SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2016; **00**:1–25
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Pattern-based Multi-Cloud Architecture Migration

Pooyan Jamshidi[1], Claus Pahl[2*], Nabor C. Mendonça[3]

[1] *Imperial College London, United Kingdom*
[2] *Free University of Bozen-Bolzano, Italy*
[3] *University of Fortaleza, Brazil*

SUMMARY

Many organizations migrate on-premise software applications to the cloud. However, current coarse-grained cloud migration solutions have made such migrations a non transparent task, an endeavour based on trial-and-error. This paper presents V-PAM (Variability-based, Pattern-driven Architecture Migration), a migration method based on (i) a catalogue of fine-grained service-based *cloud architecture migration patterns* that target multi-cloud, (ii) a *situational migration process framework* to guide pattern selection and composition, and (iii) a *variability model* to structure system migration into a coherent framework. The proposed migration patterns are based on empirical evidence from several migration projects, best practice for cloud architectures and a systematic literature review of existing research. V-PAM allows an organization to (i) select appropriate migration patterns, (ii) compose them to define a migration plan, and (iii) extend them based on the identification of new patterns in new contexts. The patterns are at the core of our solution, embedded into a process model, with their selection governed by a variability model.

Received . . .

KEY WORDS: Cloud Architecture; Microservice Architecture; Cloud Migration; Migration Pattern; Multi-Cloud, Situational Method Engineering, Variability Model

1. INTRODUCTION

The migration of software applications to the cloud [1] enables to benefit from the cloud promise of converting capital expenditure to operational cost [2]. Mixing cloud architecture with private data centers adds operational efficiency for workload bursts while legacy systems on-premise still support core business services [3]. Instead of re-architecting applications, they can be re-hosted from on-premise to possibly multiple cloud offerings, either private or public ones. We are concerned with the migration of legacy on-premise software to multi-cloud architectures. According to a Gartner report [4], multicloud strategies will become common for 70 percent of organizations by 2019. Multi-cloud deployment is particularly effective in dealing with the following challenges [5]:

- Users are widely distributed around multiple data centers.
- Country regulations limit options for storing data in specific data centers.
- Circumstances which require public clouds to be used jointly with on-premises resources.
- Cloud-based applications must be resilient to the loss of a single data center or cloud provider.

Current cloud migration methods are coarse-grained, making detailed planning difficult. In particular, existing cloud migration processes do not consider a migration plan as a verifiable multi-step artifact [1]. The plan is prepared at either a very broad strategic level with no technical reference

*Correspondence to: Faculty of Computer Science, Free University of Bozen-Bolzano, 39100 Bolzano, Italy.

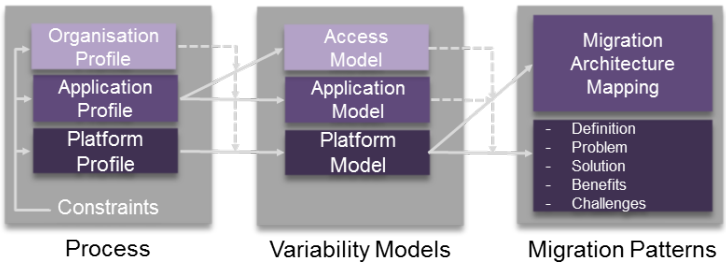


Figure 1. V-PAM Migration Framework – Components and Dependencies.

or very detailed and technical not suitable for non-technical stakeholders [6]. Thus, the repeatability of those migration processes is limited. Consequently, we need:

- Architecture migration patterns as building blocks for planning a migration.
- A migration process to explicitly guide activities that include migration planning and execution via a repeatable and transparent pattern selection and composition.
- A variability model to help identify decision points and map these to architectural solutions.

We address the reorganization of multi-tier applications into disjoint groups of services, such that each group can be deployed separately in different platforms (i.e., cloud platforms or on-premise platform), while preserving and in most cases enhancing the desired properties of the application.

In this paper, we present V-PAM (Variability-based, Pattern-driven Architecture Migration), a cloud architecture migration method, see Figure 1. V-PAM defines activities to plan and execute cloud migration [7] based on the concept of patterns or templates, here describing the entities involved in the process. To account for the situational context of applications, *e.g.*, security, performance, availability needs, existing approaches suggest a trade-off between flexibility and ease of migration using a fixed set of migration strategies [1]. We propose an assembly-based approach based on our experience in situational method engineering [8] where a method is constructed from reusable method fragments and chunks [9]. This allows creating a migration plan from scratch by combining existing migration building blocks in the form of migration patterns.

We present 9 core and 6 variant cloud-specific architecture migration patterns, extracted based on empirical evidence from a number of migration projects [10], best practice for cloud architectures [5], [11] and a systematic literature review [1]. Our main contribution is a set of fine-grained service-oriented migration patterns, framed in a migration process, that allows architects (i) to plan the migration based on patterns and (ii) communicate the migration plan and the decision has been made with non-technical stakeholders. The patterns define architectural changes in the application re-engineering and deployment setting, through which an application is gradually modernized and deployed in a multi-cloud environment. In this context, a migration plan is defined as a composition of selected patterns for addressing specific architectural situations and needs.

We extended our earlier work in [6] to provide a generic framework. The pattern selection and migration plan formulation is embedded into an overarching migration process [12] and a variability model [13] that has been repurposed to support the pattern selection activity, see Figure 1. The process focuses on the identification of the pattern application or situational context, consisting of the organisation, the target software application and the selected cloud platform, captured as profiles and relevant migration constraints. The technical aspects identified for the application and platform are then used (taking organisational constraints into account) to identify selection criteria for the patterns based on a 3-dimensional variability model. The variability model looks at patterns from the system access, application and platform perspectives. Taking the platform aspects as the key requirements that influenced by access and application concerns, suitable patterns are then selected.

We outline our research methodology in Section 2. Section 3 introduces the process model and Section 4 discusses the variability model. In Sections 5, we overview the pattern-based approach and detail the pattern catalogue. We describe its application in a situational context in Section 6.

DOI: 10.1002/spe

The usability of the approach is then evaluated through a cloud migration case study in Section 7, before ending with related work (Section 8) and conclusions (Section 9).

2. RESEARCH METHODOLOGY

The first step to determine a migration process and patterns was to identify the concerns of organizations moving on-premise applications to the cloud. We have identified four categories based on feedback from industry partners in IC4 research centre [10]:

- Availability. Cloud environments typically guarantee a minimum availability.
- Management. Use runtime information to monitor and support on-the-fly changes.
- Scalability. Scale out to meet bursts in demand and scale in when demand decreases.
- Resiliency. Provide ability for systems to gracefully handle and recover from failure.

We use three methods to determine process and patterns under consideration of the key migration concerns. These have been used independently to identify (a) common activities of a migration process and (b) a collection of candidate patterns that is as complete as possible.

- Focus groups and expert interviews have primarily been used to identify a common process.
  We used focus groups to identify migration process concerns. The organizations involved were consultants for SME migration and larger multi-nationals technology providers and systems integrators [10, 14]. Through migration expert interviews, we looked at common processes for migration towards cloud as a framework for more fine-grained patterns. These covered IaaS, PaaS and SaaS migration projects.
- A Systematic Literature Review (SLR) has been used to identify documented patterns.
  We recorded existing cloud design and architecture patterns [5, 11]. A major role in this process was played by a SLR on cloud migration [1]. We detected shortcomings associated with these design patterns when we applied them in migration planning. The patterns were either limited to specific platforms [5] or fine-grained at a very technical level [11]. To redesign an on-premise application with these patterns requires a deep knowledge of vendor-specific services as well as a fair understanding of detailed design documents. Thus, a migration plan based on these patters cannot be communicated with non-technical stakeholders. Thus, we generalize the architectural elements of these cloud architectures with general concepts of software architecture, as we presented in [15], i.e., components, connector, on-premise/cloud platform, cloud service, cloud broker.
- Empirical Analysis of projects we had access to and Pattern Synthesis have been used to both identify and formulate patterns and also the processes in which they are used.
  We analyzed migration projects for a range of CRM and retail systems as well as PaaS platform services. We generalized emerging patterns, considering patterns retrieved from the SLR based on different architecture scenarios that satisfy the migration concerns. Coarse-grained on-premise applications are not agile enough to respond to variations in workload. In the cloud, the deployment of high-usage components can be optimized independently of low-usage ones. Re-architecting into independent components reduces dependencies and enables optimization for scalability and performance. However, challenges remain: (1) on-premise application modernized in isolation, not part of a consistent architecture; (2) modernization performed primarily for technical reasons resulting in sub-optimal response to business change; (3) architectures determined bottom-up from existing APIs and transactions may need re-evaluation for multi-clouds.

Their combination aims to provide a systematic, unbiased and comprehensive process and pattern extraction method.

DOI: 10.1002/spe

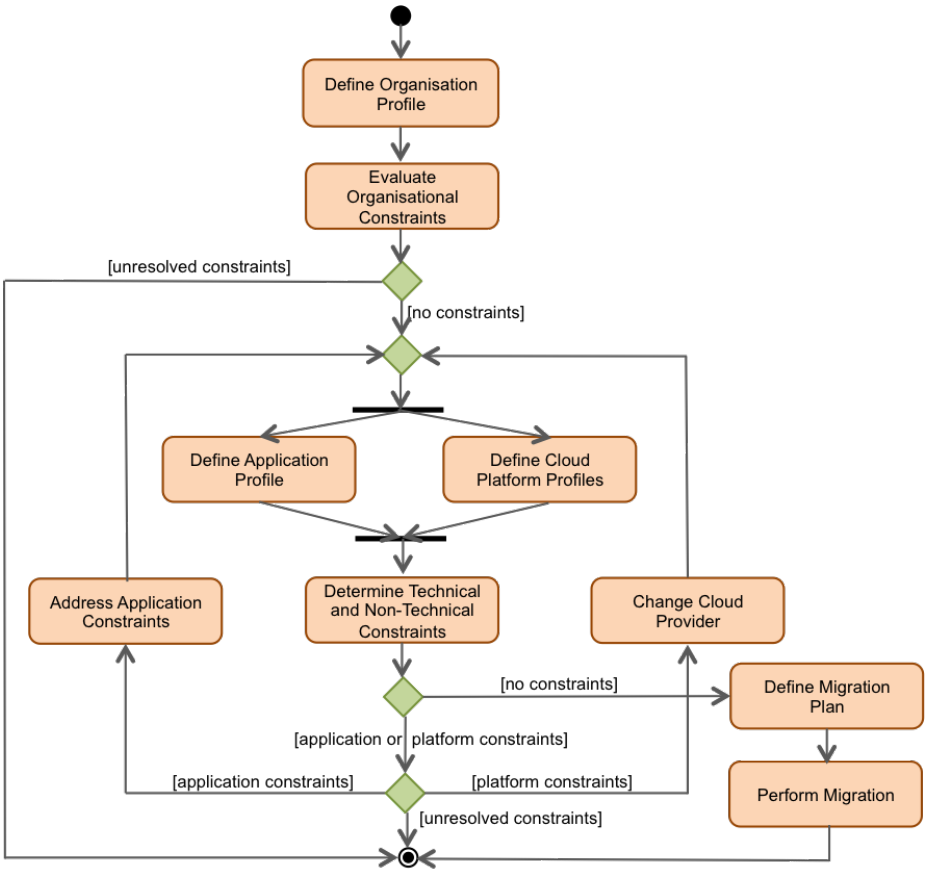


Figure 2. The situational migration description and process model.

3. SITUATIONAL DESCRIPTION AND PROCESS MODEL

Central activities of the overall migration process are the guided identification and analysis of factors that might influence the selection of the cloud architecture and the planning of the migration task. These factors relate to the characteristics of the main entities in the migration decision – which are the interested organization, the on-premise legacy architecture, and the possible cloud provider platforms. We use profiles to characterise those entities – which is a form of a situational description. The profiles act as reusable templates, i.e., with an increasing number of profiles, the identification of new profiles close to those of the entities involved in the migration scenario becomes easier. Once the entity profiles are created, the next step is an analysis to identify potential migration risks and constraints and map them onto architectural patterns.

The situational description process is organized into nine activities – from the definition of profiles to the actual migration of the application to the cloud, see Figure 2. Please note that we do not describe the process part of the model here in full – refer to [12] for details. The situational description part acts here as a conceptual framework into which the more architecture-oriented pattern migration solution is embedded. One of the key reasons that we selected the process model in [12] in this paper is that it covers all the migration processes (including planning, execution, evaluation and crosscutting concerns) in the Cloud-RMM reference framework [1].

3.1. Define Organization Profile

The organization profile needs to capture information about legal or administrative characteristics relevant for the migration. Sample characteristics are policies, guidelines, laws or other governance

DOI: 10.1002/spe

rules. The aim here is to allow the detection of organizational constraints that might affect a cloud migration decision, for example:

- How does the organization acquire and allocate its computing resources?
- Where does the organization develop, test and deploy its software products and services?
- Is the organization subject to any law or legal restriction on the physical location of its data and/or applications?

These examples can guide the identification of potential organizational constraints:

- Divergences between emergency handling policies established within the organization and those implemented by the cloud provider;
- Loss of governance and/or control over existing IT resources;
- Dependence on legacy applications and/or data that cannot be accessed from outside the organization;
- Risk of an unauthorized third party accessing critical business data that is kept in the cloud;
- Legal restrictions on the physical location of critical IT resources (*e.g.*, governmental data that must be stored within regional or national boundaries);

3.2. Define Application Profile

The next step is to create a profile for the application to be migrated, which feeds later into the pattern selection. The application profile captures properties of the on-premise application that might impact on its migration. This allows an analysis of the suitability with respect to candidate cloud architecture models. Two aspects are covered – its usage and the technical characteristics.

Usage characteristics refer to features of the application related to its use and operation. The aim is to identify key functional and non-functional aspects of the application possibly affecting its migration to the cloud, such as:

- Features. What are the main features of the application?
- Users. How many users access the application and from which locations?
- Usage Patterns. What are the usage patterns of the application (*e.g.*, periods of low, normal and high user demand)?
- Cost. What is the cost required to operate and maintain the application by the organization?

Technical characteristics relate to the technologies used/needed to implement and deploy the application:

- Architecture. What are the key components of the application and their dependencies (overall structure and interdependencies between architectural components)?
- Technologies – Implementation. What are the technologies used to implement the application components (*e.g.*, operating system, programming language, development platform, third party components and frameworks)?
- Technologies – Deployment. What are the technologies necessary to run the application (*e.g.*, operating system, execution environment)?
- Data Management. What are the technologies used by the application to handle its data (*e.g.*, file system, database, persistence mechanism)?
- Data Traffic. What is the data traffic received/sent by the application? Is there any stringent quality-of-service (QoS) requirement for the application (*e.g.*, performance, availability, reliability and security requirements)?
- Configuration. What is the minimum hardware configuration necessary to run the application?
- Platform Services. Is there any other system or application whose services or data the target application depends upon? Where are those systems located? Can those systems be accessed from outside the organization?

DOI: 10.1002/spe

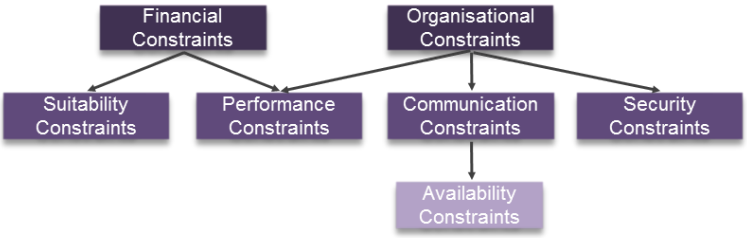


Figure 3. Constraints influence diagram.

3.3. Define Cloud Platform Profiles

Profiles of one or more cloud platforms (including multi-cloud scenarios) should also be created. Each candidate cloud platform and its provider can be captured in terms of whether they satisfy the organisational and technical constraints.

- Service models. What are the service models offered by the provider (*e.g.*, infrastructure-as-a-service, data- as-a-service, platform-as-a-service)?
- Resources. What types of resources (*e.g.*, virtual machines, storage space, development environments) does the provider offer as part of each of its service models?
- Pricing Models. What are the costs and price models (*e.g.*, per hour on demand, per hour reserved, market bidding) for each type of resource?
- SLA. Does the provider offer any form of service level agreement (SLA) guarantees?
- Location. How many and where are located the provider's data centers?
- Services. Does the provider offer any other useful additional service (*e.g.*, backup, monitoring, auto-scaling)?
- Security. What are the security mechanisms put in place by the provider?
- Implementation. What implementation technologies/resources (*e.g.*, programming languages, development platforms, software licenses) does the provider support?
- Monitoring. Does the provider allow access to its internal operational logs (*e.g.*, for auditing or forensic purposes)?

Platform profiles constrain the pattern selection decisions, as we will see later on in the paper.

3.4. Determine Technical and Non-Technical Constraints

Now, it is important to analyse the joint consistency of the organization, application and cloud platform profiles. We propose a set of constraint types, see Figure 3, that a developer should define and evaluate. We have extracted seven main constraint types based on reviews of cloud migration approaches (*e.g.*,[16][17][18][19][20][21][7],[1]), and on our on experience in deploying real-word applications in the cloud [22]. These constraint types include financial, organizational, security, communication, performance and availability constraints. They should all be defined and evaluated within the same context.

Two types of *non-technical constraints* are financial and organizational constraints:

- Financial constraints. An example is the cost to operate the application in the cloud. Calculating this cost may be non-trivial, involving technical (*e.g.*, number and types of cloud resources required) and non-technical (*e.g.*, expected user demand) factors.
- Organizational constraints. Examples are organizational constraints where the evaluation depends on specific knowledge of the application or candidate cloud provider, *e.g.*, if the organization is legally required to keep application data within a certain region.

The remaining four types are all *technical constraints*:

- Security constraints. These help to determine whether the application and/or the organizations security requirements are in accordance with the security mechanisms offered by the cloud

DOI: 10.1002/spe

provider. A typical example is the level of encryption supported in the cloud – both within and into the cloud infrastructure.
- Communication constraints. These address the application's communication requirements, usually expressed in terms of bandwidth, latency or data transfer rate. This aspect is largely influenced by the quality of the network services used by the organization and the provider's data center.
- Performance constraints. These relate to the capacity of the application to serve its users in a timely manner. This aspect is connected to the capacity of the cloud resources (*e.g.*, processing, memory, storage) offered by the platform as well as to the communication quality.
- Availability constraints. Availability of resources in the cloud is related to the SLA guarantees offered by the cloud provider. They can also be affected by communication constraints.

Those different types of constraints can have interdependencies and, therefore, should not be analysed in isolation. Figure 3 shows an influence diagram that illustrates common dependencies between the constraint types discussed above.

There are two steps (cf. Figure 2) that could cause the process to iterate. Thus, any application constraint problems identified should be addressed, *e.g.*, by changing the application profile. Moreover, the cloud provider might also need to be changed if platform conflicts occur, in which case a revised cloud platform profile should be considered.

3.5. Define Migration Plan and Perform Migration

Finally, the definition of a migration plan combine all the concerns raised during the situational capture process. This process starts with a collection and preparation exercise of relevant information for decision making about the cloud migration. Constraints need to be addressed by either finding solutions or by circumventing them. At the core of the method, which we detail over the next three sections for a multi-cloud setting, is a collection of architecture-oriented migrations patterns that help to manage and find solutions for the *technical constraints* in particular. Using a variability approach is central in this method. The selected patterns are composed to construct a migration plan. The migration plan is then executed by performing sequential architectural refactorings while in each step of the execution the identified constraints need to be satisfied.

4. VARIABILITY APPROACH TO MIGRATION DEFINITION

In order to build manageable and scalable cloud applications that meet the communication, availability or performance constraints just discussed in previous section, a multi-cloud deployment is often appropriate [23, 24]. The V-PAM (Variability-based, Pattern-driven Architecture Migration) method proposed in this work aims at facilitating migration pattern selection and customisation for applications that run on multiple independent clouds.

4.1. Motivation of Multi-cloud and Variability

Multi-cloud denotes the usage of multiple, independent clouds by a client or a service. A multi-cloud environment is capable of processing user demand and distributing work to resources deployed across multiple clouds [25]. A multi-cloud is different from a federation where a set of cloud providers voluntarily interconnect their infrastructures to allow sharing of resources among each other [25]. Hybrid deployments can be considered as a special case of multi-cloud where an application is deployed in both on premise infrastructure as well as on cloud platform(s). Such a deployment model is essential in cases where critical data needs to be kept in-house in corporate data centers. Different application types and requirements may benefit from and even demand a multi-cloud deployment – see [23] for supplementary materials.

In a multi-cloud configuration perspective, parts of the application can be deployed on PaaS, IaaS or both [24, 15], see Figure 4 for an example. The wide range of cloud providers currently available and their platforms likely to host the application makes the selection a proper architectural

DOI: 10.1002/spe

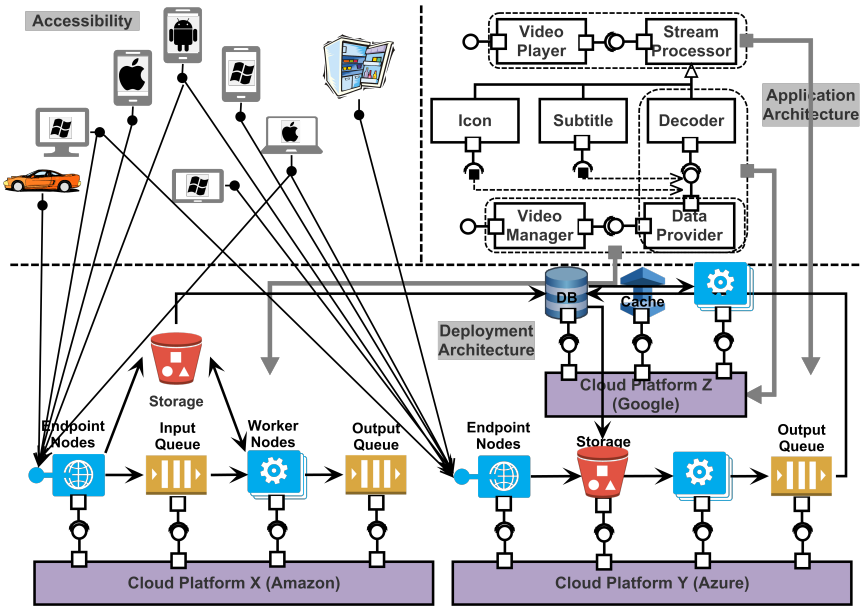


Figure 4. Application and deployment architecture of a video processing system.

configuration a difficult task. To match these requirements and dimensions, we identified 15 suitable patterns, reported in [23]. The key reasons behind a multi-cloud migration are already indicated in the situational description and process model captured in the organization, application and platform profiles:

- Location. Users are widely distributed around multiple data centers.
- Legal. Country regulations limit options for storing data in specific data centers.
- Architecture and Platform. Circumstances require public clouds to be used jointly with on-premises resources.
- Technical Constraints. Cloud-based applications must be resilient to the loss of a single data center or cloud provider.

To address the challenges identified here and allow to guide the architecture migration process, we define an orthogonal variability model, as we explain below.

4.2. *Variability Models*

As the first step to translate the profiles and constraints from the previous section into an architecture-oriented migration plan, we define *variability models* to

- enable users to configure functional and non-functional aspects of the application – an *application model* for application developers and users;
- enable users to choose their preferences of accessibility options – an *access model* for cloud operators and users;
- capture cloud providers' visible options for deployment – a *platform model* for cloud operators and users.

Three individual variability models, that have their origin in the description and process model and its profiles, can be identified, see Figure 1. The first two capture an external perspective that frames the migration implementation:

- The *Application Model* is based on the Application Profile, covering technical constraints and application domain features, which define a functional specification of the system through choosing the options in the application variability model VM_{func}.

DOI: 10.1002/spe

- The *Access Model* is based on the Application Profile, covering usage characteristics, which allow to choose user accessibility options comprising multi-device and multi-platform capabilities in the accessibility variability model VM_{access}. The device used for accessing the application is central here.

The third model addresses an internal perspective, i.e., the core of the migration implementation:

- The *Platform Model* is derived from the Platform Profile and the Application Profile, covering the technical characteristics. It allows to, firstly, select alternatives for realizing/deploying the application on a (multi-)cloud platform and, secondly, to select non-functional preferences of the system through the platform variability model $VM_{platform}$.

These models address a variety of concerns that we can categorise as follows: (i) QoS concerns (elasticity, availability), (ii) resource type (compute, storage, network), (iii) architecture patterns (pipes&filters, cache, etc.) and (iv) platform provider (AWS, Azure, OpenStack, etc.). These concerns can also be organised into two dimensions. The external dimension includes availability as a quality concern, but also external resources such as network, storage and databases. The internal dimension focuses on the platform with its compute resources, for instance elasticity as a quality objective and architectural patterns as orchestration options. Table I illustrates three deployment configurations classified along the variability model.

Table I. Deployment configurations.

Variation point		Configuration 1	Configuration 2	Configuration 3
External	Availability	Standard	Standard	High availability
	Bandwidth	1000	1000	10
	Storage	Multiple-instance	Multiple-instance	Geo-specific, single instance
	DB	SQL	SQL	No-SQL
Internal	Platform	Azure	AWS	Azure/AWS/Google
	Compute	Multiple-instance	Multiple-instance	Multiple-instance
	Elasticity	Auto-scale	Auto-scale	Auto-scale
	Pattern	Pipes and filters	Pipes and filters	Cache-aside, Pipes & Filters

We combine the three different variability models at different levels of abstraction for the three above-mentioned concerns into a single model that gives strong support to all consumers involved in cloud deployment. As illustrated in Figure 5, the approach allows users to (i) define a functional specification of the system through choosing the options in the application variability model (VM_{func}). Also, it allows users (ii) to select from architectural alternatives for realizing and deploying the application on a (multi-)cloud platform as well as selecting the non-functional preferences of the system in the platform variability model ($VM_{platform}$). Such aspects themselves affect some internal non-visible aspects of the system (red triangles in Figure 5), which are only visible for the development team of the cloud-based application. This will be realized by combining several valid cloud configurations to fit the requirements. (iii) To choose the accessibility options that are required by the users comprising multi-device and multi-platform capabilities in the accessibility variability model (VM_{access}). Therefore, our joint model consists of three individual variability models at different levels addressing different concerns of multi-cloud application deployments. VM_{func} is a fully fledged variability model that represents both functional commonalities and variabilities of the cloud-based software products.

4.3. Variability Model Properties

The variability model exhibits a number of interesting properties. The $VM_{platform}$ and VM_{access} models represent only variabilities that determine the non-functional aspects of the cloud-based products. They represent, in other words, a reference point to where different variants regarding the platform deployment options or accessibility can be attached. The variants manifest a concrete variability in terms of deployment or accessibility. In this model, all variation points in $VM_{platform}$

DOI: 10.1002/spe

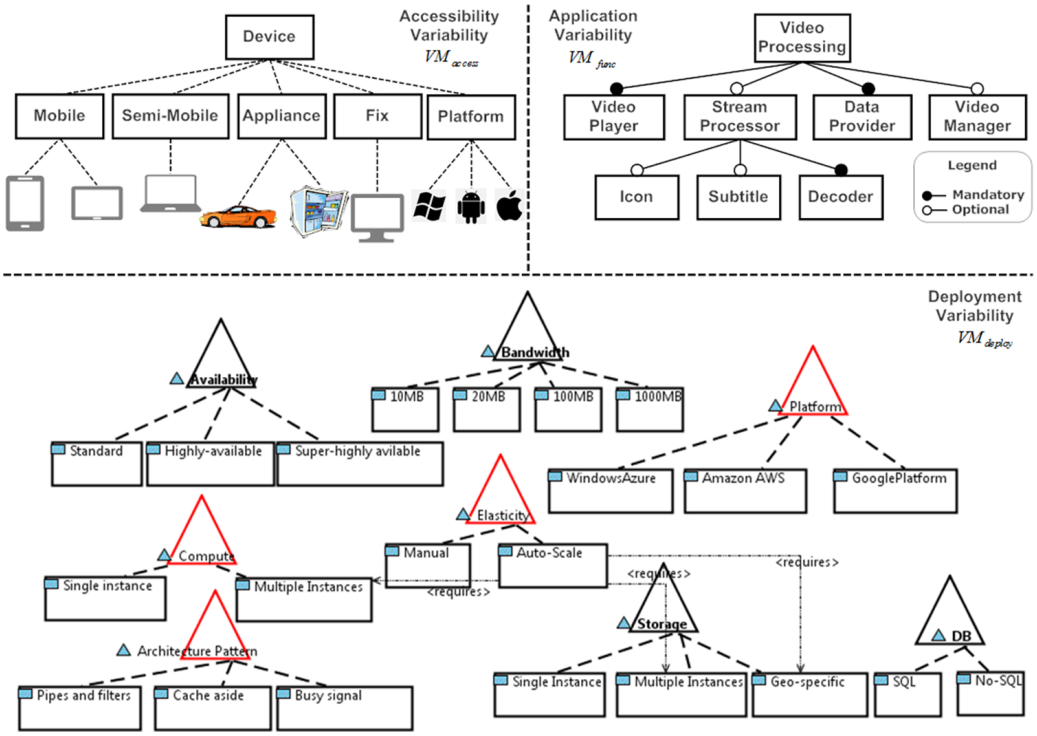


Figure 5. Orthogonal variability model showing the accessibility-driven, application-driven and platform-driven variability.

and VM_{access} are related to at least one functional variant in VM_{func}, and all variations in VM_{func} are related to at least one variation point in either $VM_{platform}$ or VM_{access}. This reduces the complexity of the variability model and therefore enhances the readability of the model facilitating a robust application customization.

For defining VM_{func} and VM_{access}, we utilise a feature model defined in [26], while for specifying $VM_{platform}$, we employ the OVM (Orthogonal Variability Model) introduced in [3]. The reason behind this choice for $VM_{platform}$ is that the OVMs are smaller and less complex since they only model variability and not the commonalities. This is useful in the context of multi-cloud environments since for modeling the deployment space we only need to consider different variability that each platform may offer and not thinking about their commonalities.

The faceted variability model distinguishes different roles: application developers (Dev), cloud operation experts (Ops) and the consumers (Con). Application developers provides the functional variability and commonality points of the system, resulting in the corresponding variability model (i.e., VM_{func}). Cloud operation experts are involved in the platform specific descriptions. They describe cloud platform variability and commonality points, thus providing the corresponding variability model (i.e., $VM_{platform}$) to the architecture. Operations experts are also responsible for providing the accessibility variability model (i.e., VM_{access}). Consumers are all user groups involved in externally visible option (bold triangles in Figure 5) selection through such orthogonal variability models. Using this approach only requires having the role-specific knowledge to properly configure the cloud application and to cooperate to develop a migration solution.

We now briefly discuss the key model properties. We can reduce the complexity of the variability model as all variation points in $VM_{platform}$ and VM_{access} are related to at least one functional variant in VM_{func}. Furthermore, all variations in VM_{func} are related to at least one variation point in either $VM_{platform}$ or VM_{access}.

The 3-pronged variability model is loosely based on other established models, combined here to specifically address the cloud migration concerns:

DOI: 10.1002/spe

- $VM_{platform}$ is based on the OVM Orthogonal Variability Model [27], which allows us to use a simpler model that does not model commonality.
- VM_{access} and VM_{func} are based on [28].

4.4. Customisation and Selection Process

The earlier situational description and process model from Section 3 is here extended and detailed to focus more on the architectural aspects of migration. We need a customisation process – part of the last process stage of the situational migration plan definition:

- Variant determination [consumer focus]. Involving the determination of functional and non-functional aspects of the application and access dimensions.
- Platform [consumer focus]. Involving the merging of new consumer requirements with existing ones.
- Platform [provider focus]. Includes the use of quality annotations to choose the most appropriate configuration for the provider from the given options.
- Reconfiguration. Focuses on decisions regarding single vs. multiple instance mode and live migration.

The overall framework was shown in Figure 1 that illustrates how the profiles relate to the variability models. The focusing on architectural concerns by moving from constraints to profiles to variability models to patterns selection and application is evident from the models. Non-architectural concerns become constraints on the architectural process steps. The different roles indicate the responsibility – provider and consumer jointly for the platform configuration and deployment, the consumer for the application properties.

5. MIGRATION PATTERN FRAMEWORK

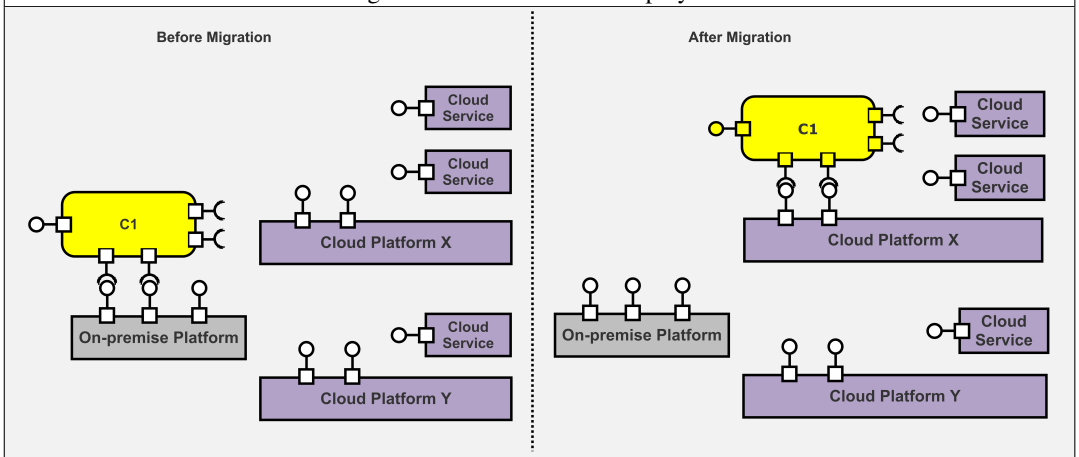
Architecture migration is a special step in the overall migration process that is organised around the application of migration patterns. The variability model helps to map the profiles and constraints from the process model onto the patterns. We first introduce the structure and content of architecture migration patterns and the multi-cloud deployment setting before providing a more comprehensive catalogue of patterns in the next section.

5.1. Migration Patterns in Multi-Cloud Setting

Our migration patterns are sequences of architectural changes (refactoring) in the application deployment setting, through which the current application is gradually modernized. For each migration pattern, an architectural migration schema has to be defined. A migration pattern is represented by an architecture diagram of the service architecture deployment before and after migration, i.e., a migration pattern is a transformation triple consisting of source and target architecture together with the applied pattern as the transformation specification. Each architecture is represented by well-defined architectural elements including services and connectors, deployment platforms (on-premise and cloud-based) and cloud services. The notation here is loosely aligned with UML component diagrams, with specific component types color-coded. A service component can either be atomic or contain internal components allowing for hierarchical decomposition. For example, the migration pattern MP1 below consists of a coarse-grained component that consumes services of an on-premise deployment platform. These can be coordination services that orchestrate different components in larger compartments or simply configurable IaaS resources providing required operating system or storage features. After migration, this component, instead of using on-premise platforms, uses services offered by a public cloud platform. Thus, the application component is re-deployed as-is on a cloud platform. The current architecture is mirrored in the cloud, but can take advantage of virtualization to not only reduce operational expenditure, but also to create multiple instances of the application to improve scalability and failover without increasing capital expenditure. The key risk is that underlying architecture issues are not addressed.

DOI: 10.1002/spe

Migration Pattern MP1: Re-deployment



Definition:	An application (component) is re-deployed (moved, re-hosted) as-is on cloud platform(s)
Problem:	Resource constraints limit scalability, Need to improve performance, Single point of failure, Reduce cost, Modernization
Solution:	Re-deploy on cloud environments, make use of elastic resources, multiple cloud deployment for failover and scalability.
Benefits:	Improved Backup and Failover, Coarse-grained scalability at application level, Simple coarse-grained re-deployment.
Challenges	Existing architecture constrains portability, deployment time/cost, scalability, integration may introduce complexity.

A monolithic legacy application in the cloud is still monolithic with limitations such as lack of scalability. Scalability is coarse-grained and cannot easily be achieved if, *e.g.*, the architecture does not allow the database to be updated by multiple instances.

In order to build highly scalable and reliable applications, a multi-cloud deployment is often appropriate. Our objective is to provide architectural guidance for migrating cloud-based systems that run on multiple independent clouds. Multi-cloud denotes the usage of multiple, independent clouds by a client or a service. A multi-cloud environment is capable of distributing work to resources deployed across multiple clouds [29]. A multi-cloud is different from a cloud federation – for the latter a set of cloud providers voluntarily interconnect their services to allow sharing of resources [29]. Hybrid deployment is a special case of multi-cloud where an application is deployed in both on-premise as well as cloud platforms.

Note that we primarily target Platform-as-a-Service (PaaS) clouds that provide middleware services to host and manage application services. PaaS clouds like Microsoft Azure, IBM Bluemix, or Cloud Foundry generally provide mechanisms to support the re-architecting activities described here.

5.2. Migration Pattern Selection

The technical factors that are captured in the deployment variability model are the cornerstone in the pattern selection process. Of key importance are: expected quality-of-service; the resources needed; the architecture patterns meant to be preserved or employed; and the platforms chosen to host the application in the cloud. These are reflected in the situational process profiles and need to be mapped to patterns. To identify a suitable pattern, the patterns are specified by pattern descriptions that focus on quality and resources/patterns to guide the selection process:

- Definition – provides a succinct architectural migration perspective
- Problem – refers to the quality concerns to be targeted by the pattern
- Solution – explains how the quality concerns are addressed at the architecture level
- Benefit – summarises the quality impact of the architecture solution
- Challenges – where the pattern can be applied (application constraint)

DOI: 10.1002/spe

We have identified 15 patterns in total through the empirical extraction process outlined in Section 2. To make the selection easier, these 15 patterns can be categorised into Core Patterns and Pattern Variants. Variants for the following core patterns can be found in [23]:

- Re-deployment (core pattern MP1): variant pattern MP2 (re-deployment in public cloud)
- Relocation (core pattern MP3): variant pattern MP4 (relocation for multi-clouds)
- Multi-Cloud Refactoring (core pattern MP5): variant patterns MP6 (hybrid refactoring), MP7 (hybrid refactoring with on-premise adaptation), MP8 (hybrid refactoring with cloud adaptation), MP9 (hybrid refactoring with hybrid adaptation)
- Multi-Cloud Rebinding (core pattern MP10): variant pattern MP11 (rebinding with cloud brokerage)
- Replacement (core pattern MP12): variant patterns MP13 (replacement with on-premise adaptation), MP14 (replacement with cloud adaptation)

Further variants can be added, but we will show the sufficient completeness of the given set to model common PaaS migration scenarios in the case study evaluation.

The core pattern and variant structure guides the migration pattern selection. Architecture (from the application and platform profiles) and the technical quality constraints are the starting selection criteria. The variability model then allows to define the pattern selection as a variability management problem in three dimensions that distinguishes internal (provider-based deployment) and external (application and application access) perspectives.

Some applications are integrated and support core business processes and services, but many of them support utility needs, are certainly non-core applications and are independent. The latter category may be obvious candidates for direct re-deployment. For the former integrated core ones, refactoring (re-architecting or redesigning) is more appropriate.

5.3. Cloud Architecture Migration Patterns

To obtain unambiguous pattern descriptions and to ground pattern-based migration planning in the description, process and variability models, we use a template-based definition of migration patterns. This definition is based on the semantics of architectural sachems before and after migration. In some migration patterns, it may only be possible to deploy application components in a public cloud. However, for those patterns that consider re-architecting, the application can be deployed in hybrid public/private platforms. Due to space limitations, we do not describe all patterns fully here – for more details refer to [23]. Only the core patterns are presented. The patterns missing from this list are variants of some core patterns (discussed earlier). The core patterns highlight the different construction principles for the cloud architecture: re-deployment, cloudification, relocation, refactoring, rebinding, replacement and modernization. The usability of the patterns in migration planning will be shown through a method engineering process in Section 6 and through a case study in Section 7.

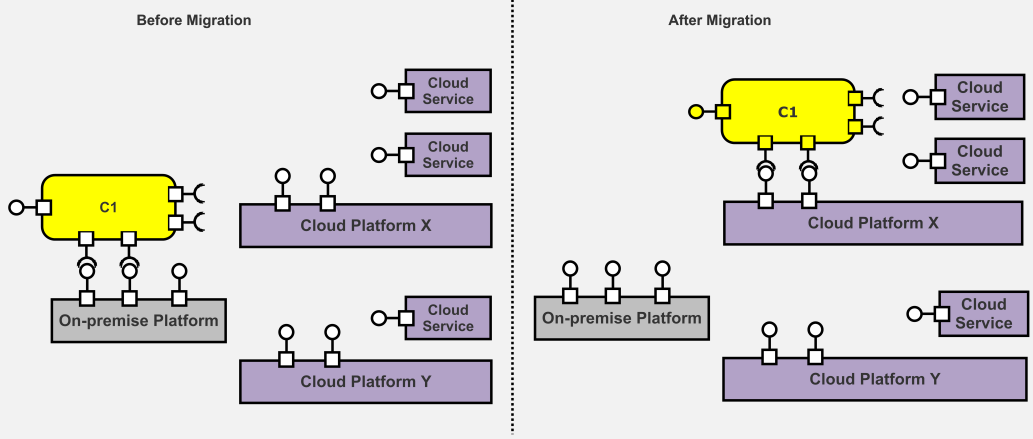
6. ASSEMBLY-BASED SITUATIONAL ARCHITECTURE MIGRATION

The description of the situational context through profiles and constraints leads to a selection of patterns that need to be assembled into a staged architecture migration process. In this section, we refine the ‘Define Migration Plan’ and ‘Perform Migration’ activities from Section 3.5.

Situational Migration and Assembly. To implement and perform a migration plan as a tractable process, appropriate building blocks have to be selected and combined. Migration patterns embed desirable principles for the target architectural deployment. Migration patterns represent fine-grained migration activities to be combined into a migration plan, ensuring that combined patterns do not violate pattern properties and the constraints imposed on the migration. For example, a pattern for the replacement of an on-premise component can be combined with a pattern for refactoring. This ensures that an architecture migration plan can be created incrementally, based on the situation

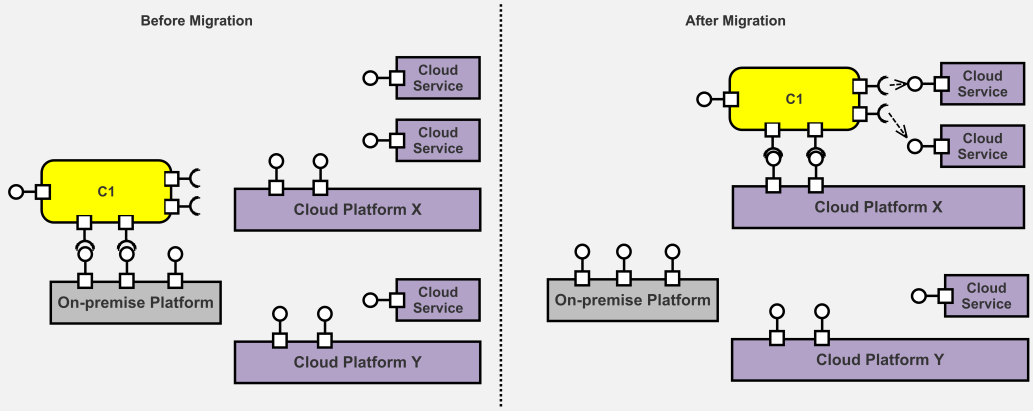
DOI: 10.1002/spe

Migration Pattern MP2: Cloudification



Definition:	Application hosted on-premise as-is but use public cloud services for added capabilities instead of on-premise ones.
Problem:	Need to improve reusability, extensibility, avoid redundancy 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.

Migration Pattern MP3: Relocation [see variant MP4]]

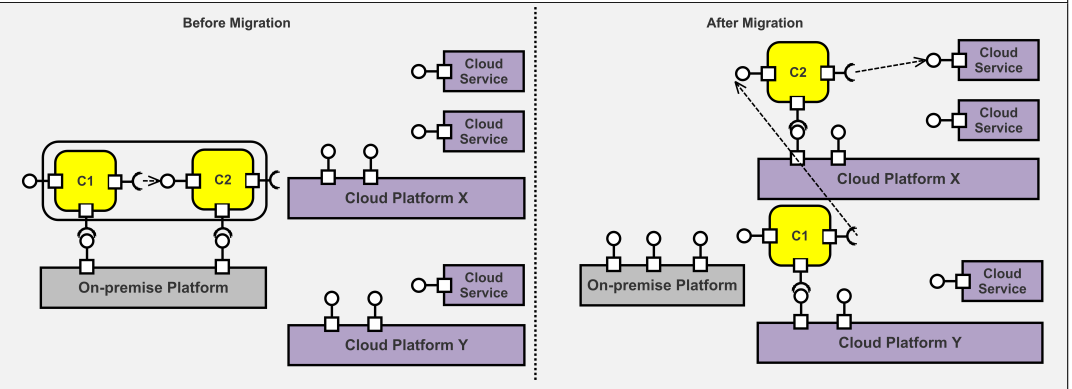


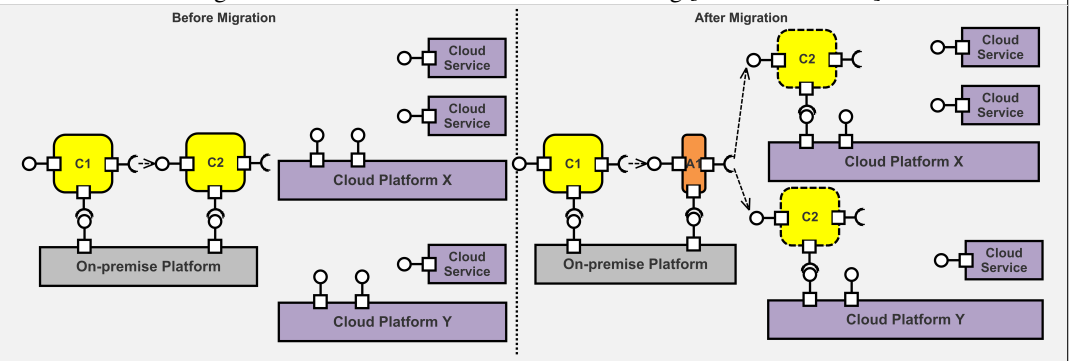
Definition:	Component re-deployed (relocated) on cloud platform is cloudified but without evolution in the application architecture.
Problem:	Enhance performance without significant architecture change, without capital expenditure for on-premise hardware.
Solution:	Use cloud services to improve throughput by leveraging Queues, Database partitioning/sharding, NoSQL, Cache
Benefits:	As component re-hosting in cloud and optimized performance.
Challenges	The type of application requests changes over time (for example, proportion of read-only calls reduces); cloud provider does not provide the necessary services to wrap the optimizations around the application without re-architecting.

reflected through our profiles and constraints. Figure 6 shows this pattern composition process. The patterns form a sequence of activities by which an application is gradually migrated and refined.

Assembling a Migration Graph. A migration transition graph provides a generic migration plan based on situations and possible migration patterns. The graph nodes are current architectural configurations and edges are migration patterns. The directed nature of the graph shows sequencing of patterns. Since multiple edges can enter a node, the model is able to represent many candidate

DOI: 10.1002/spe

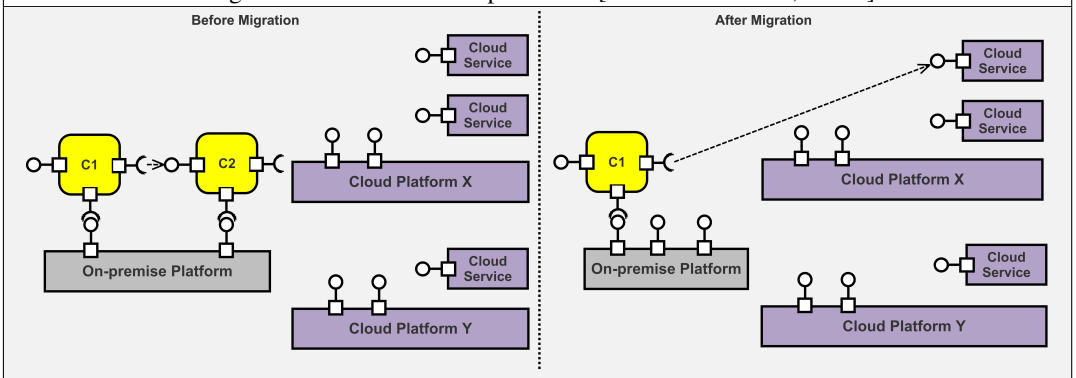
Migration Pattern MP5: Multi-Cloud Refactoring [see variants MP6, MP7, MP8, MP9]	
	
Definition:	An on-premise application is re-architected for deployment on cloud platform to provide better QoS.
Problem:	Coarse-grained applications are not agile enough to respond to requirement changes or variations in workload, and cannot take full advantage of the performance improvements that can be offered by cloud platforms.
Solution:	Application re-architected into fine-grained components; deployment of high-usage comp. optimized independently of low-usage ones; parallel design for better throughput to multi-cloud platforms; components as independent integrity units.
Benefits:	Optimal scalability/performance, range of multi-cloud deployment options, agility to respond to business/IT change.
Challenges	On-premise application is modernized in isolation; modernization is performed primarily for technical reasons; component architecture is only determined bottom-up and may need to be re-evaluated because of multi-cloud environment.

Migration Pattern MP10: Multi-Cloud Rebinding [see variant MP11]	
	
Definition:	A re-architected application is deployed partially on multiple cloud environments and enables the application to continue to function using secondary deployment when there is a failure with the primary platform.
Problem:	Failure such as a bug or configuration error that impacts cloud services may cause a failure to a cloud platform.
Solution:	Architecture for resilient systems (routes users to closest data center) used for failover: monitor services, if unavailable, traffic is routed to healthy instances. On-premise adapter (bus or load balancer) provides integration of components
Benefits:	As unhealthy services become healthy again, traffic can be delivered, returning system responsiveness to maximum.

plans. There are initial and target architectures, but also intermediate application architectures. Migration plans are a set of consequential triples <source config, pattern, target config> each of which corresponds to a migration step to achieve the target configuration from a specific configuration following a particular pattern. Note that one path from the source configuration (current on-premise application architecture) to the target (multi-cloud application architecture) will be chosen.

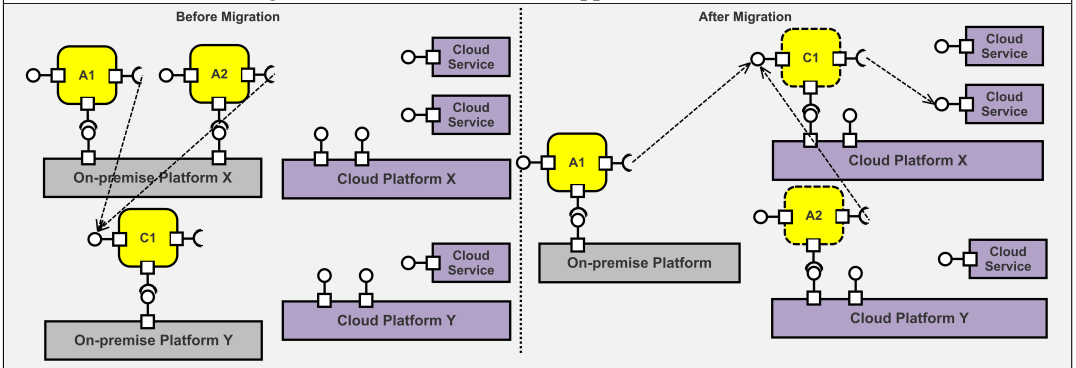
DOI: 10.1002/spe

Migration Pattern MP12: Replacement [see variants MP13, MP14]



Definition:	Individual capabilities in a re-architected solution are re-provisioned rather than re-engineered.
Problem:	Some existing components provided by current application are not the best alternative to meet business requirements.
Solution:	Analyze and identify capabilities to be replaced by cloud services (capabilities that can be supported by re-architected system), identify alternative cloud services with benefit over re-engineering of current capability to replace components
Benefits:	The solution is improved though best-in-class cloud services, Re-engineering costs and effort are saved.
Challenges	Cloud services presume specific communication protocol that make the replacement a challenging tasks.

Migration Pattern MP15: Multi-Application Modernization



Definition:	Different on-premise applications A1/A2, C1 are re-architected as a portfolio and deployed on cloud environment.
Problem:	The re-architecting of on-premise applications in isolation does not remove inconsistencies in data or duplicated functionalities, nor reduce the cost of their combined operation or maintenance.
Solution:	Current applications are analyzed jointly to identify opportunities for consolidation/sharing. Separation of service and solution architecture enables the identification of components (capabilities) that are shared by more than one solution.
Benefits:	Consistent information / rules in shared components, Reduced operation / maintenance costs for shared components, Challenges: Lack of business commitment to shared capabilities.

Mapping Patterns. Table II shows the pattern collection as a mapping of migration patterns and concerns for which they are suitable – these concerns are constraints derived from the profile determination. The patterns can be used, guided by the constraints, to form a plan (see Figure 6). This mapping is used to narrow down the related patterns and we can select the final pattern by comparing the situation through the benefit part in the pattern template. The selected patterns can be integrated based on the presence/absence of overlaps between patterns. The flexibility of this approach is restricted only by the set of available migration patterns. The patterns can be

DOI: 10.1002/spe

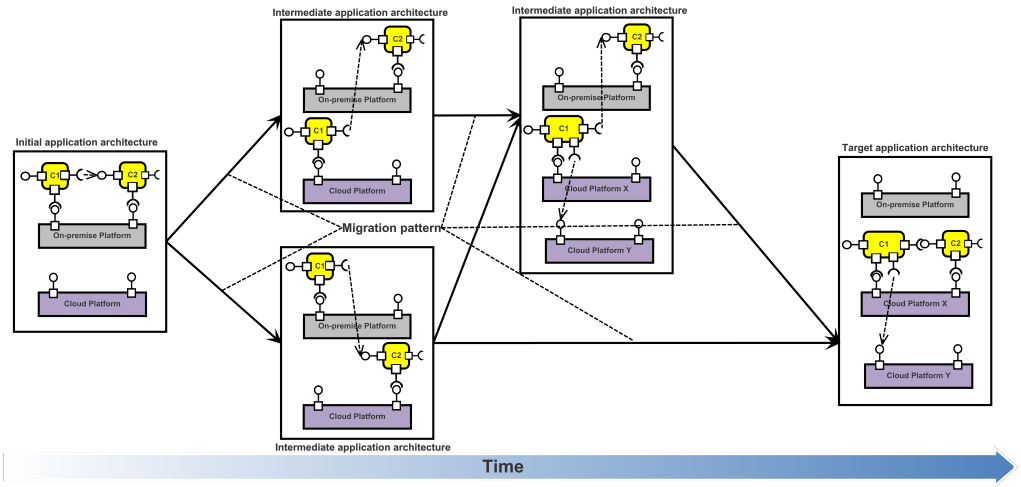


Figure 6. Migration transition graph.

extended over time, *e.g.*, by integrating a new solution to new problems at hand. For a more detailed description of the assembly-based approach, see the supplementary material in [23].

Table II. Cloud migration pattern selection

Objective	MP1	MP2	MP3	MP4	MP5	MP6	MP7	MP8	MP9	MP10	MP11	MP12	MP13	MP14	MP15
Pattern Category	Rh	Cl	Rl		Rf					Rb		Rp			Mo
Time to market	Y	-	N	N	N	-	-	-	-	N	N	Y	Y	Y	Y
New capabilities	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-
Reduce operational cost	Y	Y	-	-	N	-	-	-	-	N	N	Y	Y	Y	Y
Leverage investments	Y	Y	-	-	-	Y	Y	Y	Y	Y	Y	N	N	N	Y
Free up local resources	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Scalability	N	-	-	-	Y	Y	Y	Y	Y	Y	Y	-	-	-	-
Operational efficiency	Y	-	-	-	Y	-	-	-	-	Y	Y	Y	Y	Y	-

with Rh:Re-host, Cl: Cloudification, Rl: Re-location, Rf: Re-factor, Rb: Re-binding, Rp: Re-placement, Mo: Modernisation pattern categories.

7. CASE STUDY AND VALIDATION

The usability of the migration patterns in the V-PAM method shall be validated through a case study. We use a sample migration project based on our work with Microsoft Azure as a PaaS cloud for illustration and validation [30]. This project acts as a representative for a range of migrations we examined (and for the latter two categories also implemented). These include several CRM systems (*e.g.*, larger configurations based on commercial products), online retail solutions and services utilizing cloud storage solutions. Usability refers to the suitability of the pattern set to provide options and facilitate staged migration plans. Thus, we need to demonstrate two important properties: firstly, the utility of all patterns applied in the migration process and, secondly, also that the set is sufficiently complete to model a range of cases.

7.1. Case Study Setting

Company Context – [Organization Profile]. A financial services company decides to migrate in-house applications to the cloud. It uses Microsoft technologies, but it also has legacy systems deployed on UNIX. Some applications have external ports, while others are exclusively for internal use. The importance of the applications ranges from marginal to critical. A significant portion of the IT budget is spent on maintaining applications with marginal importance.

Software Application – [Application Profile]. The migration starts with the Expense application. This allows employees to submit and process expenses and request reimbursements. Employees

DOI: 10.1002/spe

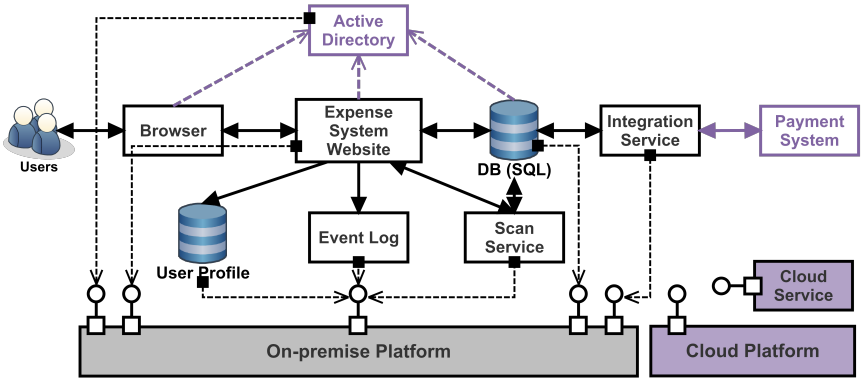


Figure 7. Application architecture before migration to the cloud.

can tolerate occasional hours of downtime, but prolonged unavailability is not acceptable. Most employees submit expenses within the last days before the end of each month, causing high demand peaks. The infrastructure for the application is scaled for average use only. The application is deployed on-premise. It requires high volume storage because most stored receipts are scanned.

An objective is to improve the user experience. Some applications vary in usage (*e.g.*, used once every two weeks, like salary-wages, but rarely at other times) – a usage concern. The company would benefit from the cloud-based increased responsiveness during peak times – a technical concern.

Platform Concerns [Platform Profile]. New applications take long for deployment, causing problems with adapting to changes. For any application, requirements must be analyzed, procurement processes must be initiated and networks must be configured. The existing platform is used inefficiently. The majority of servers are underutilized. It is difficult to deploy new applications with the required SLA to the existing hardware. Applications in a public cloud platform can take advantage of economies of scale and have automated processes for managing.

An objective is to expand ways to access applications. Applications located in the public cloud are available over the Internet, but authentication concerns exist. Another goal is portability, i.e., it can be moved between a public cloud platform and a private data center without modification to application code or operations. Furthermore, a tractable migration plan to the cloud platform is essential.

7.2. Migration Plan

Expense is an ASP.NET application. It uses Windows authentication for security. To store user preferences, it relies on ASP.NET profile providers. Exceptions and logs are implemented with Enterprise Library's Exception Handling Application Block and Logging Application Block. It uses Directory Service APIs to query data. It stores information on SQL Server. Receipts are stored in a file system. The architecture is illustrated in Figure 7.

The migration plan follows the process defined in Section 3. The existing servers, networks, and associated systems such as power supply and cooling are managed by the company. We present a sequence of migration steps and decisions made to reach a tractable migration plan by adopting the presented patterns.

1. Move the application to a cloud platform unchanged providing infrastructure reliability and availability. Management costs for running the hosted operating system and OS licenses must be considered, but development costs can be reduced as applications do not need to be refactored. Migration patterns MP1, MP3, MP4 suit, of which MP1 was selected, because only copy-as-is to the cloud without need for environmental services required.
2. An alternative is to adapt Expense to run as hosted on a platform by an external partner. This would avoid costs of porting the application to a different system and reduces management

DOI: 10.1002/spe

cost. There is work involved in refactoring the application to run in cloud-hosted roles. MP5-MP11 can be selected, since the user profiles were to be kept on-premise. Pattern MP6 was selected because there was no need for any interface adaptation (as in MP7-MP9) or multi-cloud deployment (as in MP10 and MP11).

3. Abandon the own payment application and rent a typically more generic cloud service, which needs to be evaluated regarding security, performance, and usability. MP12, MP13, MP14 suit, but a need to integrate Expense with a Payment service favors MP13.
4. For an external hosting decision, data storage facilities offered by cloud platforms are required. Expense requires a relational database system and NoSQL storage to store receipt images. MP12 was selected as Azure SQL and Storage offerings meet requirements.
5. Remote applications need to be integrated with other cloud services and on-premise for data access and monitoring. A systems operation or authentication tool could be used for monitoring, requiring remote services to be integrated. MP7, MP8, MP9, MP12, MP13, MP14 can be selected. Due to a need for some adaptations, MP14 was selected.
6. Although only employees use Expense, the payment sub-system also used by other applications must always be available. MP10, MP11 can be selected, but if the development of failover rebinding is to be avoided, a broker as in MP11 is utilized (*e.g.*, to deploy the payment system on Amazon and keep a mirror on Azure to route requests in case of failure).
7. Value-added services from the cloud such as caching can maximize performance when retrieving data or can cache output, session state and profile information. MP3 was selected to accommodate these environmental services of the cloud provider.

Note that, as quality of the cloud platform deployment is the key concern here, the platform variability model $VM_{platform}$ is the key driver for pattern selection. It guides the selection of alternatives for deploying the application on a (multi-)cloud platform and also configuring non-functional preferences within the cloud platform. The access variability model VM_{access} points to selecting patterns facilitating remote access and possible interface adaptation (*e.g.*, MP7-MP9). The application variability model VM_{func} has not been utilized as the functional scope of the application has not been changed during migration.

Table III. The summary of the Expense system migration plan.

Migration Step	Platform Requirement	Chosen Patterns
1	Minimal code changes to application and familiarity with platform	MP1
2	Granular control of resource usage and opportunity for auto-scaling	MP6
3	Lower cost although some limitations on feature availability	MP13
4	Replacing on-premise storage with cloud offerings	MP12
5	Integration with cloud utility services	MP14
6	Highly available service replacement	MP11
7	Better user experience, improved efficiency, and load leveling	MP3

A possible migration path based on the patterns determined above is presented in Table III. The migration steps are illustratively represented in [23]. Depending on the concerns of an organization, different combinations of hosting, data store and cloud services are possible. For example, MP1 step 1 follows a gradual migration by adopting the hosting approach, but uses SQL Server hosted in a VM before moving to an Azure SQL Database. Using MP3 instead would take advantage of storage capabilities (table/blob storage) and caching instead of relational databases to improve performance early rather than late.

The result of applying the steps to the source architecture is the architecture provided in Figure 8.

7.3. Discussion of Use Case and Industrial Case Studies

For the migration plan for the Expense system we had several different requirements, but we were able to find a satisfactory set of patterns. Thus, the suitability requirement of the V-PAM method in this case is achieved and met by the fact that the selection and composition of the proposed patterns actually results in a satisfactory target architecture that meets organisational constraints, functional

DOI: 10.1002/spe

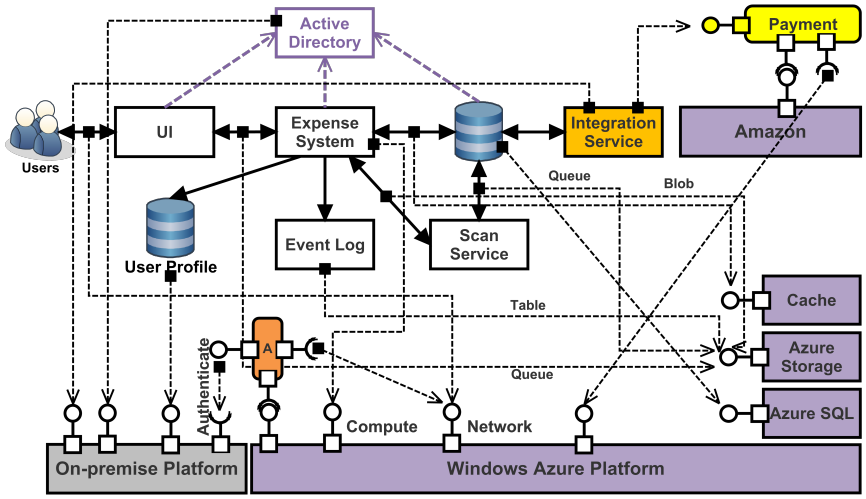


Figure 8. Application architecture after migration to the cloud.

application and non-functional platform requirements [8]. Technically, we can only conclude that the migration patterns are complete and useful for all situations arising from the use case.

However, we have analyzed and considered other migration projects, *e.g.*, different IaaS/PaaS/SaaS migration processes [10]. These include completed or ongoing migrations[†] of

- an e-commerce application with high availability and performance needs,
- a document processing system that needs a multi-cloud integration with ERP system components,
- a financial services application with a hybrid on-premise/cloud architecture and the need for integrated security management,
- components of an ERP system with the need for mobile access,
- sensor-based IoT-cloud integration solutions where sensor data is stored and analysed in the cloud.
- an more recently a commercial Mobile Backend as a Service (MBaaS) platform [31, 32].

These migration projects cover a range of application types, giving us certainty that a variety of application areas can be successfully covered through the proposed patterns.

- Distribution: we considered single public cloud, hybrid on-premise/public cloud and heterogeneous multi-cloud settings.
- Complexity: from cloud-based storage to multi-tier applications with 10 subsystems with more than 50 individual services.
- Applications: from traditional structure data-oriented transactional processing to high-volume, high-speed image processing and multi-tenant mobile applications that can handle billion transactions.
- Sectors: from software vendors to larger financial services and food sector.

Common to many of these is the need to integrate with different components of a distributed business process across heterogeneous multi-cloud settings, which highlights the need for multi-cloud pattern support. The V-PAM method has been employed in these projects particularly in the early stage, supporting feasibility studies and establishing initial migration plan. The profiles and patterns were

[†]See also further documentation at http://computing.dcu.ie/~pjamshidi/Materials/Files/ExpenseSystem-Case.pdf, http://computing.dcu.ie/~pjamshidi/Materials/Files/DAM-Case.pdf and http://ase.ce.sharif.edu/pubs/techreports/TR-SUT-CE-ASE-2015-01-Microservices.pdf

DOI: 10.1002/spe

found to be valuable to capture, analyse and compare different migration scenarios with different target architectures. The companies were SME, some software vendors, but also in the financial services or food sector. Despite some IT and development background, non had a comprehensive cloud background. Here a structured, systematic approach proved to be a valuable solution that has helped to deliver cloud solutions that meet the expectations and to keep the projects on track and avoid unnecessary delays.

A common problem during migration is the need to refactor the architecture if the aim is to fully benefit from cloud performance and flexibility promises. For instance, the storage refactoring options relating to relational, table and blob storage, that we investigated and documented in [30], are particularly addressed by patterns MP1 and MP3. There, we highlighted the re-architecting options that advanced PaaS clouds offer, but also showed that while quality concerns such as scalability or availability are covered, their quantification and a trade-off analysis with cost aspects is not covered. Often, which specific paths are chosen is driven by more in-depth quality concerns. Our solution focuses on functional architecture aspects and only includes quality and cost concerns qualitatively.

We can conclude from the use case detailed in this section and also other concrete migration projects we were involved, that the proposed V-PAM method is effective as it guides the process and gives particularly the companies in question assurance of a predictable process and allows for a guided and structured project execution. Furthermore, the variety of case studies we conducted established a relevance of the models (profiles, variability) and the sufficient completeness of the pattern catalogue.

8. RELATED WORK

We conducted a literature review [1] aiming to identify, taxonomically classify, and systematically compare the existing research focused on planning, executing, and validating migration of legacy systems towards cloud computing platforms based on earlier architecture evolution work [33]. We found a lack of repeatable and verifiable practices as one of the key reasons that cloud migration is not a fully mature domain. In the context of the Cloud-RMM migration framework [1], our work here can be categorized as a contribution to migration planning.

Cloud migration is a form of software modernisation [34]. As a consequence, it requires sound continuous development frameworks with methodologies and patterns, languages and tool support [35].

Cloud migration frameworks. Cloud migration approaches range from decision making to enabling legacy software migration with approaches reporting best practice, experience and lessons learned in between. Decision making for cloud adoption (*e.g.*, [21, 17, 36, 37]) is inherently complex and influenced by multiple factors, such as cost and benefits through migration [38]. In contrast, some approaches enable the actual migration of legacy software in terms of architectural adaptation (*e.g.*, [39, 40]). Some other work reports on lessons learned and best practices from real migration case studies (*e.g.*, [16, 24, 41]) — providing empirical evidence for further cloud migration research. Our work is complementary to those approaches, as none of them provides a variability-driven planning solution based on a constructive pattern catalogue, and which is particularly suited to support migration decisions targeting multi-cloud architectures.

Cloud migration patterns. A number of migration strategies and best practices have been suggested in terms of patterns in [25, 15, 26]. These are rather informal and do not consider a multi-cloud setting. The objective there was mainly classification of existing best practice into migration strategies. The key advantage and novelty of our work, more than a set of patterns, is the notion of assembly-based situational migration at the architecture level, specifically towards pattern-based migration planning for multi-cloud deployment. It enhances the state-of-the-art by proposing a tractable planning approach based on composable patterns.

DOI: 10.1002/spe

A pattern catalogue for cloud migration is proposed in [26, 11], but it differs from our approach in at least two important ways. First the patterns in their work is mainly development patterns that are useful for application developers, while the patterns that we proposed are migration patterns. The second difference is that they only proposed a pattern catalogue, while we went one step further by proposing a methodology for pattern selection and composition that altogether facilitate a systematic migration plan.

A direction similar to [26] is taken by Di Martino *et al.* [42]. They extend an existing ontology for design pattern description aiming at representing both classical design and cloud architecture patterns together with their architectural implementations. This provides an agnostic representation to define a mapping between design patterns and cloud patterns. However, their pattern notion is only applied to the architecture of cloud-based system. Our solution could be extended by applying these cloud patterns, and associated design patterns, to the source (i.e., non-cloud based) and target (i.e., cloud-based) architectures of our architectural migration patterns.

Differently from Di Martino *et al.* [42], Bruneliere *et al.* [43] suggest the TOSCA standard for Topology and Orchestration Specification for Cloud Applications to define cloud architectures. A language like CloudML [44] could also be utilised for the purpose of architecture description. Again, our solution could be extended to incorporate those architectural notations, specifically as a way to facilitate the description and deployment of our target cloud architectures through the specification of TOSCA orchestration plans.

Software modernisation and model-driven migration. The ARTIST project [43] introduces a migration framework defined by six dimensions, namely technical space, origin, purpose, architectural viewpoint, environment and size. These are similar to the properties we use to describe our patterns. An integration of our pattern description with the ARTIST dimensions could be made by extending our pattern properties with those dimensions not yet covered, such as size. Models play a central role to capture the essential structure and qualities of architectures in the ARTIST approach. This allows for model transformation techniques to be utilised as part of the cloud migration process [45]. In this regard, the ARTIST approach is more geared towards the execution and verification of actual migrations than our solution, which provides a cloud migration analysis and planning tool.

Models at runtime can be used to coordinate the continuous deployment of services in the cloud, as proposed by the MODACLOUDS project in combination with the CloudML language [44]. Again, we differ here in that automation of the deployment of the target architecture is not our priority. Rather, we focus on the analysis phase where the identification and then selection of migration architecture options and the definition of a migration plan are central.

Also based on a model core is the CloudMIG framework proposed by Frey and Hasselbring [46]. Like our solution, the need to address technical quality concerns like scalability is recognised. At the centre is a hierarchy-structured model that guides decision processes and determines the most appropriate migration strategies. While the two solutions coincide in their aim to define a migration plan (the process model in our case), our solution focuses on the combination of variability modelling and patterns to plan and reason about the required architectural transformations.

Microservice migration Microservices have been discussed recently as an architectural style suitable to design, deploy and manage services in the cloud [47, 31]. In [32], we have reported our experience on migration to microservices architecture. Based on a monolithic source architecture, an incremental stepwise migration approach to a microservices architecture was implemented. A system can be evolved in terms of three aspects, including re-architecting the current system, introducing new supporting components, and enabling Continuous Delivery using containerization in the context of DevOps [48]. Using a Situational Method Engineering migration approach proposed in [49], a monolithic source architecture can be migrated to a target microservices architecture through reusable migration patterns (see our initial catalogue of microservices migration patterns in [49]). This confirms our assumption that the pattern-based migration approach proposed here can be suitable for all kinds of service-oriented architectures.

DOI: 10.1002/spe

9. CONCLUSION AND OUTLOOK

We have presented a cloud migration method – V-PAM for Variability-based, Pattern-driven Architecture Migration – built around architecture change patterns, which allows planning the migration of applications for multiple cloud platform deployment. The introduction of migration patterns complements existing migration practices and allows for an engineering approach towards constructing transparent and verifiable migration plans. The migration patterns are reusable and composable architectural change patterns that we see as building blocks of an overall migration process, reflected through a migration plan as a sequence of pattern applications.

The migration process needs to rely on a combination of suitably selected and properly assembled patterns. In order to address this, we provided a framework that allows to capture the situational context through profiles and constraints. A three-dimensional variability model then facilitated the detailing and mapping of architectural concerns onto patterns. The variability models act as a link to manage migration variability through a product-line style approach.

Architecture-oriented patterns for multi-cloud settings are important for two reasons. Firstly, architectures are often refactored to adapt an application to the cloud platform, to benefit more from cloud characteristics such as elasticity or simply to modernize a legacy application. Secondly, applications often need to be integrated with other components as part of a larger business process in often multi-cloud environments. Our implicit assumptions here included the possibility to componentise legacy applications and also to target a cloud native architecture. Our brief discussion of microservices as a recently emerging cloud native architectural style demonstrates the importance of servitisation, but also the need to provide a framework that is generic enough to support the different service flavours in the context of DevOps.

Automation will play a major role in the further development of the method and will include the development of a migration pattern repository as a tool that facilitates migration planning as well as application of the patterns to new domains and migration cases. To demonstrate the usability and completeness of the patterns beyond business-oriented SaaS and standard PaaS-level services such as storage, currently we are in the process of evaluating others for migration planning in three cases with our industry partners. We also plan to formally represent the relations between migration patterns in order to form a pattern map and work toward a pattern language for migration practices.

ACKNOWLEDGEMENT

The research work described in this paper was in parts supported by the Irish Centre for Cloud Computing and Commerce (an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority). Nabor C. Mendonça is partially supported by Brazil's National Council for Scientific and Technological Development (CNPq), under grants 487174/2012-7 and 310611/2014-8.

REFERENCES

1. Jamshidi P, Ahmad A, Pahl C. Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing* 2013; **1**(2):142–157.
2. Fox A, *et al.*. Above the clouds: A Berkeley view of cloud computing. *Technical Report*, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Technical Report UCB/EECS 2009.
3. Idu RKA, Hage J, Jansen S. Legacy to SOA Evolution: A Systematic Literature Review. *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, Ionita AD, Litoiu M, Lewis G (eds.). IGI Global, 2012; 40–70.
4. van der Meulen R. What Data Center Architects Can Learn from Building Architects 2015. [Available Online at http://www.gartner.com/smarterwithgartner/what-data-center-architects-can-learn-from-building-architects/].
5. Wilder B. *Cloud Architecture Patterns: Using Microsoft Azure*. O'Reilly, 2012.
6. Jamshidi P, Pahl C, Chinenyeze S, Liu X. Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective. *10th International Workshop on Engineering Service Oriented Applications (WESOA)*, 2014.
7. Tran V, Keung J, Liu A, Fekete A. Application migration to cloud: a taxonomy of critical factors. *2nd International Workshop on Software Engineering for Cloud Computing (SECLOUD)*, ACM, 2011; 22–28.
8. Gholami MF, Sharifi M, Jamshidi P. Enhancing the OPEN Process Framework with service-oriented method fragments. *Software & Systems Modeling* 2014; **13**(1):361–390.

9. Mirbel I, Ralyté J. Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requirements Engineering* 2006; **11**(1):58–78.
10. Pahl C, Xiong H, Walshe R. A Comparison of On-Premise to Cloud Migration Approaches. *Service-Oriented and Cloud Computing*, *Lecture Notes in Computer Science*, vol. 8135, Lau KK, Lamersdorf W, Pimentel E (eds.). Springer, 2013; 212–226.
11. Fehling C, *et al.*. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014.
12. Beserra PV, Camara A, Ximenes R, Albuquerque AB, Mendonça NC. Cloudstep: A step-by-step decision process to support legacy application migration to the cloud. *IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, 2012; 7–16.
13. Jamshidi P, Pahl C. Orthogonal Variability Modeling to Support Multi-Cloud Application Configuration. *1st Workshop on Seamless Adaptive Multi-cloud Management of Service-based Applications (SeaClouds)*, 2014.
14. Pahl C, Xiong H. Migration to paas clouds – migration process and architectural concerns. *IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, 2013; 86–91.
15. Mendonca NC. Architectural options for cloud migration. *Computer* 2014; **47**(8):62–66.
16. Khajeh-Hosseini A, Greenwood D, Sommerville I. Cloud migration: A case study of migrating an enterprise IT system to IaaS. *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, 2010; 450–457.
17. Khajeh-Hosseini A, Greenwood D, Smith JW, Sommerville I. The Cloud Adoption Toolkit: Supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience* 2012; **42**(4):447–465.
18. Khajeh-Hosseini A, Sommerville I, Bogaerts J, Teregowda P. Decision support tools for cloud migration in the enterprise. *IEEE 4th International Conference on Cloud Computing (CLOUD)*, 2011; 541–548.
19. Chauhan MA, Babar MA. Migrating service-oriented system to cloud computing: An experience report. *IEEE 4th International Conference on Cloud Computing (CLOUD)*, 2011; 404–411.
20. Mohagheghi P, Sæther T. Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the REMICS project. *IEEE 7th World Congress on Services (SERVICES)*, 2011; 507–514.
21. Saripalli P, Pingali G. MADMAC: Multiple attribute decision methodology for adoption of clouds. *IEEE 4th International Conference on Cloud Computing (CLOUD)*, 2011; 316–323.
22. Cunha M, Mendonça N, Sampaio A. Investigating the impact of deployment configuration and user demand on a social network application in the amazon EC2 cloud. *IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011; 746–751.
23. Jamshidi P, Pahl C. Cloud Migration Patterns – Supplementary Material. [Available Online at http://www.computing.dcu.ie/~pjamshidi/Materials/CMP.html].
24. Andrikopoulos V, Binz T, Leymann F, Strauch S. How to adapt applications for the cloud environment. *Computing* 2013; **95**(6):493–535.
25. Wilkes L. Application Migration Patterns for the Service Oriented Cloud 2011. [Available Online at http://everware-cbdi.com/ampsocl].
26. Fehling C, *et al.*. Service Migration Patterns – Decision Support and Best Practices for the Migration of Existing Service-Based Applications to Cloud Environments. *IEEE 6th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2013; 9–16.
27. Pohl K, Böckle G, van der Linden FJ. *Software product line engineering: foundations, principles and techniques*. Springer, 2005.
28. Kang KC, Lee J, Donohoe P. Feature-oriented product line engineering. *IEEE software* 2002; **19**(4):58–65.
29. Grozev N, Buyya R. Inter-Cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience* 2014; **44**(3):369–390.
30. Xiong H, Fowley F, Pahl C, Moran N. Scalable Architectures for Platform-as-a-Service Clouds: Performance and Cost Analysis. *European Conference on Software Architecture (ECSA)*, Springer, 2014; 226–233.
31. Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software* 2016; **33**(3):42–52.
32. Balalaie A, Heydarnoori A, Jamshidi P. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. *1st International Workshop on Cloud Adoption and Migration (CloudWay)*, 2015.
33. Jamshidi P, Ghafari M, Ahmad A, Pahl C. A framework for classifying and comparing architecture-centric software evolution research. *17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2013; 305–314.
34. Menychtas A, *et al.*. Software modernization and cloudification using the ARTIST migration methodology and framework. *Scalable Computing: Practice and Experience* 2014; **15**(2):131–152.
35. Ferry N, *et al.*. Continous Deployment of Multi-cloud Systems. *Proc. of the 1st International Workshop on Quality-Aware DevOps (QUDOS)*, 2015; 27–28.
36. Frey S, Hasselbring W, Schnoor B. Automatic conformance checking for migrating software systems to cloud infrastructures and platforms. *Journal of Software: Evolution and Process* 2013; **25**(10):1089–1115.
37. Andrikopoulos V, Darsow A, Karastoyanova D, Leymann F. CloudDSF - The Cloud Decision Support Framework for Application Migration. *Service-Oriented and Cloud Computing*, *Lecture Notes in Computer Science*, vol. 8745, Villari M, Zimmermann W, Lau KK (eds.). Springer, 2014; 1–16.
38. Misra SC, Mondal A. Identification of a companys suitability for the adoption of cloud computing and modelling its corresponding return on investment. *Mathematical and Computer Modelling* 2011; **53**(3):504–521.
39. Kwon YW, Tilevich E. Cloud Refactoring: Automated Transitioning to Cloud-Based Services. *Automated Software Engineering* 2014; **21**(3):345–372.
40. Vasconcelos M, Mendonça NC, Maia PHM. Cloud Detours: A Non-intrusive Approach for Automatic Software Adaptation to the Cloud. *Service Oriented and Cloud Computing*, *Lecture Notes in Computer Science*, vol. 9306, Dustdar S, Leymann F, Villari M (eds.). Springer International Publishing, 2015; 181–195.

DOI: 10.1002/spe

41. Maenhaut PJ, Moens H, Ongenae V, De Turck F. Migrating legacy software to the cloud: approach and verification by means of two medical software use cases. *Software: Practice and Experience* 2016; **46**(1):31–54.
42. Di Martino B, Cretella G, Esposito A. Semantic and agnostic representation of cloud patterns for cloud interoperability and portability. *Proc. of the IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2013; 182–187.
43. Bruneliere H, *et al.*. Software modernization revisited: Challenges and prospects. *Computer* 2015; **48**(8):76–80.
44. the evolution of cloudml and its manifestations.
45. Burgueno L, *et al.*. Static fault localization in model transformations. *IEEE Transactions on Software Engineering* 2015; **41**(5):490–506.
46. Frey S, Hasselbring W. The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. *International Journal on Advances in Software* 2011; **4**(3 and 4):342–353.
47. Newman S. *Building Microservices*. O'Reilly, 2015.
48. Brunnert A, van Hoorn A, Willnecker F, Danciu A, Hasselbring W, Heger C, Herbst N, Jamshidi P, Jung R, von Kistowski J, *et al.*. Performance-oriented devops: A research agenda 2015; .
49. Balalaie A, Heydarnoori A, Jamshidi P. Microservices Migration Patterns. *Technical Report*, Sharif University of Technology, Technical Report No. 1, TRSUT-CE-ASE-2015-01 2015. [Available Online at http://arminb.me/microservices/report.pdf].

DOI: 10.1002/spe