A Framework for Architecture-driven Migration of Legacy Systems to Cloud-enabled Software

Aakash Ahmad[1], Muhammad Ali Babar[2, 1]

[1]IT University of Copenhagen, Denmark

[2]University of Adelaide, Australia

[1]aahm@itu.dk, [2]ali.babar@adelaide.edu.au

ABSTRACT

With the widespread adoption of cloud computing, an increasing number of organizations view it as an important business strategy to evolve their legacy applications to cloud-enabled infrastructures. We present a framework, named **Legacy-to-Cloud Migration Horseshoe**, for supporting the migration of legacy systems to cloud computing. The framework leverages the *software reengineering* concepts that aim to recover the architecture from legacy source code. Then the framework exploits the software evolution concepts to support architecture-driven migration of legacy systems to cloud-based architectures. The Legacy-to-Cloud Migration Horseshoe comprises of four processes: (i) *architecture migration planning*, (ii) *architecture recovery and consistency*, (iii) *architecture transformation* and (iv) *architecture-based development of cloud-enabled software*. We aim to discover, document and apply the *migration process patterns* that enhance the reusability of migration processes. We also discuss the required tool support that we intend to provide through our ongoing work in this area.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Evolution

General Terms

Software Architecture, Reengineering, Maintenance and Evolution

Keywords

Reverse Engineering and Evolution, Cloud Computing, Service-driven Architecture, Architecture-centric Software Migration.

1. INTRODUCTION

Cloud computing has emerged as a platform that allows organizations to leverage the distributed and interoperable services to deploy their software systems over publically available resources [1, 2]. From a business point of view, organizations can benefit from the pay-per-use model offered by cloud services rather than an upfront purchase of costly and over-provisioned infrastructure [2]. From a technical perspective, the scalability, interoperability, and efficient (de-)allocation of resources through cloud services can enable a smooth execution of organizational operations [7]. Given the potential benefits of cloudification, an increasing number of organizational business-critical applications – so called *legacy systems* [4, 5] – are being migrated to cloud computing.

WICSA '14, April 07 - 11 2014, Sydney, NSW, Australia.

http://dx.doi.org/10.1145/2578128.2578232

One significant example is the migration of oracle client/server software[1] to cloud-based (hardware and software) infrastructure [10]. The trend to migrate the legacy systems to cloud computing has highlighted the importance of providing processes, patterns and frameworks for supporting a systematic migration of legacy application to cloud systems [2, 11].

Architecture-centric software evolution is considered as a successful solution to accommodate the changing requirements in existing software while maintaining a global (component-connector) view of the system [12, 19]. Software architecture represents a system's structure and behaviour as hierarchical configurations of computational components and their interconnections by abstracting the implementation specific details [4, 6]. By following ISO/IEC 14764 standard for software maintenance [12, 24], we consider migration as an *adaptive evolution* and exploit software architecture abstraction to migrate legacy systems to cloud-enabled software. ***Cloud-based architectures*** utilize the software as a service (SaaS) model to develop systems that are elastic to frequent variations in their operational environments [1]. In contrast to the more traditional (object-oriented and component-based) architectures [6], cloud architectures are characterized by a set of (service) elasticity requirements. According to Herbst and colleagues [7], elasticity in cloud systems is defined as "*the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible*". To meet the demands for an autonomic resource provisioning and de-provisioning, elasticity is characterized by:

(1) *Scalability:* a system's ability to robustly control the workload fluctuations corresponding to acquiring and releasing resources.

(2) *Cost-Time efficiency:* a system acquiring required resources and releasing unutilised resources in a time and cost efficient manner.

Therefore, during legacy systems migration, the evolved architecture must satisfy the elasticity requirements associated with cloud services. A recent study of Capegemini – based on interviews with 460 business and IT executives – has concluded that migration of legacy applications to cloud-based system is driven by the organisational needs to achieve business agility and cost efficiency [14]. Our work on systematically investigate the state-of-research on legacy migration to service-oriented and cloud systems [11] revealed that there do not exist any solution that provides the necessary processes and tool support for architecture-driven migration of legacy applications to cloud-enabled software. Our focus on architecture-driven migration was motivated by the potential benefits such as (i) abstracting source code refactoring with a component and connector-based view of evolution [19], (ii) preserving the design rationale (architecture styles and patterns) from legacy to cloud during transformation process [21], and (iii) exploiting architecture model as a blue-print for target code generation [23].

[1] Oracle Client/Server Architecture http://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch20.htm

In order to respond to the needs of systematic support for migration of legacy systems to cloud computing [11], we propose a framework that supports a process-driven and incremental migration of architectures of legacy systems to cloud-based service architectures. By incremental migration, we mean a *decomposition of the coarse-grained architectural migration process into a set of fine-grained sub-processes and activities that can be tackled in a stepwise manner to enable architectural migration*. We have extended the classical *reengineering horseshoe model* [8] and the OMG's *Architecture Driven Modernization* (ADM) framework [5] to support migration to clouds. The proposed framework provides a round-trip engineering by taking the legacy source code as an input and architecturally transforming it to cloud-enabled code. The process can be repeated on the target (cloud-enabled) code for refinements. The main objective of this research is:

To provide a framework as a set of migration processes and their underlying activities to support an incremental migration – exploiting architectural abstractions – of legacy systems to cloud-enabled infrastructure. Our planned tool support for migration shall provide customisation and (semi-) automation of the migration processes.

Solution Overview: In the proposed solution, we unify the concepts of software reengineering and software evolution to *identify* the architecture of a legacy system from source code, *transform* the legacy architecture towards a cloud service architecture and *derive* the cloud-enabled target code from transformed architecture. An overview of the proposed framework, named **Legacy-to-Cloud Migration Horseshoe**, is shown in Figure 1 that comprises of four processes: (i) *architecture migration planning*, (ii) *architecture recovery and consistency*, (iii) *architecture transformation*, and (iv) *architecture-based development of cloud-enabled software*. The migration process can be repeated on the cloud-enabled code for future refinements in architecture and code.

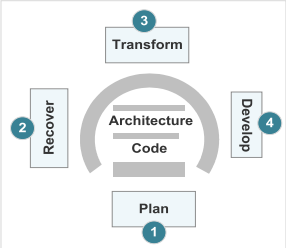

Figure 1. Overview of the Proposed Solution

The primary contributions of this research are:

- Extending the traditional architectural re-engineering and modernisation models to develop a framework that provides a process-driven solution for legacy migration to clouds.
- The proposed solution argues about the needs for architectural migration planning process that drives the migration processes which is lacking in the existing frameworks.
- Establishing round-trip process engineering for architecture-driven migration that serves as a reference model to develop appropriate tool support for automation and customization of the legacy migration.

The rest of this paper is organized as follows. An overview of the related research to justify the proposed contributions is presented in Section 2. The proposed framework is presented in Section 3. Conclusions and future research are discussed in Section 4.

2. STATE-OF-RESEARCH ON LEGACY MIGRATION TO CLOUD

Software migration is not a new concept; in 1970 the ISO/IEC 14764 standard for software maintenance [24] classified migration as a type of adaptive maintenance and evolution – adapting existing software to its new requirements [12]. In recent years, there is a steady growth of research [14, 15] and practices [10] on legacy migration to cloud computing platforms. To investigate a collective impact of the existing research, we conducted a systematic review of research on legacy migration to clouds that highlighted the needs for processes and patterns that enable a systematic cloudification of existing on premise legacy software to clouds [11]. To justify our contributions, we provide an overview of some of the related ongoing projects (Section 2.1), research and practices on legacy migration to clouds (Section 2.2) and discuss existing frameworks for architectural migration (Section 2.3).

2.1 Ongoing Projects for Legacy Migration

A number of ongoing projects [25, 26, 17] are focused on provisioning and migration of legacy systems to cloud based environments (Infrastructure as a Service (IaaS) and Platform as a Service (PaaS)).

More specifically, **REMICS**[2] is an FP7 project that aims to provide reusability and migration of the legacy systems to cloud systems [25]. It exploits the model-driven techniques to transform the legacy model to cloud services. It is vital to mention about the **CloudMIG**[3] initiative that aims at supporting SaaS providers to semi-automatically migrate existing enterprise software systems to scalable and resource-efficient PaaS and IaaS-based applications [26]. These two projects are similar in their objectives to our proposal, however; based on our experience with migration projects [13, 15] we believe that migration to cloud is more than just the evolution of software. Legacy migration entails some critical decisions including but not limited to decisions about cloud providers, cost-risk analysis, legal and cultural issues. Therefore, in contrast to [25, 26] in our framework we argue about the needs for a migration planning process that drives architectural migration. We aim to accommodate the pre-migration activities and decision (stakeholders view, technological trade-offs etc.) to develop a migration plan as an for migration process that is not supported in existing approaches.

The **ARTIST**[4] project aims to facilitate with the planning, designing, implementation and validation of the automated evolution of legacy software to SaaS and the Cloud Computing delivery models [27]. In contrast to the migration of software modules in [27], our framework is limited to architectural-driven legacy migration. We focus on abstracting source code level details and to preserve the required properties of legacy systems with architecture-driven transformation. ARTIST project is more comprehensive than our approach (by including the deployment and certification processes) that ultimately requires more efforts (cost-time) to enable migration.

2.2 Process Models for Legacy to Cloud Migration

In [11], the authors have presented the *Cloud Reference Migration Model* (Cloud-RMM) that comprises of three process (i) Migration Planning, (ii) Migration Execution and (iii) Migration Evaluation along with a set of cross-cutting concerns affecting these processes. This migration model is developed based on a systematic review of the literature and represents a theoretical reference to the existing research on planning, executing and evaluating legacy migration to clouds. In contrast to Cloud-RMM model, the proposed solution is specifically focused on architecture-driven migration. In addition, the proposed cloud migration horseshoe provides a round-trip approach for migrating legacy source code to cloud-enabled code that is not supported with Cloud-RMM. The proposed round-trip approach enables a continuous refinement of the cloud-enabled code and architecture for future needs.

Framework(s) and Experience Report(s) for Migration to Cloud: In [15], the authors have presented a process framework to support migration of legacy software to cloud computing platforms. The authors

[2] REuse and Migration of legacy systems to Interoperable Cloud Services. http://www.remics.eu/

[3] CloudMIG Xpress. http://www.cloudmig.org

[4] Advanced software-based seRvice provisioning and migraTIon of legacy SofTware. http://www.artist-project.eu/approach

have developed a framework based on their experiences from migrating a legacy open source system (OSS) to two different cloud computing platforms. In the migration framework, the authors have presented a total of seven distinct activities for legacy migration focused on identification of requirements for migration, evaluating the alternative cloud providers, migration to clouds and evaluation of migrated OSS.

The experience report for migrating service-oriented systems to cloud computing [13] highlights the needs for migration planning phase that evaluates some critical issues before legacy migration. These critical issues include the decision about (i) the cloud platform for deployment, (ii) evaluation of source and evolved architecture, (iii) design and deployment decisions for components to be migrated, and (iv) evaluating the business and technical requirements for migration. The proposed Legacy-to-Cloud Migration Horseshoe also incorporates the migration planning process to derive migration strategies based on analyzing requirements, feasibility and decision about cloud providers.

2.3 Existing Frameworks for Software Migration

The *software reengineering horseshoe* [8] proposed by the Software Engineering Institute (SEI) represents the classical framework for architecture-driven reengineering of software systems. Our solution is inspired by the reengineering horseshoe but we specifically focus on migrating legacy applications towards clouds. In addition, we believe the horse model needs to be augmented with an additional process of *architecture migration planning* that drives the migration process.

It is worth mentioning about the OMG's *Architecture Driven Modernization (ADM) horseshoe model* [4]. The ADM horseshoe model consists of three architectural views: *business architecture*, *application and data architecture,* and *technical architecture*. The transformation from legacy to target architecture represents the path of evolution. The ADM method involves transformation of the existing legacy architectures in an incremental fashion to the target architectures.

Finally we discuss the research on migration of legacy application to SOAs [5]. Our solution is conceptually similar to the horseshoe model for migrating legacy applications to SOAs [5], however in comparison to SOA migration we focus on migration to clouds. For a technical distinction between SOAs and cloud-based architectures and their migration efforts (based on service provisioning, design principles and cross-cutting concerns, etc.) please refer to [11].

3. LEGACY-TO-CLOUD MIGRATION HORSHOE FRAMEWORK

In this section, we present the proposed framework called Legacy-to-Cloud Migration Horseshoe in Figure 2 that provides a set of processes and activities for a stepwise and incremental migration of legacy system to cloud-enabled software. An incremental migration refers to the decomposition of a coarse-grained migration process into a set of fine-grained sub-processes and activities that can be addressed in a stepwise manner to achieve software migration.

In this section, we provide details about the main building blocks of the cloud migration horseshoe framework – so called framework *processes, sub-processes and activities* in Figure 2. The processes in the framework define *what* needs to be done and the activities in a process demonstrate *how* it is done [4, 8]. A bottom-up view of the migration horseshoe framework is presented in Figure 2, with a summary of underlying processes, sub-process, and activities in Table 1.

3.1 Process I – Architecture Migration Planning

The first process in legacy to cloud migration involves the development of a migration plan to guide the rest of the processes in Figure 2. With the migration planning process we aim to address the question:

How to develop a migration plan based on the migration needs and feasibility analysis to guide the architectural migration of legacy systems? To answer this question, a description of process activities described in detail subsequently with a process overview in Figure 3.

Process Income - the proposed framework is unique in comparison to the reengineering horseshoe [8] and the ADM framework [4] that it supports the development of a migration plan before architectural recovery and reengineering. There is no specific input to the migration planning process; instead the plan is developed based on the business and technical requirements to cloudify the legacy systems [10, 11].

Activity I – Feasibility Study: as the first activity towards developing the architecture migration plan, a feasibility study is required to evaluate the benefits and the consequences of migrating a legacy application. In the proposed solution this feasibility study is to be conducted by the stakeholders' of the legacy system that is candidate for migration. Two critical concerns during the feasibility analysis are:

1. Required efforts and cost of migration in regard to the perceived benefits. In our systematic review of cloud migration research [11], we discovered that the primary drivers for migration are economic benefits, business agility and smooth execution of the business process [1, 2].

2. Type of System to be Migrated. It is vital to identify that the system to be migrated can fully benefit from cloud-enabled environment to justify the cost and efforts of migration. For example, some of the applications cannot easily utilise cloud-based environments such as safety-critical software [3]. Alternatively, some other type of software such as embedded systems cannot necessarily benefit from cloud-enabled environment. In our solution we focus on migrating the business-critical software [10, 15] so it benefits from the cloud-based environment.

Activity II – Requirement Analysis: this activity deals with the eliciting the architecturally significant requirements [16] for deployment in the cloud. The requirement analysis is a manual activity in which the system stakeholders can gather requirements with a special focus on cloud elasticity requirements. In contrast to the more traditional (object-oriented and component-based) architectures [6], cloud architectures are characterized by a set of (service) elasticity requirements.

Elasticity in general refers to [1, 2, 7]: (1) *Scalability* and (2) *Interoperability* (system's ability to compose/decompose itself using interoperable services). Cloud-based architectures utilise the software as a service (SaaS) model to develop systems that are elastic to frequent variations in their operational environments.

Activity III – Decision on Cloud Providers: The migrated system is based on the Software as a Service (SaaS) model that needs to operate in a Platform as a Service environment (PaaS). Once the requirements are elicited and prioritized, an important activity is to consider the cloud (PaaS) provider [11]. In general the cloud providers can be classified into five distinct categories as (a) *public cloud*, (b) *private cloud*, (c) *hybrid cloud*, (d) *virtual private cloud*, (e) *community cloud*. In our case we focus on public clouds, however cloud provider selection [22] is independent of the outcome of the migration planning – a systematic selection of the most appropriate cloud deployment model is out of the scope of our work.

Activity IV – Architecture Migration Strategy: the last activity in the migration planning process is to develop and evaluate the alternative migration strategies [11, 15]. A migration strategy involves the necessary steps with associated cost and efforts of migration that helps the stakeholders' to adapt a specific strategy based on cost-benefit analysis. The strategy selection is based on evaluating and maintaining the trade-offs between required cost and efforts of migration [12].

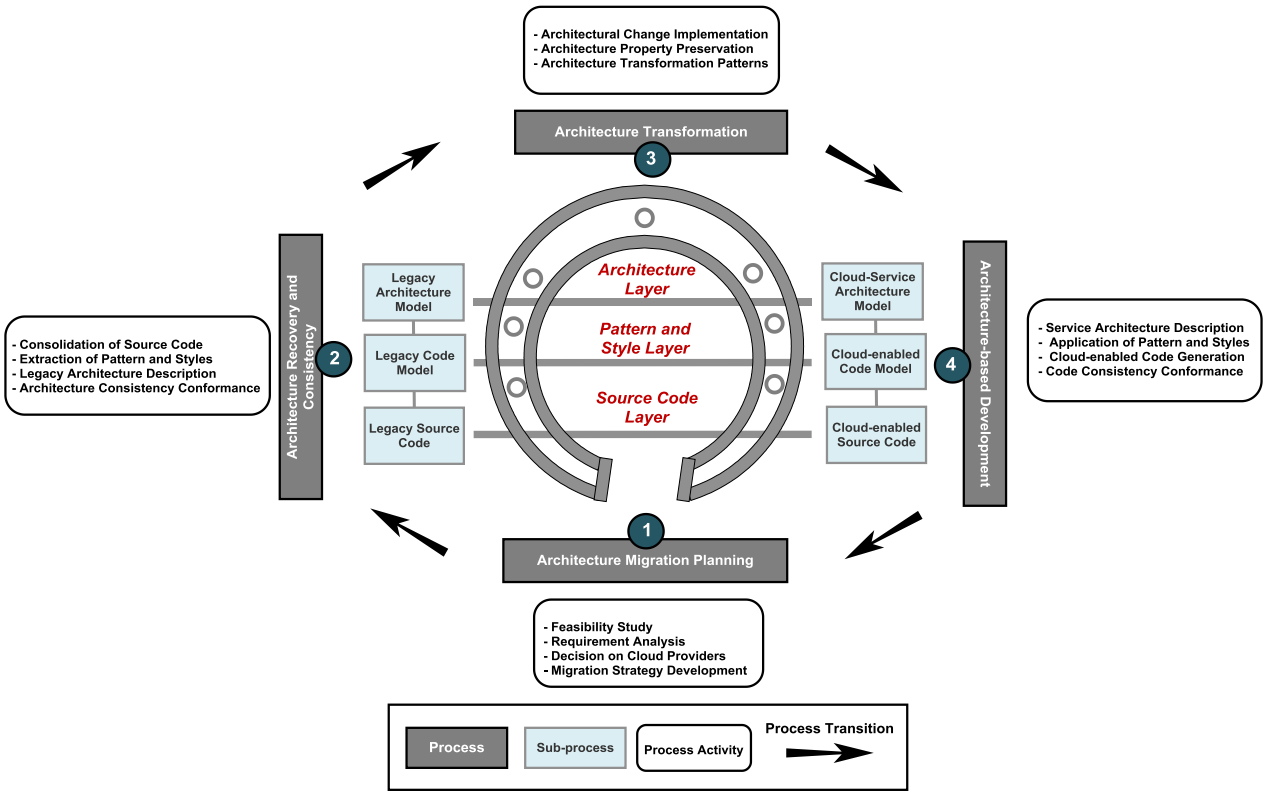

Figure 2. An Overview of the Legacy-to-Cloud Migration Horseshoe Framework

Process Outcome – the outcome of the architecture migration planning process is a *migration plan* that acts as a document for stakeholders' communication about the execution of the migration process.

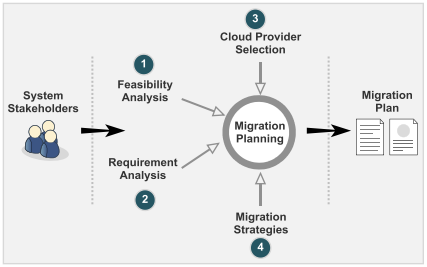

Figure 3. Overview of the Architecture Migration Planning Process

Automation Support – as illustrated in Figure 3, the migration planning process involves a number of distinct activities with an active involvement from cloud migration stakeholders. We propose the migration planning as a human-centric and manual process. Automation of activities can only be seen as a future research in the existing scope.

3.2 Process II – Architecture Recovery and Consistency Conformance

After migration planning, the next step is to recover the legacy architecture as well as confirm its consistency from legacy source code. The code to architecture consistency conformance ensures that recovered architecture model is consistence with the implemented model. With the architecture recovery and consistency conformance process we aim to address the following question:

How to recover the architecture model from legacy source code and ensure that the architectural descriptions are consistent with the source code? To answer this question, a description of process activities are detailed as below with a high-level process overview in Figure 4.

Process Income – the input to this process is architecture migration plan (*Process I* in Figure 2) as illustrated in Figure 4.

Processes	*Sub-processes*	*Activities*	*Process Income*	*Process Outcome*	*Automation Support*
Architecture Migration Planning	(not required)	- Feasibility Study - Requirement Analysis - Decision on Cloud Providers - Migration Strategy Development	(Needs for Migration)	**Migration Plan**	Manual
Architecture Recovery and Consistency Conformance	- *Legacy Code Identification* - *Legacy Code Modeling* - *Legacy Architecture Modeling*	- Consolidation of Source Code - Extraction of Patterns and Styles - Legacy Architecture Description - Architecture Consistency Conformance	Migration Plan	**Legacy Architecture** (from code)	Semi-automated
Architecture Transformation	(not required)	- Architecture Change Implementaion - Architecture Property Preservation - Architecture Transformation Patterns	Legacy Architecture	**Cloud-service Architecture**	Semi-automated
Architecture-based Development	- *Cloud Architecture Modeling* - *Cloud-enabled Code Modeling* - *Cloud-enabled Code Generation*	- Service Architecture Description - Application of Patterns and Styles - Cloud-enabled Code Generation - Code Consistency Conformance	Cloud-service Architecture	**Cloud-enabled Source Code**	Semi-automated

Table 1. A Summary of the Processes, Sub-processes and Activities for Legacy Migration to Cloud-enabled Software.

***Sub-process I - Identification and Consolidation of Legacy Code**:* the first step is the identification and consolidation of the legacy source code that needs migration. In an ideal situation, we expect to recover the architecture model from the object-oriented code. However, in a practical context it is common to have legacy code represented in different programming languages and developed on different platforms.

Activity I – Consolidation of the Legacy Source Code: in order to unify the source code representation, we provide a consolidated source code model. The code consolidation refers to a unified representation of the code gathered from different sources (programming languages). The existing research in [17] guides our model for code consolidation.

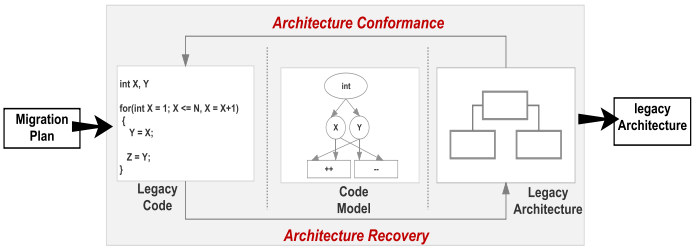

Figure 4. Overview of the Architecture Recovery and Consistency Conformance Process

***Sub-process II - Legacy Code Modeling**:* Once the source code is consolidated we need to present a high-level representation of the source code. A high level representation is essential to identify the critical design rationale in terms of the design patterns and styles that needs to be preserved during transformation. Preserving the desired patterns and styles enable the transition of desired quality attributes from legacy to cloud systems – preserving a layered architecture style.

Activity II – Pattern and Style Extraction from Code Model: We extract the patterns and styles from the source code model as a critical design decisions. We envisage a graph-based model for source code representation, i.e; if source code model is represented as a graph, then application of graph mining techniques can help us to mine frequent patterns in the legacy source code [18].

***Sub-process III - Legacy Architecture Modeling**:* as the last step in architectural recovery and consistency we aim to provide the architectural description and confirm its consistency to source code.

Activity III – Legacy Architecture Description: we represent the architecture model as the topological *configurations* (**CFG**) based on a set of architectural *components* (**CMP**) as the computational entities, linked through *connectors* (**CON**) [6]. Furthermore, architectural components are composed of component *interfaces* (**INF**), while connectors are composed of *endpoints* (**EPT**) to bind component interfaces.

An ideal notation for architectural modeling is to utilize an Architecture Description Language (ADL) [6]. However, ADLs do not provide an explicit support for the evolution of architectural descriptions. This means ADLs can help us to represent the recovered architecture but ADLs fall short of explicitly supporting changes to the architecture model. Therefore, we prefer a graph-based modeling of the recovered legacy architecture (graph node as components and graph edges as component connectors). By modelling architecture as a graph, we imply that: if architecture model can be represented as a graph, then we can exploit graph transformation techniques to evolve architecture model.

Activity IV – Legacy Architecture Consistency Conformance

Finally, the last activity in the architectural recovery process is to confirm the consistency of the legacy architecture model with the source code. Architectural consistency conformance ensures that the recovered architectural descriptions are consistent with specification of the source code [20] (*Activity I* and *Activity II*). Any inconsistencies must be resolved to ensure that desired architectural model is being migrated.

Process Outcome – the outcome of the architecture recovery and consistency process is a legacy architecture model (and with its consistency confirmed) that needs to be transformed to the target cloud architecture.

Automation Support – the architectural recovery and consistency conformance process needs automation especially the consolidation of the code model and recovering the architecture [17, 18]. An appropriate user intervention is required to guide the architectural recovery process.

3.3 Process III – Architecture Transformation

After recovery of the legacy architecture, the architecture transformation process evolves the legacy architecture towards a service-driven cloud architecture. By supporting the architecture transformation process we aim to answer the following question:

How to enable a reuse-driven transformation of a source (legacy architecture) towards a target (service-driven) architecture by preserving the required architectural properties during the transformation process.

Process Income – the input to the architecture transformation process is the legacy architecture (recovered and confirmed to legacy code) as illustrated in Figure 5.

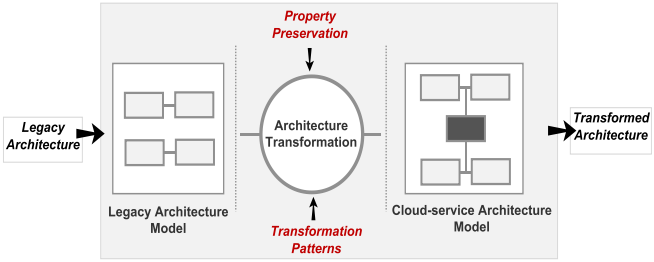

Figure 5. Overview of the Architecture Transformation Process

Activity I – Architecture Change Implementation: during transformation the architectural changes are implemented by means of architecture transformation operators. The transformation operators are fundamental to operationalising and parameterising the architectural changes. We utilise a set of transformation operations that enable the architectural changes by adding (**ADD**), removing (**REM**), and modifying (**MOD**) elements in architecture model (cf. architecture descriptions in architectural recovery process). The architectural composition (component interfaces, connector endpoints) during change execution is preserved with:

– *Atomic Change Operations:* these enable fundamental changes in terms of adding, removing, or modifying the component interfaces (**INF**) and connector endpoints (**EPT**). For example, addition of a new interface I in an existing component C is expressed as follows ($\in$ represents type of element).

$$\text{Add } (I \in \text{INF}, C \in \text{CMP})$$

– *Composite Change Operations:* these are sequential collections of atomic change operations, combined to enable composite architectural changes. These enable adding, removing, or modifying architectural configurations (**CFG**) with service components (**CMP**) containing interfaces, connectors (**CON**) containing endpoints (for component interface binding). For example, addition of a new component C with an interface I in a configuration **G** is specified as follows ($\prec$ represents operational sequence).

$$\text{Add } (C \in \text{CMP}, G \in \text{CFG}) \prec \text{Add } (I \in \text{INF}, C \in \text{CMP})$$

We propose that the transformation operators are *primitive changes* that are composed into *pattern-based changes* to enhance reusability of

architectural transformation. Patterns abstract the addition, removal, and modification of components and connectors to facilitate frequent composition, decomposition, and replacement of architecture elements.

Activity II – Architecture Property Preservation: during transformation it is required to preserve certain properties and design decision [21]. More specifically, in the legacy architecture we are interested in preserving the patterns and styles that represent a design rationale in legacy architecture. For example, in a typical object-oriented system the layered program design (*interface*, *business logic* and *data access* layers) can be preserved as a 3-layered evolved architecture. To preserve the architectural properties, we apply a set of constraints that refer to a set of conditions in terms of pre-conditions (**PRE** - the conditions before application changes) and post-conditions (**POST** – the conditions after application of a changes) to ensure the consistency of architectural transformations. In addition, the invariants (**INV** – the conditions satisfied during application of a change) ensure structural integrity of individual architecture elements during change execution [21]. For example, during addition of a component C, the preconditions ensure that a component C does not exist in a configuration G, and the post-conditions ensure that a component C containing interface I has been successfully added in a configuration G.

Activity III – Applying Architecture Transformation Patterns: a '*transformation pattern*' is defined as a generic, first class abstraction to support potentially reusable architectural changes. It provides a first-class abstraction that can be operationalized and parameterized to support potentially reusable architectural change execution expressed as: PAT<name, intent>:

$$\text{PRE[OPR(arch} \in \text{ARCH)]} \xrightarrow{\text{INV [OPR(arch} \in \text{ARCH)]}} \text{POST[arch}' \in \text{ARCH]}$$

A transformation pattern has a *name* and an *intent* that represents a recurring, constrained (CNS) composition of change operationalization (OPR) on architecture elements ($ae_m \in$ ARCH). Change patterns for architecture evolution are reported in our previous work [29] that can guide architectural transformation.

Process Outcome – the outcome is the transformed architecture in terms of service components representing the PaaS model. The service architecture provides a dynamic composition and interoperability to support the elasticity requirements when deploying service architecture in a cloud-enabled environment.

Automation Support – we aim to support (semi-) automation of the architectural transformation process. The necessary user intervention is required to monitor and customize the transformation process.

3.4 Process IV – Architecture-based Development

Once the legacy architecture is transformed, the last process involves the development of cloud-enabled source code from cloud-service architectures in Figure 6. With the architecture-based development process we aim to answer the following question:

How to exploit architectural abstractions to develop the cloud-enabled source code?

Process Income – the input to the process is the transformed architecture to generate source code.

Sub-process I – Cloud-service Architecture Model

In the architecture-based development of cloud-enabled software, the cloud-service architecture provides a blue print (component and connector view) for developing cloud-enabled source code.

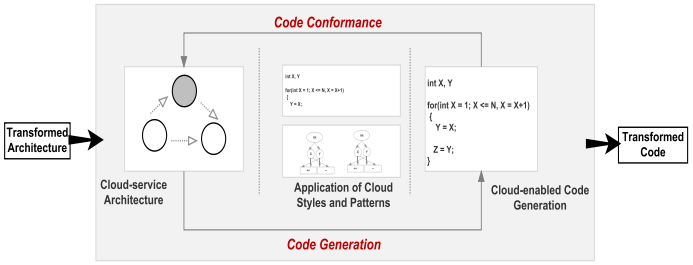

Figure 6. Overview of the Architecture-based Development Process

Activity I – Cloud-service Architecture Description: in order to support a consistent specification of the transformed architecture, the first activity involves a description of the cloud-service architecture. We have already provided the component and connector view of the architecture model (cf. Process II architecture recovery and consistency). However, the legacy architecture is specified using the more conventional object/component-based model. In contrast, the transformed service-architecture model follows the service component architecture (SCA) specifications[5].

Sub-process II – Cloud Pattern and Styles

Once the architecture is specified, the framework support architectural refinement by applying the cloud-specific patterns and styles.

Activity II – Application of Cloud Patterns and Styles: the patterns and styles represent the design rationale to support the quality and extensibility of the architectural design [22]. We augment the transformed cloud architecture with patterns and styles. For example, the identified façade pattern in the legacy architecture could to be preserved and transformed into the service façade pattern [22].

Sub-process III – Cloud-enabled Code Generation

As the last step in architecture-based development, the framework support cloud-enabled code generation and confirming the code consistency with the architecture model.

Activity III – Code Generation and Consistency Conformance: as the last activity of the migration horseshoe framework, we generate the executable source code from cloud-service architecture model [23]. In addition to the code generation, it is also important to confirm the consistency of the generated code with the architectural description. The consistency conformance is vital to ensure that code implementation is consistent with the target architecture.

Process Outcome – the outcome of the architecture-based development process is the cloud-enabled source code. The process represents the migration loop, if a refinement or future migration is required; otherwise the migration process is exited. The round-trip approach enables future reengineering and refinement of the cloud-abled code.

Automation Support – The architecture-based development process can be partially automated in regard to the automatic code generation from the architecture model [23]. The necessary user intervention is required during the pattern and style application.

4. DISCUSSION AND FUTURE RESEARCH

In this paper, we have presented a framework that extends the classical reengineering horseshoe model to support a process-driven approach for legacy software migration to cloud-enabled systems. The proposed framework is the first attempt towards providing a reference model (processes and activities) to support migration to clouds by exploiting software architecture models. We provided a framework named Legacy-to-Cloud Migration Horseshoe by unifying the concepts of

[5] Service Component Architecture (SCA) http://oasis-opencsa.org/sca

software reengineering and *software evolution* for legacy migration to cloud. Considering the growing needs to cloudify the legacy application the framework is beneficial for:

– Researchers in software engineering and software architecture in particular, who need an identification of processes for cloud migration. A reference model provides a foundational body of knowledge to develop innovative theory and solutions, analyse the research efforts, and to establish future dimensions for legacy to cloud migration.

– Practitioners interested in understanding the methods and solutions to exploit the formalism and develop tool support to model, analyse, and implement architecture-driven cloud migration.

In the following we conclude our discussion by presenting our plans for evaluation (Section 4.1), the intended tool support (Section 4.2) and dimensions for future research (Section 4.3).

4.1 Evaluation Plan

The proposed framework organises a set of processes that need to be evaluated with case studies. Case studies provide a scenario-based scenario-driven approach to validate the individual processes and activities of the framework. Currently, we are utilising the case of migrating the on premise mail server to cloud-based email client for IT University of Copenhagen mailing system [28]. The qualitative data and the feedback from the mail system stakeholders (IT department, faculty and students) is being analysed to derive the migration plan. This migration plan (outcome of Process I) is fundamental to proceed towards extraction of the legacy architecture (Process II).

In an overall context of evaluating all the processes of the framework, we aim to evaluate the framework applicability on modernizing an Open Source Software (OSS) framework named Hackystat [15]. We aim to leverage the existing architecture of the systems to utilise the flexibility and scalability of cloud computing through architectural evolution. The data from the case studies and future tool support shall allow us to satisfy the desired objectives, i.e; process and tool support for modernizing legacy systems to cloud-enabled environment.

4.2 Tool Support for Legacy-to-Cloud Migration

We also aim to (semi-) automate the migration process with tool support. The tool support shall allow us to evaluate the results in terms of solution adequacy with real case studies of legacy migration and extending our previous work [13, 15]. We specifically focus automation and customisation of the architectural recovery and architectural transformation processes based. We plan to represent the recovered architecture as a graph model. Graph based modelling of software architecture allows us to exploit the graph transformation techniques for a formal and automated architectural transformation. The graph modelling language (GML)[6] provides an appropriate notation for tool-based interpretation and transformation of graph (architecture) models.

4.3 Dimensions of Future Research

In future, we primary focus on pattern-driven reuse of architectural migration. Considering migration as a recurring problem, in future we aim to exploit the *migration process patterns* as reusable solutions to frequent problems of architectural migration. A *migration process pattern is defined as a generic and repeatable solution that address the frequently occurring migration problems*. We aim to investigate the architecture migration logs – history of architectural changes – to empirically discover the migration process patterns. We propose to exploit pattern as a repeatable solution that can be (i) *empirically discovered* from migration logs, (ii) *systematically documented* in a pattern template, and (iii) *reused* for future migration tasks. A catalogue of migration process patterns is envisaged as a pattern collection that shall guide the migration process.

[6] The GraphML File Format http://graphml.graphdrawing.org/

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A view of cloud computing," *Communications of the ACM,* vol. 53, no. 4, pp. 50-58, 2010.

[2] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the Fifth Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, 2009.

[3] L. Baresi, E. Di Nitto, C. Ghezzi. Toward Open-World Software: Issue and Challenges. *IEEE Computer Society*. 39(10), 36–43, 2006.

[4] W. M. Ulrich and P. Newcomb. Information Systems Transformation: Architecture Driven Modernization Case Studies. *Morgan Kaufmann*, 2010.

[5] A. Winter and J. Ziemann. Model-based Migration to Service-oriented Architectures. In *International Workshop on SOA Maintenance and Evolution*, 2007.

[6] N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pages 70–93, 2000.

[7] N. R. Herbst, S. Kounev and R. Reussner. Elasticity in Cloud Computing: What It Is, and What It Is Not. In *International Conference on Autonomic Computing*, 2013.

[8] R. Kazman, S. G Woods and S. J.Carrière. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In *Fifth Working Conference Reverse Engineering*, pages 154–163, 1998.

[9] M. M. Lehman. *Laws of Software Evolution Revisited*. *In Software Process Technology*, pages 108–124. Springer, 1996.

[10] T. Laszewski, Tom, and P. Nauduri. Migrating to the cloud: Oracle client/server Modernization. *Syngress Publishing, First Edition*, 2011.

[11] P. Jamshidi, A. Ahmad, C. Pahl. Cloud Migration Research: A Systematic Review. *In IEEE Transactions on Cloud Computing*, vol 1, no. 1, 2013.

[12] B. J Williams and J. C. Carver. Characterizing Software Architecture Changes: A Systematic Review. *Information and Software Technology*, vol. 52, no. 1, pages 31–51, 2010.

[13] M. A. Chauhan and M. A. Babar. Migrating Service-oriented System to Cloud Computing: An Experience Report. In *IEEE International Conference on Cloud Computing (CLOUD), 2011*.

[14] Capegemining. Business Cloud: The State of Play Shifts Rapidly. [Online]: http://www.cloud-finder.ch/fileadmin/Dateien/PDF/Expertenberichte/Business_Cloud__The_State_of_Play_Shifts_Rapidly.pdf accessed on December 28, 2013.

[15] M. A. Babar and M. A. Chauhan. A Tale of Migration to Cloud Computing for Sharing Experiences and Observations. *In 2nd International Workshop on Software Engineering for Cloud Computing*. ACM, 2011.

[16] L. Chen, M. A. Babar, and B. Nuseibeh. Characterizing Architecturally Significant Requirements. *In IEEE Software, Vol, no pp 38 – 45*, 2013.

[17] M. P. Chase, S. M. Christey, D. R. Harris, and A.S Yeh. Recovering Software Architecture from Multiple Source Code Analyses. *In ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. *1998.*

[18] J. Dong, Y. Zhao, and T. Peng. A Review of Design Pattern Mining Techniques. In *International Journal of Software Engineering and Knowledge Engineering. vol 19, no 6,* pp 823-855. 2009.

[19] L. Baresi, Luciano, R. Heckel, and S. Thöne. Style-based Modeling and Refinement of Service-oriented Architectures. In *Software & Systems Modeling, vol 5, no 2,* pp 187-207, 2006.

[20] L. Passos, T. R., Valente, M. T. Diniz, R and N. Mendonça. Static Architecture-conformance Checking: An Illustrative Overview. In IEEE *Software,* vol 27, no 5, pp 82-89, 2010.

[21] L. Grunske and E. Lueck. Application of Behavior-Preserving Transformations to Improve Non-Functional Properties of an Architecture Specification. *International Journal of Parallel Programming,* vol 5, no 2, 2004.

[22] B. Wilder. Cloud Architecture Patterns. *O'Reilly Media*, 2012.

[23] A. Stavrou, and G. A. Papadopoulos. Automatic Generation of Executable Code from Software Architecture Models. In *Information Systems Development: Challenges in Practice, Theory and Education*. vol 2, pp 1047-1058, Springer, 2009.

[24] ISO/IEC 14764:2006 Software Engineering -- Software Life Cycle Processes – Maintenance. [Online]: http://www.iso.org/iso/catalogue_detail.htm?csnumber=39064, accessed on December 28, 2013.

[25] P. Mohagheghi and T. Sæther, Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project," in World Congress on Services (SERVICES), 2011, pp. 507– 514.

[26] J. Alonso, L. Orue-Echevarria, M. Escalante, J. Gorroñogoitia and D. Presenza. Cloud Modernization assessment framework: Analyzing the impact of a Potential Migration to Cloud. In *IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems* (MESOCA), 2013.

[27] S. Frey and W. Hasselbring. *The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications.* International Journal on Advances in Software, 4 (3 and 4). pp. 342-353, 2011

[28] M. A. Babar Tales of Empirically Understanding and Providing Process Support for Migrating to Clouds. [Online:] http://www.sei.cmu.edu/community/mesoca2013/upload/Babar-MESOCA-2013.pdf, accessed on December 28, 2013.

[29] A. Ahmad, P. Jamshidi and C. Pahl. *Graph-based Pattern Identification from Architecture Change Logs*. Advanced Information Systems Engineering Workshops, pp. 200-213. Springer. 2012