy sets describe a workload situation (informally labeled L for 'low') and a performance situation (informally labeled S for 'sufficient'). These sets are the results of expert quantifications of qualitative labels often used in cloud auto-scaling techniques. In a concrete situation, the closest matching fuzzy set is selected and then a product-based calculation of a defuzzified value for the actual decision on resource management is done. This latter process actually is the adaptation rule. In the concrete case shown, a single resource (e.g., a VM) is added. The control surface in Figure 7 shows responses for all input situations (two dimension at the bottom defining normalised utilisation rates from 0 to 100). The response is the change in resources, here ranging from removing two to adding two resource instances.

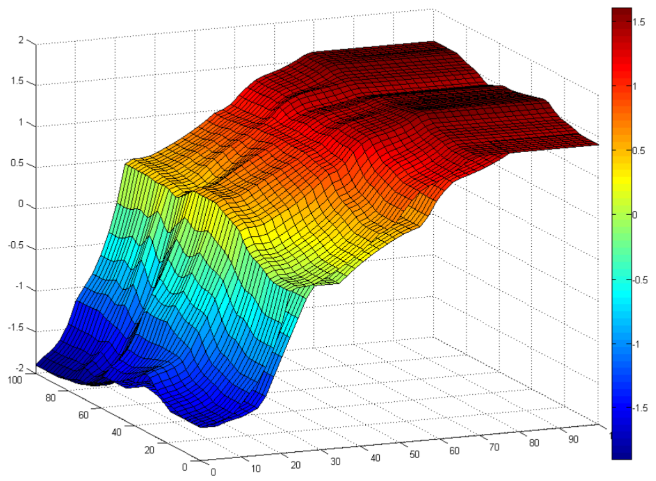


Figure 7. Control surface – linking performances in an SLA maintenance context to cost issues through rules that have a direct link to costs (increasing or decreasing the number of VMs), with normalised workload and performance values (x,y-axis) and number of VMs added/removed (vertical z-axis).

This technique maintains goal continuity, here that performance guarantees are enforced and that costs are minimised at the same time. Self-management of goals can be illustrated through an auto-scaling scenario that aims at maintaining QoS or keeping cost stable. Fuzzy mechanisms including fuzzy learning to determine quality reliably is at the core of the solution. Fuzzy models help with adaptation rule determination. Fuzzification helps to deal with uncertainty, here addressing the different views of different stakeholders. Self-learning as an automated model construction mechanism realises true adaptivity. Machine learning techniques (e.g., Q-learning) can be utilised [6]. Robustness and effectiveness have been demonstrated experimentally by artificially introducing noise up to 10%. We have used different workload patterns for running the experiments. In our experiments, we introduce noise artificially to force the controller to work with uncertain fluctuations up to 10% [5]. The positive results verify the robustness against uncertainty in the monitoring and analysis process.

Note that this exploration of model-based single-layer adaptation targets the IaaS layer in the cloud. The difference at other layers is more in the adaptation actions rather than at the input. Management at the PaaS-layer would involve architecture-level changes rather than resource allocation changes as the change actions. In all cases, input for the adaptation are common concerns such as utilisation rates for the resources at the specific layers, performance or also cost factors. The adaptation actions could involve capacity-related factors (such as storage, just now e.g., at database rather than disk level). However, other relevant adaptation actions could be exchanging functionally equivalent services. In the database context, a relational database service could be replaced by other formats such as NoSQL databases.

## 7. EVALUATION

We need to demonstrate that the model-based change solutions are effective, i.e., help to manage quality and cost in the expected way. We look at evolution and adaptation separately as the model construction and utilisation context is different – although evolution requires synchronisation between the adaptation and evolution processes, which we will address in the evolution part:

- Evolution: we demonstrate model and method suitability using an ALMA-based evaluation and show applicability in multi-cloud setting empirically referring to five use case studies.
- Adaptation: we demonstrate fitness for purpose through experimental analysis of goal maintenance (performance and cost concerns) and show accuracy of prediction through experimental analysis of records.

We discuss long-term technical sustainability (and also economical sustainability), with an emphasis on autonomic mechanisms to create stability in uncertain situations with fluctuating parameters. In this evaluation we draw on different sources: (i) our experience in the IC4 cloud technology centre working with 40 companies, (ii) analyses as part of systematic literature reviews that we have done in the evolution context on cloud migration and architecture evolution, and (iii) related work on adaptation that we considered within our own implementations of adaptive systems.

The techniques reported on have all been implemented and evaluated empirically through use cases and experimentally evaluated implementations. The experimental included simulations (MathLab), implementations in the Azure environment and a dedicated Openstack cluster.

### 7.1. *Evolution*

Evolution is a process that runs non-autonomously, usually involving humans to define goals and identify goal changes. Thus, there is a need to improve on existing architecture methods in terms of cross-stakeholder analysis and documentation of target architecture quality and evolution costs.

*7.1.1. Evolution Model Construction.* The models, e.g., the architecture migration patterns, have been empirically constructed from expert input in the form of cloud migration cases and literature reviews. The variability model has been defined from concerns that emerged from initial feasibility and planning meetings with companies. The patterns reflect common, documented cloud migration cases ranging from simple re-hosting to more advanced architectural scenarios involving multi-cloud settings with external services replacing internal components. The details of the research methodology that we have followed are documented in [50].

We have carried out five case studies involving companies migrating into or between clouds. The companies cover a number of domains: banking, insurance, food and business management.

Furthermore, we carried out experimental migrations of existing systems (e.g., the expense system used earlier). We will provide more details about these case studies below. Available literature on migration approaches was also considered in the process of definition the pattern-based architecture migration framework [27]. The models were here informally presented. Their formalization using a graph-based representation of architectures and graph transformations between these source and target architectures is possible, though [43].

*7.1.2. Empirical Application of Patterns in Use Cases.* Five migration case studies demonstrate the benefits of pattern-driven migration. The patterns provide an analysis mechanism between architect and customer that helps identifying migration concerns (both requirements and also things to avoid), help to determine the scope of a project) in terms of cost and time and, finally, have the benefit of documentation. The case studies cover the following application types (with the drivers and causes of uncertainty regarding goal continuity):

- business management solutions (document image processing) – as described in the use case.
- business management solutions (enterprise repositories) – here internationalisation is the driver, allowing clients to access their services through the cloud.
- an integrated banking solution (account management plus ATM operations) – the solution is provided in different countries in Africa and Asia, raising uncertainty concerns from security to legal in addition to purely architectural ones.
- an insurance solution for multi-product management in multiple countries – uncertainty arises from variability of a single product across different regions and jurisdiction.
- an ERP solution for food production and sales – where a stable in-house solution is prepared for launch as a product into different markets. Food safety regulations impact on the architecture a cloud-based solution.

The projects have been carried out in a 2-stage setting: firstly, an initial feasibility study determining the cloud needs and benefits taking different cloud architecture scenarios into account and, secondly, a full migration project based on the options and plans analysed in the first phase. They have been demonstrated to work for multi-cloud settings. We have already described the IaaS to PaaS migration of the document management use case. The insurance use case also involves a multi-cloud setting where customer relationship management and telephony systems for the call-centres are provided as third-part services. The variability model helps to frame the project. Architecture migration patterns as change rules allow to plan and execute change within a MAPE-style cycle. The migration patterns adequately provide solutions for multi-cloud settings.

As part of the empirical evaluation, we have carried out surveys with both cloud architects providing migration and cloud support (acting as advisors and consultants) and software architects (representing the company's software development team in charge of the actual system development). Generally, two company architects and two consultant architects were involved at this stage for each of the projects.

- For the migration and cloud architects, in-depth familiarity with the approach was required, but this proved beneficial to give less experienced staff guidance to manage migration projects.
- For the company architects, the pattern approach proved to be a method that helped them link their goals with properties of the proposed architecture. It also helped in understanding the need for an incremental approach reflected by the migration path.

A survey has been carried out with architects involved in the migration studies (with at least three architects for each use case), based on a 5-element Likert scale. This includes architects both within the company in question and also the architects from the consulting organisation. Both groups acknowledge the benefits as an analysis tool that helps in the decision making process. It was confirmed by both groups that the solution is suitable for analysis and initial planning, but that it would not constitute a fine-grained work plan.

- All participants agree that the method is suitable for the analysis of cloud migration concerns.

- 88% agree or strongly that the migration method is suitable to analyse and discuss functional and non-functional architecture requirements for migration.
- Its acceptance as a documentation tool (for requirements analysis and migration plan specification) is even higher.

One limitation has been flagged. Whereas 55% strongly agree that the method is suitable for SMEs and that is also suitable for multi-cloud migration (more than 80% positive), almost 43% have concerns with its applicability for large-scale migration projects.

*7.1.3. Experimental Resource Configuration Adaptation and Architecture Evolution.* In addition to implementing migrations of existing IT systems, we did evaluations of possible architectural configurations in the migration process [23]. The migration patterns allow different architectural configurations to be proposed as different migration paths. Different patterns and consequently the composed paths have different QoS properties – which we have experimentally applied to and evaluated for storage patterns. Again, quality and cost are interdependent quality concerns. The detailed results are presented elsewhere [23]. What they demonstrate for this setting is the validity of the quality attributes associated to the migration patterns, i.e., that certain migration patterns have performance or management or deployment benefits.

Here the link between architecture evolution as in the reconfiguration of software components such as databases and resource reconfiguration through autonomic adaptation becomes apparent [43]. If goal continuity cannot be achieved through adaptation alone, a more invasive evolution step might be necessary, i.e., failure to adapt triggers evolution.

*7.1.4. ALMA-based Validation.* As already introduced, ALMA, can be applied and has been followed. ALMA is a scenario-based evaluation method for software architecture quality attributes, focusing on modifiability. Modifiability analysis usually has one of three goals, which we link to the migration pattern framework:

- Prediction of future modification costs: patterns are steps that are easy to cost
- Identification of system inflexibility: patterns help to identify crucial properties that the current system does not possess (or needs to be avoided in the future)
- Comparison of alternative architectures: different migration paths can easily be identified

This shows the suitability of the framework as a modifiability analysis tool in the sense of ALMA from a higher viewpoint.

## *7.2. Adaptation*

Adaptation needs to work autonomously and needs to improve quality goals, i.e., be better than fixed/simple settings. We used a mixed method of empirical and experimental evaluation for these cases and the adaptation models used in them. The use cases considered were:

- Migration studies including performance analysis (the five use cases as introduced plus additional Azure experiments on standard provided solutions).
- Self-management (auto-scale) technique, implemented and evaluated in Openstack and Azure as two different cloud platforms.
- Prediction and mapping: performance and accuracy have here been evaluated experimentally.

This paper brings together the results from individual technical contributions. We therefore omit the technical details, but refer to the relevant papers where relevant.

*7.2.1. Adaptation Model Construction.* The adaptation models are constructed empirically from monitoring logs, literature review and expert input, which demonstrates their validity. The fuzzification is a formalisation of the elicited models, including even qualitative and vague input on autoscaling for cloud resources from the human participants. Formalisation is however necessary for autonomy, be that for the model representation or the decision calculation based on the model.

The prediction model was constructed by mining and analysing monitoring logs and evaluating the model experimentally using real services. The adaptation model was constructed based on expert input, then further calibrated using simulations and later experimentally evaluated using implementations in real-world systems.

We can also employ machine learning for the model construction. While our first adaptation of the auto-scaling was based on fuzzified, empiricially determined expert input [5], we also implemented a machine learning solution based on Q-learning for the rule determination process [6].

*7.2.2. Implementation and Experimental Evaluation.* Experimental results show that we can maintain goals even in the presence of uncertainty [5]. A controller implementing the fuzzy set models can largely maintain the expected quality requirements. We analysed this for performance and workload as the input parameter and tested this with a number of common workload patterns such as gradual increases and decreases, unpredictable fluctuations, steady behaviour and unexpected spikes. Apart from the spikes, all other patterns have been dealt with successfully. The sudden spikes cause principal difficulties due to the latency of the resource provisioning. Normally the time need to launch a new VM prevents a timely reaction to spikes. Prediction can to some extent anticipate this. Exponential smoothing can be applied for trend prediction.

Note that 'slowly varying' 'dual phase', 'steep tri phase', 'large variation, 'big spike' and 'quickly varying' were the workload patterns used for the evaluation as these cover common situations. In our experiments with the five use cases, these six workload patterns turned out to be sufficient to capture the workloads we observed.

Model and rule patterns such as fuzzy sets or workload mappings and adaptation rules (cloud configuration changes resulting from the model analysis) are proven to improve quality (such as performance or resource consumption). The fuzzy sets maintain quality, while also optimising the costs by only deploying the smallest number of resources need to maintain quality. We can demonstrate that our strategy outperforms common over and under provisioning strategies.

For the prediction, we need accuracy of the prediction, but also good performance to employ a potentially time-consuming technique autonomously in a dynamic cloud environment [31]. In this case, our solution combining traditional collaborative filtering with a workload pattern approach improves the accuracy of standard collaborative filtering as past observations are gathered in the form of service workload patterns. Also providing a quality matrix based on workload patterns and the services as input dimensions allows us to associate quality predictions that can be accessed very efficiently at runtime, making the approach almost independent of the data size.

With the fuzzy controller for auto-scaling and the prediction-based QoS-driven configuration, we have two effective instruments to dynamically adapt cloud systems to maintain goal continuity.

### *7.3. Discussion and Threats to Validity*

In all use cases, there is a need for cross-organisational interaction in the architecture change management. This adds to the complexity of the layered architectural setting of cloud systems, with applications and platform generally owned by different organisations. For the evolution, the software provider (client) works with a cloud systems integrator (independent third party) to facilitate cloud-based software systems. For the adaptation, the cloud service user needs to interface with service provider. In order to achieve technical and, consequently, also business sustainability for those organisations with software delivery at the core of their mission, the impact of technology constraints and e.g., provider pricing models is crucial. In both cases, using various forms of patterns proved useful in the communication – both in the form of human interaction, but also in the automated mapping of qualitative user input to automatable adaptation patterns, e.g., the controller construction for autonomous management. Patterns are discrete solution templates that are easy to communicate and understand, but using abstraction and mapping techniques, they can also be used in guiding adaptation in autonomous systems. We also need the distinction into decision support model and rules to adequately support the feedback loop. These observations confirm the fit-for-purposeness of our pattern-based solution that distinguishes decision models and change rules.

Some threats to validity exist. An internal validity concern is that the change management method does not fully includes costing in the solution explicitly – although some support is integrated. Cost is a quality factor that links the trade-off between technical concerns to business sustainability.

- As part of the evolution, migration paths with the constituent patterns allow an experience architect to judge migration cost based on the number of patterns and the complexity of each.
- For the adaptation, we have considered workloads and resource configurations. These can also be translated into costs for these resources. The auto-scaling based on fuzzy sets even explicitly aims to reduce the resources consumed (thus, lowering costs).

We have focussed on technical quality management such as performance here. This provides reliability of the results for this concern. Despite costs being partly covered as in the cloud architecture and quality decisions, for instance related to workload can be directly translated into costs, their full and explicit representation of cost could be considered in future work.

An external validity threat is the generalisability of the results, which is important for a high-level framework as our cloud architecture change model. We have aimed to address this by extracting accepted concepts from the literature and documenting solution patterns through a variety of application cases. The inclusion of experts in the evaluation has also aimed at receiving wider feedback regarding the validity.

## 8. CONCLUSIONS

Sustainability is required in the presence of economical, technical, societal and environmental changes that impact on IT systems. Architectural and goal continuity are our proposition to make ecosystems with software at the core more sustainable in both economical and business terms, the latter in particular if software is at the core of an organisation's operation. Change management in the cloud aims to enable the continuity of the architecture of cloud systems to meet the goals continuously and to make architecture change sustainable. Our change models link evolution and adaptation. What we have demonstrated here is that despite the different concerns both are linked through the notion of goal continuity as our practical experience through the use cases indicates. The mapping of goals to architectures becomes then important to allow the system to continuously meet the goals. We have used a concept of discrete solution templates that capture this mapping.

Cloud applications are software systems with layered, distributed architectures that utilise layer-specific resources provides through services. Software systems in the cloud often spread this architecture both horizontally and vertically. The goals that drive evolution and adaptation often affect different layers. A goal might be expressed at one layer, but needs to be addressed architecturally at a different layer. A cloud reference architecture helps in aligning the concerns better. Evolution and adaptation support requires suitable change models and rules to deal with a multi-layered distributed setting. Due to the uncertainty that prevails in the cloud, using change patterns at the core of models and rules has helped to map uncertain situations into manageable ones. In an evolution context, the cloud as a multi-stakeholder and heterogeneous environment requires a multi-dimensional approach to selecting a suitable evolution process, here done through a variability model driving a staged evolution based on migration patterns. To deal with adaptation, the uncertainty is mastered through statistical and logical approaches that help in defining and identifying adaptation rules. While there are other effective methods based on other models, our investigation singles out important requirements, such as the architectural complexity and uncertainty, and also important commonalities of suitable models. Models drive the decision process and need to aid the change determination, i.e., the selection of suitable change rules. Here, the pattern notion is a commonality. Patterns capture common uncertain situations and are centred around quality concerns.

An important observation to note is that the evolution and adaptation models in the cloud are somewhat distinct (causing differences in the degree of formality needed, e.g., for autonomous processing), albeit also linked: (i) evolution changes the application architecture itself in a process with human intervention and (ii) adaptation adapts the cloud resources (platform and infrastructure provided through services) in an autonomous process. However, the drivers of change are in both cases goals and their continuity. The two loops are linked through goal continuity, which in the case of goal unattainability through adaption triggers an evolution activity. Continuous architecting is a term that serves well as a framework, referring to continuous and incremental improvement of architectural designs of cloud systems and relevant properties. Providing methodological and technical support to this continuous architecting exercise reduces the (re-)design efforts and increases the quality of cloud architecture (re-)deployability by saving the effort of trial-and-error experiments on expensive infrastructure.

We have already discussed the need for a more explicit representation of costs in the decision models. Costs aspects are key for cloud adoption. Furthermore, an important concern for future work is to focus more on learning from past experience in the change process. For the fuzzy resource configuration adaptation, we have already mentioned machine learning to define and improve the adaptation rules and their selection. A comprehensive framework should record all decisions and learn from the effectiveness of change decisions.

## ACKNOWLEDGEMENT

The authors are grateful to the Irish Centre for Cloud Computing and Commerce IC4 (www.ic4.ie) where the migration and scalability work has been carried out by PJ and CP over the past two years. IC4 has also acted as the key academic partner in the five case studies referred to.

## REFERENCES

1. Oreizy, Peyman and Medvidovic, Nenad and Taylor, Richard N, “Runtime software adaptation: framework, approaches, and styles” in *Companion of the 30th international conference on Software engineering*, 2008
2. S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth, “Self-adaptation challenges for cloud-based applications: A control theoretic perspective,” in *International Workshop on Feedback Computing*, 2015.
3. M. Iftikhar and D. Weyns, “Assuring system goals under uncertainty with active formal models of self-adaptation,” in *Companion Proc. International Conference on Software Engineering*. ACM, 2014.
4. D. Weyns, M. Caporuscio, B. Vogel and A. Kurti, ‘Design for Sustainability = Runtime Adaptation U Evolution’, Workshop on Sustainable Architecture: Global collaboration, Requirements, Analysis (SAGRA). 2015.
5. P. Jamshidi, A. Ahmad, and C. Pahl, “Autonomic resource provisioning for cloud-based software,” in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS’14. 2014, pp. 95–104.
6. P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, G.Estrada, ‘’Fuzzy Self-Learning Controllers for Elasticity Management in Dynamic Cloud Architectures”. 12th Intl ACM Sigsoft Conference on the Quality of Software Architectures QoSA. 2016.
7. C. Pahl and P. Jamshidi, “Software architecture for the cloud - a roadmap towards control-theoretic, model-based cloud architecture,” in *Europ Conference on Software Architecture ECSA’15*, 2015.
8. R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, “Software engineering for self-adaptive systems: A second research roadmap,” in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.
9. P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, “Requirements-aware systems: A research agenda for re for self-adaptive systems,” in *International Requirements Engineering Conference (RE)*, 2010, pp. 95–103.
10. L. Baresi and C. Ghezzi, “A journey through smscom: self-managing situational computing,” *Computer Science-Research and Development*, vol. 28, no. 4, pp. 267–277, 2013.
11. C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli, “Managing non-functional uncertainty via model-driven adaptivity,” in *Proceedings of the 2013 International Conference on Software Engineering*. 2013, pp. 33–42.
12. K. Chan, I. Poernomo, H. Schmidt, and J. Jayaputera, “A model-oriented framework for runtime monitoring of nonfunctional properties,” in *Quality of Software Arch and Software Quality*, 2005.
13. A. Filieri, M. Maggio, K. Angelopoulos, N. D’Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, “Software engineering meets control theory,” in *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems SEAMS’2015*, 2015.
14. S. Kounev, “Self-aware software and systems engineering: A vision and research roadmap,” *GI Softwaretechnik-Trends*, vol. 31, no. 4, pp. 21–25, 2011.
15. V. Andrikopoulos, T. Binz, F. Leymann, S. Strauch: How to adapt applications for the Cloud environment. Computing, vol. 95, no. 6, pp. 493535 (2012)
16. B. Fitzgerald and K.-J. Stol, “Continuous software engineering and beyond: Trends and challenges,” Intl Workshop on Rapid Continuous Software Engineering RCoSE. 2014.
17. P. Bengtsson, N. Lassing, J. Bosch, H. van Vliet: ‘’ Architecture-level modifiability analysis (ALMA)”. Journal of Systems and Software, 69(1), 129-147. 2004.
18. A. Brunnert et al., “Performance-oriented devops: A research agenda.”
19. D. Garlan and M. Shaw, *An introduction to software architecture*. World Scientific, 1994, vol. 1.

20. O. Zimmermann, "An architectural decision modeling framework for service oriented architecture design," Ph.D. dissertation, Universität Stuttgart, 2009.
21. O. Zimmermann, P. Krogdahl, and C. Gee, "Elements of service-oriented analysis and design," *IBM developerworks*, 2004.
22. C. Pahl and H. Xiong, "Migration to PaaS clouds-Migration process and architectural concerns," IEEE Intl Symp on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), pp. 86–91, 2013.
23. H. Xiong, F. Fowley, C. Pahl, N. Moran, ''Scalable architectures for platform-as-a-service clouds: performance and cost analysis,".European Conference on Software Architecture, 2014.
24. H. Xiong, F. Fowley, and C. Pahl, "An architecture pattern for multi-cloud high availability and disaster recovery," in *Workshop on Federated Cloud Networking FedCloudNet'2015*, 2015.
25. P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.
26. P. Jamshidi, C. Pahl, S. Chinenyeze, and X. Liu, "Cloud migration patterns: A multi-cloud service architecture perspective," in *Service-Oriented Computing - ICSOC 2014 Workshops*, 2015, pp. 6–19.
27. P. Jamshidi, A. Ahmad, C. Pahl, Cloud Migration Research: A Systematic Review, IEEE Transactions on Cloud Computing. 2013
28. N. Antonopoulos and L. Gillam, *Cloud computing: Principles, systems and applications*. Springer, 2010.
29. S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *Intl Conf on Advanced information networking and applications*, 2012, pp. 15–22.
30. B. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*. Springer, 2009.
31. L. Zhang, Y. Zhang, P. Jamshidi, L. Xu, and C. Pahl, "Workload patterns for quality-driven dynamic cloud service configuration and auto-scaling," in *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC), 2014*, Dec 2014, pp. 156–165.
32. J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," *Intl Symp on Parallel Architectures, Algorithms and Programming*, 2010, pp. 89–96.
33. P. Cedillo, J. Jimenez-Gomez, S. Abrahao, and E. Insfran, "Towards a monitoring middleware for cloud services," in *Intl Conf on Services Computing (SCC'15)*, 2015, pp. 451–458.
34. R. Heinrich, E. Schmieders, R. Jung, K. Rostami, A. Metzger, W. Hasselbring, R. Reussner, and K. Pohl, "Integrating run-time observations and design component models for cloud system analysis," 2014.
35. V. Stantchev and C. Schräpfer, "Negotiating and enforcing qos and slas in grid and cloud computing," in *Advances in Grid and Pervasive Computing*, ser. LNCS, N. Abdennadher and D. Petcu, Eds. 2009, pp. 25–35.
36. A. Van Hoorn, M. Rohr, A. Gul, and W. Hasselbring, "An adaptation framework enabling resource-efficient operation of software systems," in *Warm Up Workshop for ACM/IEEE ICSE'10*, 2009.
37. T. Stahl, M. Voelter, and K. Czarnecki, *Model-driven software development: technology, engineering, management*. Wiley & Sons, 2006.
38. T. Vogel and H. Giese, "Model-driven engineering of self-adaptive software with eurema," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 4, p. 18, 2014.
39. Z.-S. Hou and J.-X. Xu, "New feedback-feedforward configuration for the iterative learning control of a class of discrete-time systems," 2007.
40. C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud computing patterns," 2014.
41. H. Koziolek, "Sustainability Evaluation of Software Architectures: A Systematic Review," Joint ACM Symposium on Quality of Software Architectures QoSA and Architecting Critical Systems ISARCS, 3-12, 2011.
42. Lehman, Meir (1980). "Programs, Life Cycles, and Laws of Software Evolution". Proc. IEEE 68 (9): 10601076.
43. P. Jamshidi, M. Ghafari, A. Aakash, C. Pahl (2013) A framework for classifying and comparing architecture-centric software evolution research. In: 17th European Conference on Software Maintenance and Reengineering.
44. J. M. Barnes, D. Garlan, and B. Schmerl. Evolution styles: Foundations and models for software architecture evolution. Softw. Syst. Model., 13(2):649-678, May 2014.
45. Elkhodary, N. Esfahani, and S. Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE'10, 2010.
46. C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff and C. Venters. Sustainability Design and Software: The Karlskrona Manifesto. 37th International Conference on Software Engineering. 2015.
47. Perez-Palacin and R. Mirandola. Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. Intl Conference on Performance Engineering, ICPE '14, pages 3-14, 2014.
48. D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans. Auton. Adapt. Syst., 7(1):8:1-8:61, May 2012.
49. L. Schubert, K. Jeffery, and B. Neidecker-Lutz. The future of cloud computing, opportunities for European Cloud computing beyond 2010. Expert Group report, public version, Volume 1, 2010.
50. P. Jamshidi, C. Pahl, N. C. Mendonca, Pattern-based multi-cloud architecture migration, Software: Practice and Experience, Wiley Online Library, 2016.
51. M. Bersani, F. Marconi, D. Tamburri, P. Jamshidi, A. Nodari, Continuous Architecting of Stream-Based Systems, WICSA, 2016
52. A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, IEEE Software, vol 33, no 3, 2016.
53. P. Jamshidi, C. Pahl, N. Mendonca, Managing Uncertainty in Autonomic Cloud Elasticity Controllers, IEEE Cloud Computing, vol. 3, no. 3, pp. 50-60, 2016.
54. N. Esfahani, S. Malek, Uncertainty in self-adaptive software systems, Software Engineering for Self-Adaptive Systems II, Springer, 2013.
55. J. Camara, G. Moreno, D. Garlan, Stochastic game analysis and latency awareness for proactive self-adaptation, SEAMS, 2014.