A Generic and Tailorable Cloud Migration Process Model

Abstract

Cloud computing literature provides a variety of perspectives towards the migration process, each with a different focus and mostly adopting heterogeneous technical-centric terminologies. Little, if any, studies have focused on developing an integrated and abstract process models which captures core domain constructs relevant to the cloud migration. By applying the metamodeling theoretical foundation, this article develops a generic process metamodel, as a domain language, for cloud migration. The metamodel is evaluated and refined through a three step approach including three case studies, domain expert review, and prototype system test. This research benefits academics and practitioners alike by underpinning a substrate for constructing, standardising, maintaining, and sharing bespoke cloud migration processes that suit given migration scenarios.

Keywords: Cloud Migration, Legacy Systems, Metamodel, Domain Language, Process Model

Introduction

Cloud computing technology brings many advantages to the IT-based organizations including: (i) providing wide ranges of services such as processing, data storage, and infrastructure which are universally accessible, can be acquired and released on the fly, and be paid for actual usage, (ii) reducing upgrading cost of IT infrastructure by shifting this responsibility from the organisation to the service provider, and (iii) allowing for on-demand resource elasticity based on computing needs (Armbrust, Fox et al. 2010). These benefits motivate organisations to enable their legacy systems to utilise cloud services. Accounted by (Ried S 2011), the global cloud computing market will likely to grow from $40.7 billion in 2011 to $241 billion in 2020.

A sheer volume of research has been proposed by both academia and industrial communities providing solutions for moving legacy systems to cloud environments (Fahmideh, Daneshgar et al. 2016). Some examples of well-known models, mainly originated from the software engineering literature, are Chauhan's Method (Chauhan and Babar 2012), REMICS (Mohagheghi 2011), Tran's Method (Tran, Keung et al. 2011), Cloud-RMM (Jamshidi, Ahmad et al. 2013), Strauch's Method (S. Strauch 2014), Zhang's Methodology (Zhang, Chung et al. 2004), Oracle Method (Laszewski and Nauduri 2011), ARTIST Method (Menychtas, Santzaridou et al. 2013), Amazon Method (Varia 2010), Legacy-to-Cloud Migration Horseshoe (Ahmad and Babar 2014), IVI Cloud Computing Life Cycle (Conway and Curry 2013), and MILAS (Huru 2009). Each narrows in focus and presents a different viewpoint of the same migration process. For instance, Tran's method has a cost-oriented view defining a taxonomy of the migration activities and cost factors related to these activities (Tran, Keung et al. 2011). REMICS proposed by (Mohagheghi 2011) is an agile and model-driven approach to integrate legacies with cloud services. The method suggested by (Ahmad and Babar 2014) is an architecture-centric software evolution for the migration of legacies to the cloud. There is a logical link between the above process models as they all define the same collection of activities for planning, cloud service selection, re-engineering, testing, and deploying legacies to utilise cloud services, but from different viewpoints. Nevertheless, till now, these links have not been explicitly established and integrated.

Additionally, experts in the cloud computing community who may come from different IT backgrounds use different terminologies and phrases to refer to same concepts. It is hard to find any two migration methods that adopt the same definition of migration process and associated activities. For example, the IVI Cloud Computing Life Cycle, Chauhan's Method, and MILAS define an activity related to the selecting cloud platform. IVI Cloud Computing Life Cycle defines it "as this step will select the best supplier based on value, sustainability, and quality"; Chauhan's Method define this activity as "identify a set of potential cloud computing platforms based on a project's nature, data confidentiality and sensitivity requirements, budget constraints and long-term organisational objectives"; and MILAS (Huru 2009) defines it as "selecting appropriate technology for the modernised system and technology that can run alongside and communicate with the legacy system". IVI Cloud Computing Life Cycle and Chauhan's method take into account some criteria for a cloud platform selection. However, IVI Cloud Computing Life Cycle's definition emphasises the non-functional qualities of a cloud platform whereas Chanhan's definition emphasises aspects of project constraints. On the other hand, MILAS's definition takes into account the interoperability of legacy assets with the cloud services. The above definitions are similar in meaning and context, but they have been expressed by different terms. In other words, when they are viewed collectively, the common theme among all of these definitions is the notion of proper cloud platform/service selection. While such variety and having multiple and

disparate sources of cloud migration is useful, it impedes audience to comprehend, digest, and grasp an overarching view of the cloud migration process. Regardless of operational details of cloud migration, the one question remains which is how IS researcher and practitioners grasp an abstract and overarching view of what the cloud migration process entails under such chaotic universe of cloud? The absence of a platform-independent model of cloud migration introduces communication barriers and obstructs information exchange among participating developers and organisations in a cloud migration project (Hamdaqa and Tahvildari 2012; Zimmermann, Miksovic et al. 2012). In this spirit Hamdaqa et. al. mention: "there is a need to detach the cloud application development process from specific cloud platforms" (Hamdaqa, Livogiannis et al. 2011).

Furthermore, while for some cloud migration scenarios one of the existing process models may be an appropriate fit, for many others a method engineer needs to combine concepts from two or more models to meet requirements of a given migration scenario. For example, a process model might be suitable for moving large and distributed workloads from legacy data centres to public IaaS whilst another process model might be best suited to reengineer legacies to serve as a SaaS. This essence is captured well by Mahmood (Z.Mahmood 2013) in his Book, p.64: "One solution can never fit all problems; likewise, there is a need of customised cloud for individual businesses and dynamically changed requirements of clients". In situations like this, the method engineering is suggested as a way to construct customised methods by assembling reusable method fragments obtained from existing migration methods (Ralyté, Deneckère et al. 2003).

While there are merits in adopting technical-centric existing process models, an integrated overarching view of cloud migration process comprises that can facilitate interoperability and knowledge sharing across the cloud community is still non extant in the available literature. Such model currently does not exist, and the current study can be regarded as a small step towards achieving this goal. The fact that each year a considerable number of research papers are published in the cloud computing field, each reporting different solutions, experience reports, and recommendations, itself is an evidence that the field has reached a maturity point where the development of one such generic reference model becomes mandatory. Prior research acknowledges that although variety of models for any given domain is profitable at the beginning of a research filed, a *consensual picture* of what the bunch of these models looks is eventually more efficacious (Harmsen, Brinkkemper et al. 1994; Rossi, Ramesh et al. 2004; Beydoun, Low et al. 2009). According to this account, it is helpful if common concepts in the cloud migration process such as phases, activities, and work-products could be factored out into a generic and unified process model at a convenient abstraction level. Adequately crafted, it can present a complete vision of the cloud migration process which is independent of any cloud platform, fine-tuneable according to characteristics of a migration scenario, and facilitator for consistent communication and efficient knowledge sharing and exchange across cloud computing domain. Such a generic model that, unifies the access and describes the domain can facilitate design, representation, maintenance, and sharing various cloud migration processes. Methods that are instantiated from such a generic model are expected to describe the domain concepts needed to be performed by developers in any specific scenarios of legacy to cloud migration.

In addressing the abovementioned issue, metamodels are suggested for achieving an integrated view of a domain of interest and to describe it (Atkinson and Kuhne 2003; Gonzalez-Perez and Henderson-Sellers 2008). Metamodels capture common concepts and their relationships describing a domain and the way it works. A metamodel provides a language infrastructure to freely describe a domain in a way that stakeholders can better understand the domain along with guidelines to specialize this language for a particular

context (Rossi and Brinkkemper 1996). Development of metamodels has been a common practice in various themes in information systems and software engineering domains. The significance of metamodels, as a way to abstract cloud computing concepts, has been a top priority in the cloud community (Leymann 2011; Loutas, Kamateri et al. 2011; Hamdaqa and Tahvildari 2012). This paper continues this track from the perspective of cloud migration process. Thus, the objective of this paper is to *develop and evaluate a metamodel that captures and harmonises common activities of cloud migration process and can be used to create, standardise, and share situation-specific cloud migration methods.* The metamodel produced is domain-specific (i.e., cloud computing) but is generic, context agnostic, and can be grounded and extended to suite a given cloud migration context. The proposed model contributes to the cloud computing field by identifying and distilling common activities and key features of extant cloud migration literature. Based on our knowledge, such a model does not exist in the literature.

The paper is structured as follows. The next section reviews prior literature on applying metamodels. Section Research method presents the adopted research approach. Section delineates the approach undertaken to develop the metamodel, following with the section Demonstration that shows how the metamodel can be used to describe real-world cloud migration processes. Next, section Evaluation presents the evaluation of the metamodel. The paper goes on discussion on implications, limitations, and conclusion of this study.

Theoretical foundations and related work

A domain specific language provides core concepts, relationships, notations, and semantic to simply understanding and representation of a particular domain. A key feature of such languages is that they allow domain experts construct models of their applications which can be later translated into low level representations. As suggested by (Atkinson and Kuhne 2003), one effective way to create domain languages is the use of metamodels. A metamodel is “a model of a model or a model of a collection of models” (Atkinson and Kuhne 2003). The literature pertinent to develop metamodels to demystify the multi-faceted and yet ambiguous cloud computing technology varies between different streams. We found that the majority of themes are suggested in software engineering literature with a technical-centric focus on implementation of cloud applications. We also found a few work in IS literature of application of metamodels. The following provides a synopsis of notable research works and shows how the current study situates itself in the context of the existing literature.

The first stream of metamodeling studies concentrates on abstracting the technical architecture of cloud computing. Academic research such (Zhang and Zhou 2009; Hamdaqa 2011; Liu, Tong et al. 2011; Zimmermann, Pretz et al. 2013) and white papers published by major players of cloud computing such IBM, HP, Oracle, and Cisco are subsumed under this classification.

The second stream is about distilling and sharing knowledge practice for the green cloud computing (Procaccianti, Lago et al. 2014). Herein, the application of the metamodel is to formalize a picture of how cloud data centres address the problem of reducing their energy footprint and carbon emission. Another work proposes a metamodel of the green practice for all aspects of cloud-based business processes such as environmental impact, pollution, and waste in class of patterns (Nowak, Breitenbücher et al. 2014). Dougherty et al. have proposed a metamodel-based auto-scaling resource management to reduce unnecessary idle cloud infrastructures (Dougherty, White et al. 2012).

The third stream is concerned with quality aspects of cloud services. As an example, the metamodel developed by cloud accountability project (A4Cloud) is to formulate the knowledge about non-functional properties of cloud services, and in particular, those that influence accountability of cloud providers (Nunez, Fernandez-Gago et al. 2013). The purported goal of this metamodel is to act as a language to describe cloud service accountability in terms of transparency, verifiability, observability, liability. It allows derivation of metrics from a high-level model to a tangible and measurable one, enabling consumers to monitor the quality of cloud service providers. Developing a metamodel to represent and share domain concepts of cloud services certification process has been the goals of European FP7 project as a response to how a certificate is produced, what its content is, and how it is managed (Cimato, Damiani et al. 2013). It conceptualizes the concepts involved during certification phases and allows for defining different instances of certification models.

A number of scholars report that the adoption of metamodels eases code refactoring of cloud applications. The common feature of these studies (Ardagna, Di Nitto et al. 2012; Kopp, Binz et al. 2012; Wettinger, Behrendt et al. 2013) is to address the interoperability and portability of applications across different cloud providers for supporting instantiation of application description into multiple cloud environments using metamodel transformation techniques. This stream of studies uses feature models to model application variability and retransform them for a given target cloud platform.

Capturing the common knowledge of designing of cloud architecture has been the topic of discussions in (Fehling and Retter 2011; Fehling, Leymann et al. 2012) where researchers propose a catalogue of patterns for legacy source code refactoring to enable them to use cloud services.

As the last stream in the software engineering literature, researchers have incorporated domain-specific languages (DSLs) for developing cloud applications. Research in this direction have resulted in several DSLs such as cloud risk modelling (Zech, Felderer et al. 2012), cloud service compliance management (Brandic, Dustdar et al. 2010), cryptographic cloud computing (Bain, Mitchell et al. 2011), distributed data-parallel computing (Isard and Yu 2009), cloud-mobile hybrid applications (Ranabahu, Maximilien et al. 2011), describing big data analytic algorithms for data analytics in the cloud (Weimer, Condie et al. 2011), and automatically code generation for cloud applications (Sledziewski, Bordbar et al. 2010), to maximize SaaS application reusability (La and Kim 2009). The central claim of these technical studies is on the seamless transformation of application codes to various cloud-specific platforms by using model transformation techniques.

Finally, the metamodel creation has also received attention from IS scholars. The work presented in (Martens and Teuteberg 2011; Keller and König 2014) proposes reference models to support organizations in managing and reducing risk and compliance efforts for cloud computing as a socio-technical artefact.

The current research posits that metamodeling is a legitimate and well-suited theoretical lens for understanding the cloud computing domain. However, due to the different viewpoints of metamodel creation in the literature, when it comes to design a process metamodel to establish a methodological foundation for moving legacy systems to the cloud, the research is less common. In this paper, we describe our effort to design and evaluate a generic process metamodel to standardize, tailor, and share cloud migration processes.

Research method

Overview

In the current study the proposed metamodel is viewed as a specific *artefact* and is developed according to the design science paradigm (Peffers, Tuunanen et al. 2008; Gregor and Hevner 2013) using an iterative cycle of design and evaluation. More specifically, we adopted the DSR process model suggested by (Peffers, Tuunanen et al. 2008) that includes the following phases:

Problem identification. This phase has been already described in the Introduction section of this article. That is, the cloud migration literature narrows in focus and present heterogeneous viewpoints of the same cloud migration process while there is no established correspondence among them. As such, it is hard to get an overall understanding of what activities are comprised in a typical cloud migration process. Furthermore, there is a dearth of research that suffices the method tailoring to create bespoke methods to suit a given cloud migration scenario.

Objective definition. The quality of a proposed metamodel is an integrated part of a metamodeling process. The model quality is "the totality of features and characteristics of a conceptual model that bear on its ability to satisfy stated or implied needs" (Moody 2005). Thus, the development process of the proposed metamodel of the current study was informed by design principles (DP) pertinent to design of domain languages. There are a few commonly used frameworks for examining the quality of conceptual models, which are applicable to different modelling paradigms (Lindland, Sindre et al. 1994), (Stamper 1996), (Moody 1998), and (Paige, Brooke et al. 2007). From these frameworks we identified the following common design principles (DP) that are expected to be satisfied by a proposed metamodel: *Completeness* (DP1): the metamodel should capture all important and relevant methodological constructs that cloud migration process entails, *Understandability* (DP2): the definitions and names of constructs in the metamodel should be comprehensible by domain experts, *Correctness* (DP3): the notation and relationships among the constructs in the metamodel should be correct and meaningful, and finally *Tailorability/or flexibility* (DP4): the metamodel should enable method engineers to standardise, share, and tailor cloud migration methods according to characteristics of given scenarios. These generic design principles are further specialized during the remaining phases.

Design and development. A consolidated metamodel was derived from the extant literature on cloud migration, which comprised all frequently occurring constructs in any process of legacy system migration to cloud environments and relationships among them. We first identified all relevant studies (process models, approaches, experience reports) on moving legacy systems to the cloud. Next, constructs were extracted from these studies, grouped, and refined based on their similarities and context. This step resulted in producing a set of essential constructs of the metamodel. Following harmonised constructs' definitions, they were organised into phases and relationships among them were specified.

Demonstration. The purpose of this phase was to show the expressivity of the metamodel to represent real-world enacted cloud migration processes. Three case studies were purposefully selected on the basis of (i) having clear goals for the cloud migration, (ii) reflecting various migration types such as IaaS, SaaS, and PaaS, and (iii) having available supportive documentation of performed cloud migration scenario for a detailed analysis. Three selected cases were: *InformaIT* in Sweden, *TOAS* in Finland, and *Spring Trader* in the United State. The unit of analysis was the legacy system that was planned for migration. Adherence to DP1 and DP3 were examined by tracing the origin of the metamodel constructs and their relationships to these real-world migration models.

Evaluation. This phase was to evaluate the efficacy of the metamodel version 1.1, which had been resulted after applying refinements in the demonstration phase. Firstly, the metamodel adherence to DP1, DP2, and DP3 were examined by a panel of experts in the cloud computing field. The choice of domain experts were based on either having at least one year of experience in legacy system migration to cloud environments or extensive academic knowledge of cloud migration as evidenced by publications. Four experts, denoted by E1 to E4, who were geographically dispersed and had an overall 7.5 years of experience in the area of cloud migration, were selected and provided with the textual document of the metamodel (twenty-five pages long) along with a list of open-ended questions related to the metamodel's support of DP1, DP2, and DP3. Each expert individually was asked to review and challenge the metamodel version 1.1. Neither expert was aware of the identity of other experts to avoid possible communication between them. An advantage of receiving feedback from experts with different cloud migration experience was that their expertise complemented each other by addressing different parts of the metamodel.

The deadline for receiving feedback was negotiated with each expert. Feedback received from experts was analysed and relevant refinements were applied to the metamodel. To prevent possible misinterpretation of comments made by experts, an email-based communication was established to clarify comments whenever required.

Secondly, a prototype system was implemented by the authors to show how the metamodel can be specialised for given migration scenarios and be used for standardisation of migration processes across the cloud community. The prototype system uses the metamodel as a repository of method fragments and provides interactive forms for constructing, configuring, standardising, and sharing situation-specific cloud migration methods for a scenario at hand. Qualitative feedback from two experts, denoted by E5 and E6, about the adherence of the metamodel to all design principles were sought in this evaluation step. The feedback collected from each step of evaluation was used to refine the metamodel to its next version.

Communication. The document of the metamodel in sufficient detail and its actual implementation (prototype) are also available in (MLSAC 2016).

As shown in Figure 1, this research was conducted in four consecutive iterations. In this figure, down arrows and back arrows show, respectively, the output of each phase and the metamodel refinement through design phase engine. Each iteration used the refined metamodel resulted from the predecessor iteration as the input. Starting from version 1.0, the metamodel refinements throughout iterations was labelled with an increasing version number. The first iteration resulted in the initial design of the metamodel version 1.0. The second iteration appraised the completeness and correctness of the initial metamodel through three case studies. By analysing results from these cases, the metamodel version 1.0 was refined to version 1.1 by adding new constructs which had not been captured by the metamodel version 1.0. Later, in the third iteration, a panel of domain experts individually evaluated the metamodel version 1.1 and subsequently their feedback was applied to the metamodel, yielding to the next version of the metamodel, i.e. version 1.2. Finally, in the fourth iteration, the evaluation was conducted by examining a prototype system of the metamodel version 1.2. This iteration did not result in further refinement of the metamodel.

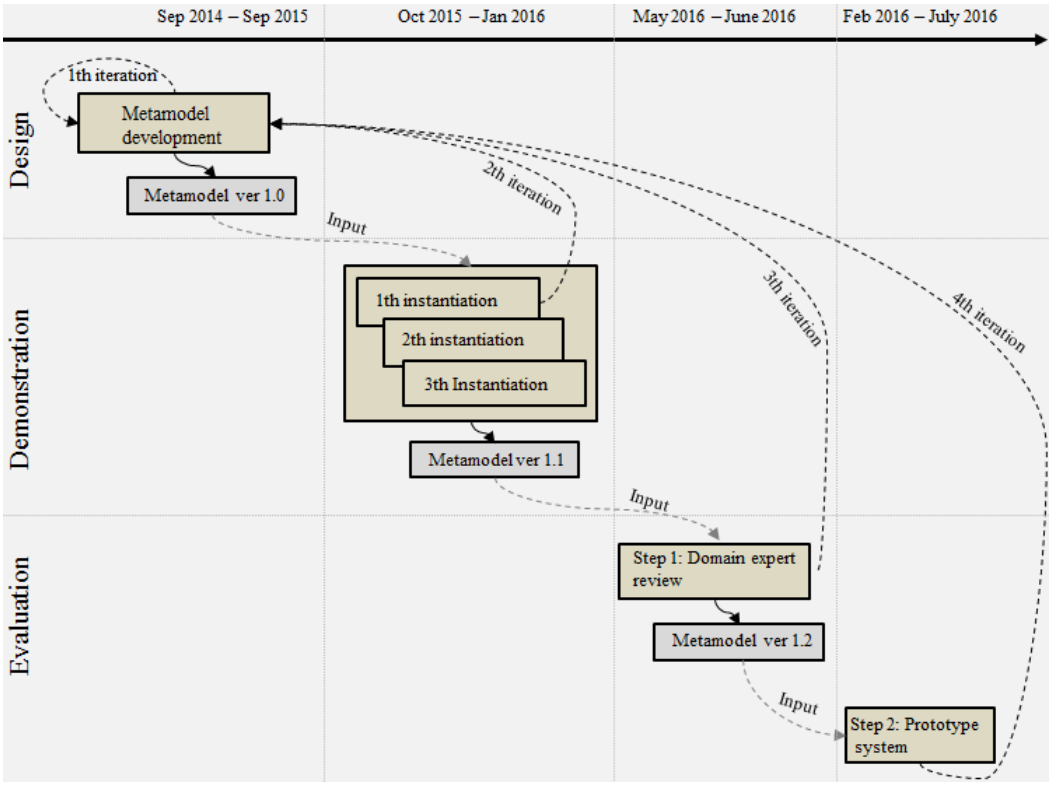

Figure 1 Design science research process specialized for this research

Design Phase

Developing Design Principles for the Metamodel

The quality of a designed metamodel is an integrated part of a metamodeling process, "the act and science of creating meta-models, which are a qualified variant of models" (Gonzalez-Perez and Henderson-Sellers 2008) (p. 32). The (meta)model quality, defined in (Moody 2005), is "the totality of features and characteristics of a conceptual model that bear on its ability to satisfy stated or implied needs" (p. 2). Accordingly, this section synthesises a few design principles as fundamental requirements that are expected to be addressed during the development and evaluation of the proposed metamodel. Results of the demonstration and evaluation shows the proposed metamodel based on these principles provides a methodological foundation for moving legacy systems to cloud platforms including support for creating, configuring, and sharing customised methods for different scenarios.

Completeness (Design principle 1). The development of the design principles are originated from the existing mainstream metamodeling frameworks and recurring concerns during a cloud migration process. Design principles that are proposed in this section are testable propositions and further are employed to develop and evaluate the metamodel. For instance, researchers can evaluate the adherence of a metamodel to design principles using case studies (Antkiewicz, Czarnecki et al. 2009; Cuadrado and Molina 2009; Karlsson and Ågerfalk 2012). Design principles and their connection with the context of this research are discussed in the following.

One concern during creating a metamodel is the level of its completeness, that is, the extent to which the metamodel can make different kinds of statements required in the domain (Lindland, Sindre et al. 1994). Mitchell states that a language designer should discover key constructs in the problem and ensure they are modelled and representable during system development lifecycle (Bain, Mitchell et al. 2011). A designer may tend to

include many domain constructs in a metamodel. However, achieving valid completeness may not be feasible. In addition, the domain may contain many constructs that are irrelevant and, hence, out of the scope of the domain language purpose. Overemphasising on a domain language with many constructs is a worth practice.

Completeness can be considered in terms of an appropriate balance between generality and specificity is an important factor in the successful development of a domain language. A language can be either too generic or too specific to the domain and in some cases both. Steven et al. (Kelly and Pohjonen 2009) suggest to include common core constructs in the domain. They mention, "Domain language isn't about achieving perfection, just something that works in practice. It will always be possible to imagine a case that the language can't handle. The important questions are how often such cases occur in practice, and how well the language deals with common cases" (p. 23). They further advise that in order to avoid analysis paralysis, one should concentrate on the core cases and build a prototype language for them. Defining a threshold for a metamodel completeness depends on the application context and, although is not easy to quantify, it can be when the model is detailed enough according to the purpose of modelling and further modelling is less beneficial (Lindland, Sindre et al. 1994). This can be examined, for example, by tracing proposed metamodel constructs to real word models (Othman, Beydoun et al. 2014) or existing counterpart models (Beydoun, Low et al. 2009).

When viewing the cloud migration from the process perspective, a good coverage on core activities and expected work-products incorporating into the migration process is important. The key concerns initially introduced (S. Strauch 2014) in and then enriched and validated in (Fahmideh, Daneshgar et al. 2016) were found good yardstick to get a feasible completeness of functional and non-functional methodological requirements to be addressed by an ideal cloud migration process model. The concerns are Analysing Organisational Context, Understanding Cloud Migration Objectives and Requirements, Proper Cloud Migration Planning, Understanding Legacy Applications, Target Cloud Platform/Service Selection, Re-Architecting Legacy Applications, Environment Configuration, Testing, and Tailoring. This leads defining the first design principle:

> *The proposed metamodel should capture all important and sound methodological constructs that are relevant for the incorporation into a typical process of the legacy system to the cloud.*

Understandability (Design principle 2). Developing a domain language needs a good knowledge of the domain. This implies that a language designer should think abstract and take his/her noise above the system code level, programming, and technical-oriented notions in the domain (Kelly and Pohjonen 2009). He/she should be able to produce good vocabularies for the domain that are understandable and interpretable by the audience of the language as referred to it as *Audience-domain appropriateness* (Lindland, Sindre et al. 1994). An adequate domain language allows for minimum multiple interpretations by audiences. Ambler (Ambler 2005) states that for better understandability of a model, it should be kept simple and avoids details not necessary for modelling. Excessive emphasis on incorporating technical or programming concepts into a domain language, although, is useful they should not be defined as core constructs otherwise they impede the expressivity power of the metamodel and lead to a poor abstraction level (Kelly and Pohjonen 2009). The understandability of a domain language is determined by many properties such as quality of diagrams or text, icons and names, and the layout and closeness of the model to the domain (Lindland, Sindre et al. 1994). In the context of this research, an understandable can clarify the meaning of activities in order to understand what cloud service is supposed to provide and what service consumer need to consider or implement to

utilize advantages of cloud computing such as availability and scalability. Thus, naming and terminologies that proposed for the metamodel are results of synthesising the content in the cloud computing literature. The second design principle is formulated as the following:

> *The definitions and names of constructs in the metamodel should be comprehensible by domain experts.*

Correctness (Design principle 3). A domain language contains names, definitions, and relationships among constructs, which are relative to the domain, and they are interpretable by human and computer for the purpose of generation and analysis (Moody 1998; Paige, Brooke et al. 2007). The correctness can be checked either by examining the language against existing domain models or domain experts. A metamodel for cloud migration process needs to specify relationships among operations, which can be in the form of a sequence, input/output, association, or aggregation. An example may illustrate this point. According to (Fahmideh, Daneshgar et al. 2016), a key concern in a cloud migration scenario is potential incompatibilities (e.g. APIs) between legacy systems and cloud services. This implies a sequence in the migration process in the sense that once a decision on the cloud platform selection is made, the next step is to identify and analyse incompatibilities between these two platforms. In this research, the relationships defined in the framework are based on the recommendations in the cloud computing literature. The second design principle is defined as the following:

> *The notation and relationships among the constructs in the framework should be correct and meaningful.*

Tailorability (Design principle 3). A domain language should support configuration and extension so that it can be specialized into a new domain and continuously evolved according to upcoming domain requirements. The more a domain language is close to the problem domain, the more simple its customisation, maintenance, and evolution (Jonkers, Stroucken et al. 2006). A domain language includes a set of generic constructs, which are abstract enough and common to represent the domain. Customised models from a domain language can then be used to generate software systems. Converting the above point to the context of this research, the proposed metamodel, which is supposed to a representation for the cloud migration process, should be tailorable to meet requirements of migration scenarios. This is due to the fact that each cloud migration scenario may have different characteristics such as system workload for moving to the cloud, chose of migration type and cloud services. Hence, there is no single applicable method for all scenarios. In situations like this, designing customisable methods or configuring existing one that fit characteristics of migration scenarios is pivotal for successful adoption of the cloud computing (Fahmideh, Daneshgar et al. 2016). With respect to this, the fourth design principle is defined as follow.

> *The framework should enable method designers in standardising, sharing, and tailoring migration methods for specific scenarios.*

Metamodel development

The steps that were undertaken to develop initial metamodel in the design phase are explained below.

Identifying studies. The development of the proposed metamodel was started by identifying all constructs relevant to the cloud migration process. We utilized past researches in the cloud migration literature as the main knowledge source for the creation of the metamodel. Due to a large volume of published researches, a systematic literature review was conducted to identify important and meaningful constructs stated in the literature for inclusion in the

metamodel. Recommendations proposed by (Kitchenham, Pearl Brereton et al. 2009) were used to identify, characterise, and assess studies suggesting solutions for moving legacy systems to the cloud. The keywords for search were *Cloud*, *Cloud Computing*, *Service Computing*, *Legacy*, *Methodology*, *Process Model*, *Reference Model*, and *Migration* were set as the main keywords and based upon them, the different search strings were defined using the logical operator OR to include synonyms for each search string as well as the logical operator AND to link together each set of synonyms. Seventy five relevant studies were identified from the cloud migration literature. The details this are presented in the Supplementary Material Appendix A.

Extracting constructs. All relevant constructs related to the cloud migration process were extracted from all the identified studies. In this study a construct refers to a (i) *Task*: a discrete and small unit of migration work that developers may perform to achieve one or more specified goals, (ii) *Work-product*: a tangible artefact that is produced during the migration process and used by other tasks, (iii) *Principle*: a consideration that should be taken into account during cloud application design, and (iv) *Phase*: a logical concept to manage the complexity of migration process and classify tasks and work-product constructs. A phase represents a particular period of a cloud migration process.

Derivation of the metamodel was based on the DP1 and DP3 as defined earlier. More exactly, for the DP1 we leveraged the identified the key common occurring concerns during legacy system migration to cloud environment as discussed in (Fahmideh, Daneshgar et al. 2016). This includes eight concerns such *as understanding organisational context*, *understanding cloud migration objective and requirements*, *proper cloud migration planning*, *understanding legacy systems*, *target cloud platform selection*, *re-architecting legacy systems*, *environment configuration*, and *testing*. There was a tendency in selection of constructs that were (i) sufficiently generic to a variety of cloud migration scenarios and (ii) also were platform and application independent. Constructs that were too general or belonged to general software engineering were not extracted as they were deemed out of the scope of this research. These included constructs related to the process governance and umbrella activities such as risk management, project management, quality assurance, configuration management, and measurement. For each construct its definition from the studies was also extracted. The full list of all constructs and their definitions are presented in the Supplementary Material Appendices B and C, respectively.

Creating overarching constructs. Through a bottom-up approach, all identified constructs from the previous step were grouped based on their similarities and definitions to derive a set of high-level overarching constructs. Classification of constructs and creating overarching constructs were undertaken on the basis of the key concerns such as understanding organisational context, re-architecting legacy system, and understanding legacy system during moving legacy systems to the cloud as discussed in (Fahmideh, Daneshgar et al. 2016).

Harmonizing and reconciliation of constructs. Various definitions of overarching constructs were reconciled to reach a set of internally consistent set of metamodel constructs. When there were several definitions for a constructs, a hybrid definition which encompassed all definitions was chosen. Back to the example mentioned earlier, selecting cloud platform has been defined as “this step will select the best supplier based on value, sustainability, and quality” in IVI Cloud Computing Life Cycle (Conway and Curry 2013); as “identify a set of potential cloud computing platforms based on a project’s nature, data confidentiality and sensitivity requirements, budget constraints and long-term organisational objectives” in Chauhan’s method (Chauhan and Babar 2012); and as “selecting appropriate technology for the modernised system and technology that can run alongside and communicate with the

legacy system" in MILAS (Huru 2009). The definition decided for the metamodel is "Define a set of suitability criteria that characterise desirable features of cloud platforms. The criteria include provider profile (pricing model, constraints, offered QoS, electricity costs, power, and cooling costs), organisation migration characteristics (migration goals, available budget), and application requirements. Based on the criteria identify and select suitable cloud providers" which is a hybrid definition that encompasses all interpretations from these models.

Classification and organising constructs into phases. Constructs were organised in terms of migration phases. The identified studies in the first step were analysed to identify generic phases of the cloud migration process. For instance, IVI Cloud Computing Life Cycle (Conway and Curry 2013) include four phases namely *Architect*, *Engage*, *Operate*, and *Refresh*. Strauch's Method (S. Strauch 2014) includes three phases as *Assessment*, *Analysis and Design*, *Migration*, *Deployment*, and *Support*. Similarly, Cloud-RMM (Jamshidi, Ahmad et al. 2013) has three phases including *Migration Planning*, *Migration Execution*, and *Migration Evaluation*. Synthesised the similarity of phases in these studies, we defined three phases including *Plan*, *Design*, and *Enable* that are described in the next section.

Defining relationships among constructs. The relationships among constructs such as sequence, association, specialization, and aggregation were defined based on the identified studies in the step one and the output from the previous step. All relationships are presented in the Supplementary Material Appendix E. To represent the metamodel in a clear and well-structured manner, a simple version of UML notation (UML 2004) was used, which is a semi-formal and de-facto standard for information modelling.

Resultant Metamodel

The objective of the metamodel is to provide a generic representation of the cloud migration process that facilitates domain understanding, standardising, creating, and sharing customised migration methods. The metamodel includes a set constructs which are common comprised in the cloud migration. The constructs are organised into three phases namely *Plan*, *Design*, and *Enable*. Operationalisation details are left to each individual instantiation of the metamodel using available techniques in the cloud computing literature and/or tools in marketplace. Figure 1 shows the developed metamodel. A brief definition of the metamodel constructs is presented in Table 1.

The *Plan* phase starts with a feasibility analysis of cloud adoption. This analysis can be related to potential changes in organisation structure, local network, and cost saving as the outcome of cloud migration. Moreover, an understanding of the current state of legacies is required to know their architecture, functional and none-functional requirements that might either be addressed by cloud services or be threatened by moving them to the cloud. This also gets estimation of required reengineering effort for making legacy systems cloud-enabled. A model of legacies including their components and their deployment relations is produced as the output of this activity. Legacy systems may have certain requirements that can be satisfied by utilising cloud services. These requirements may be related to computational, storage space, security, and system interoperability. A cloud migration also includes preparing a plan which organises the sequence of activities in the course of migration process.

In the *Design* phase a new architecture model is designed which enables legacies to utilise cloud services. The re-architecting process includes identifying suitable legacies or components for moving to and their new deployment in the cloud environment in order to satisfy non-functional requirements. Examples include data security, expected workload, and acceptable network and scaling latency, selecting cloud services which fit requirements of

these legacies as identified in previous phase, and identifying inconsistencies between underlying legacy technologies and selected cloud services' APIs. In some situations, organisation's regulations and policies do not allow for a full legacy system migration and hence some legacy components are moved to the cloud and others are kept in the local network utilising cloud services that are offered to them. In re-architecting of legacies to cloud environments, design principles play central role. For instance, in order to support the dynamic deployability, handling with failures, and independent scalability in cloud environment, system components should be designed stateless in order to minimise storing the contextual data during their execution. An important consideration during cloud architecture design is the performance variability of cloud servers and network latency between local network and the cloud which can have a negative impact on the QoS of a migrated system. Developers should implement mechanisms in legacies to detect and handle transient faults that occur in cloud environments. A key work-product of this phase is architecture model which specifies an optimum distribution of legacy components on the cloud servers and takes into account data privacy, acceptable network latency and performance variability of cloud services, the availability zone of cloud servers, the affinity of system components in the cloud, and the geographical location of servers.

The *Enable* phase consists of a set of reengineering activities to enable legacies for utilising cloud services. This will result in the realisation of the cloud architecture designed in the previous phase. Often, legacy systems have been implemented with technologies which are not compatible with cloud services (e.g. API incompatibilities or proprietary). If occurs, such incompatibilities between the legacy and cloud services should be identified and accordingly resolved through adaptation mechanisms. This can be in form refactoring legacy source codes, modifying data, or implementing wrappers code refactoring. For example, resolving inconsistencies between legacy database and a selected cloud database solution may imply a need for the data type conversion, query transformation, database schema transformation, and developing runtime emulators. Legacy systems might not have been implemented with a support for dynamic resource acquisition and release under input workload. High workloads are often addressed by adding new physical servers. Mechanisms for system elasticity in cloud environments need to be implemented in legacies for continuous system monitoring and performing actions for resource acquisition and release regarding scaling rules triggered in a specific workload threshold, event, or metric. Reengineering legacies may entail either adding new components to legacies or separately hosted in cloud servers. Local network setting is reconfigured to provide access to cloud services. In addition, legacy components and any required third party tools are installed in the cloud. Finally, in the metamodel, the test activity includes testing both functional and non-functional aspects of the migrated system. In particular, various cloud-specific tests should be performed including security test, interoperability test, and workload test.

It is important to note that adopting different service delivery models may entail incorporating different constructs of the metamodel during the migration process. The metamodel includes guidelines for the connection between a chosen service delivery model and the metamodel constructs. Situations in which a construct should be incorporated into the migration process can be mandatory, situational, and unnecessary.

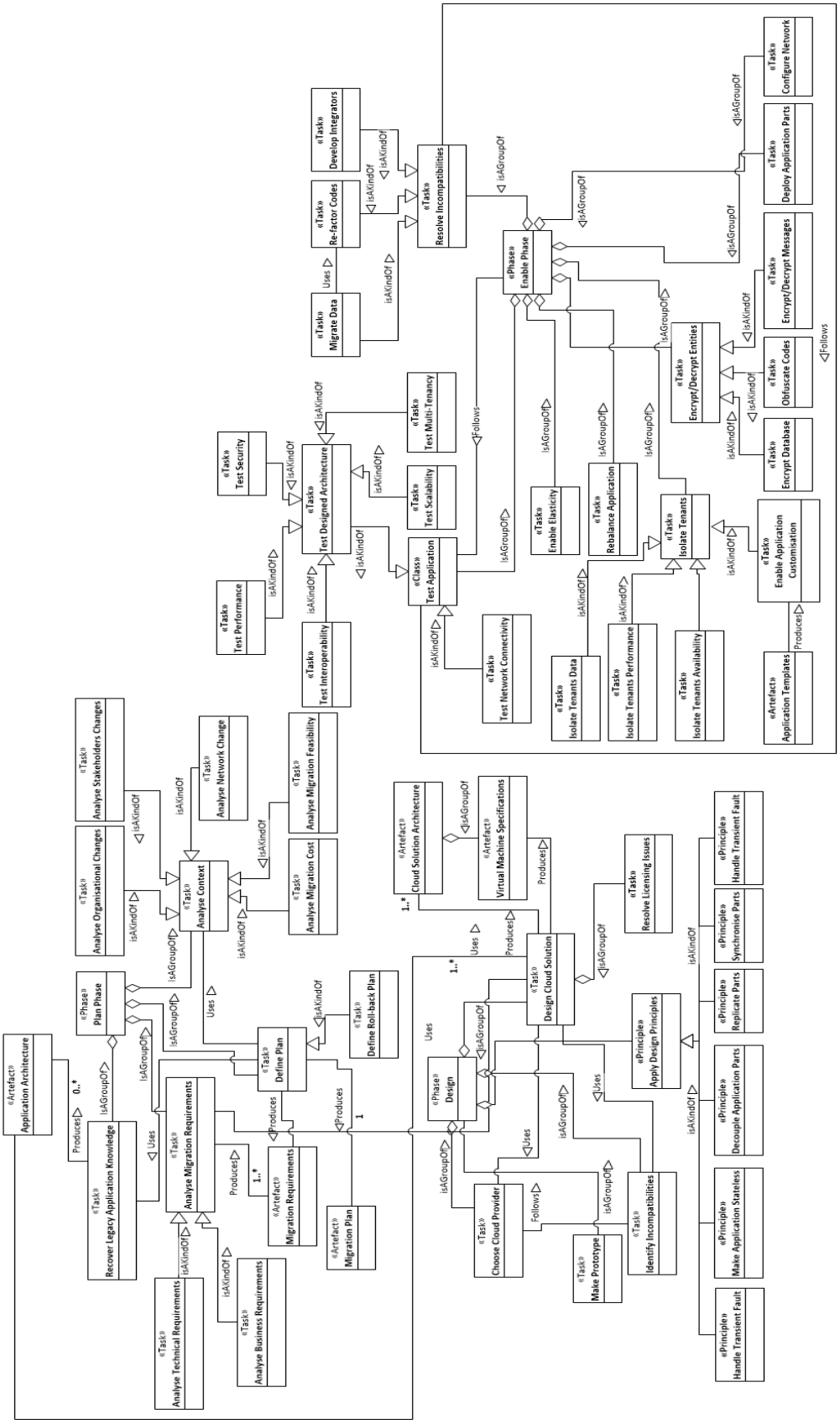

Figure 2 Metamodel for cloud migration process

Table 1. Definitions of constructs in the metamodel

Phase	Activity	Definition
Plan Phase	Analyse Business Requirements	Provide an understanding of what an organisation wants to achieve and goals and expectations are to be met by cloud migration.
	Analyse Migration Cost	Analyse migration to cloud with respect to the cost of application modification, installation, training, administration, license management, developing cloud skills, pricing models of the service providers, and infrastructure procurement imposed by the migration.
	Analyse Migration Feasibility	Identify potential organisational constraints regarding the adoption of a particular cloud model, based on information available in the organisation profile and then perform a feasibility study to evaluate the benefits and the consequences of migrating legacies to the cloud.
	Analyse Network Change	Perform an impact analysis of potential changes in organisation network due to migration in order to identify any side effect on communication between local and cloud components and the required bandwidth.
	Analyse Organisational Changes	Analyse the impact of cloud migration on the structure and resources of organisation.
	Analyse Technical Requirements	Acquire a set of legacy requirements such as computational requirements, servers, data storage and security, networking and response time, and elasticity requirements from multiple stakeholders about the target application according to the current configuration setting of the legacy application. This helps to gain a good understanding of different kinds of architectural changes that needed to be made in the legacy application.
	Define Plan	Define a correct and safe sequence of tasks that guide the migration process by analysis feedback from stakeholders. A plan may a procedure for (i) notifying legacy application users about the cloud migration and temporal unavailability of legacies during the migration period and activating them after the migration, (ii) rollback to an in-house version of the legacy in the case of occurrence of any significant risk during the migration process, (iii) retiring legacy components and infrastructure that are no longer needed, after a pre-defined period of monitoring and successful migration from original environment to the cloud.
	Recover Legacy Application Knowledge	Produce a complete representation of legacy architecture application including its data, components, dependencies among components and infrastructure, application data usage and resource utilisation model (e.g. CPU, Network, storage).
Design Phase	Choose Cloud Platform/Provider	Define a set of suitability criteria that characterise desirable features of cloud providers. The criteria include provider profile (pricing model, constraints, offered QoS, electricity costs, power, and cooling costs), organisation migration characteristics (migration goals, available budget), and application requirements. Based on the criteria, identify and select suitable cloud providers.
	Design Cloud Solution	Identify legacy components which are appropriate for the migration to the cloud regarding identified requirements (e.g. workload, data privacy, confidentiality, latency, dependencies between application components, and migration goals) and then define their deployment and distribution in the cloud environment on the basis of organisation profile, cost saving, expected workload, performance, transaction delay, availability, security, and constraints.
	Identify Incompatibilities	Identify and assess the list of potential incompatibilities between existing application components and selected cloud service. Different sources of incompatibilities should be checked including library, database, interface, behavioural, code style, communication protocol, offered QoS, and policy mismatches.
	Make Application Stateless	Provide support in the application to the handle safety and traceability of tenant's session when various application instances are hosted in the cloud.
Enable Phase	Adapt Data	Refine the current database schema for making it compliant with the schema of cloud database solution. Enhance the data access layer by adaptors and convertors so as to fulfil functionalities and necessary optimised query

		transformations which are not supported by a chosen cloud database solution. Also, convert database data type to the target cloud database solution.
	Develop Integrators	Develop a mediator component that resolves mismatches between legacy and cloud services that are plugged to the legacy. This component wraps/transforms incompatibility (e.g. message format, interaction protocols) between the legacy and cloud services.
	Enable Elasticity	Define scaling rules and provide support for dynamic acquisition and release of cloud resources.
	Handle Transient Faults	Detect and handle transient faults may occur in the cloud.
	Isolate Tenant Availability	Detect and handle faults that may incur in a tenant in a way that it cannot be propagated to other tenants.
	Isolate Tenant Customisability	Analyse commonality and variability in the target domain and provide support for the customisation of application components on the basis of particular needs and situations of tenants before and during running application in the cloud.
	Isolate Tenant Data	Protect tenants' data from to be accessed by other tenants. Each tenant should be authorised and able to access to its own data.
	Encrypt/Decrypt Messages	Secure messages transmission between the local components and those hosted in the cloud or distributed across multiple clouds using an encryption mechanism.
	Refactor Codes	Refine (or re-implement) the source code for being compatible and able to interact with the selected cloud platform programming language. Also, refine (or re-implement) component interface operations, signature, messages, and data type for being able to interact with the selected cloud platform.
	Re-configure Network	Re-configure the running environment of the application including reachability policies to resources and network, connection to storages, setting ports and firewalls, and load balancer.
	Synchronise Application Components	Provide a support in the application to synchronise multiple components (e.g. database replica) hosted legacy network and clouds.
	Test Interoperability	Test the compatibility of components with different cloud environments specifically, when components can be switched between different infrastructures.
	Test Multi-tenancy	Test if tenants can easily configure the application components, i.e. user interfaces, business logic and workflows, and functional services.
	Test Network Connectivity	Test the network connectivity between local components and components in the cloud.
	Test Performance	Test the performance (e.g. process speed, response time, throughput, latency, and etc.) of the application when subjected to increased load from multiple tenants.
	Test Scalability	Test to assure the application acquire and release the computing resources in an efficient manner.
	Test Security	Test application components and reachability policies to access these components against the security requirements.
Work-products	Application Templates	A set of models allowing tenants/application users to customise variation points and features in the application components. These allow tenants/users to configure application components.
	Cloud Solution Architecture	A complete high-level architecture document that will serve in later stages as a guidebook for the implementation
	Legacy Application Model	A model of legacy application including its components and their deployment relations.
	Migration Plan	A document that defines the execution of the migration process and sequence with which legacy application is to be moved to the cloud.
	Migration Requirements	A set of requirements as a result of task Analyse Migration Requirements.
	Virtual Model Specification	Virtual images of the application that are associated with the application components.

Demonstration

This phase shows the metamodel adherence to the DP1 and DP2. More exactly, as the proposed metamodel is sufficiently generic and abstracts domain constructs being incorporated during the cloud migration process, it is anticipated that real-world migration processes including development activities, their relationships, and work-products can be represented by the metamodel. Three case studies were analysed to examine the conformance of the real-world scenarios to the metamodel and corresponding between constructs in the metamodel and these cases. These are shown in Table 2. Due to space constraint, a detail analysis is presented for the first case (*InformIT*) only and results from two other cases are presented in the following (A full detailed of the case analysis is in the Supplementary material Appendix F).

Adherence to DP1 and DP3 were appraised by tracing the origin of metamodel constructs and their relationships to real-world migration models. Using the tracing technique (Sargent 2005) in this section is similar to the research by (Othman and Beydoun 2013), which used an analysis of existing disaster scenarios to show the capability of their suggested metamodel in expressing key domain constructs. It is also consistent with (Beydoun, Low et al. 2009) in examining an agent-oriented metamodel in the coverage of design-time and run-time constructs in the agent-oriented software development. Furthermore, the tracing technique has been also used in (Antkiewicz, Czarnecki et al. 2009) in representing the domain knowledge of application code understanding through reverse/forward engineering and software code evolution. Using the tracing technique, constructs that were incorporated in the migration scenarios were categorised and mapped into the proposed metamodel constructs according to their relevance. Some of the leading questions that were used during case review for identification of the seed cloud-specific activities enacted by the developers in each phase of migration process were as follow: (i) what activities, including techniques, were performed and deliverables were produced during each phase of your migration project?, (ii) what cloud-specific challenges were faced in each phase? Inspired by previous studies suggesting the worthiness of secondary data in the assessment of metamodels (Antkiewicz, Czarnecki et al. 2009; Beydoun, Low et al. 2009; Othman and Beydoun 2013), the secondary data for conducting the tracing technique was used during the case study analysis. Project documents from a variety of sources (e.g. project sequence, application architecture, and user histories) was used to obtain a better understanding of the enacted migration process model by developers.

Table 2 description of case studies

Case 1: *InformaIT* (Sweden)	Case 2: *TOAS* (Finland)	Case 3: *SpringTrader* (US)
InformaIT is a small independent software vendor involved in development of document management systems. The Document Comparison (DC) system, developed by *InformaIT*, is a Web-based enterprise solution for enhancing document management processes. DC provided a fast and easy way to compare textual and graphical contents of different digital documents. DC was originally designed to offer services to medium and large organisations which had enough resources, own infrastructure, and	TietoOyj is an IT service company that had built an open source platform called Tieto Open Application Suite (TOAS) based on J2EE technology. This platform was used for developing and running business applications in cloud environments. The TOAS platform aims to increase the development speed, automation, and the integrity of cloud applications through providing an integrated set of middle-wares, tools, and services according to service models IaaS, PaaS, and SaaS. A cloud migration project was launched by Tieto to migrate a	SpringTrader is an open-source Web-based system that has been originally developed by Pivotal company and maintained by many contributors over time. The system allows users to establish an account to view and manage a portfolio of stocks, lookup stock quotes, and buy and sell stock shares. Pivotal company had launched its own private cloud platform, which named Pivotal Cloud Foundry. The platform is an open-source platform for developing and deploying portable cloud-native enterprise systems. Pivotal decided to move

technicians to install and run the system. *InformaIT* considered promoting its competitiveness power via expanding DC's services around small companies. However, small companies couldn't afford DC as they would be confronted with a high financial commitment such as high cost of installation as well as the usage cost of users. A cloud model could facilitate an efficient and agile maintenance environment for the DC.	legacy system from the current Tieto's infrastructure to this new TOAS platform. The system had was processing batch tasks. Due to the outdated hardware infrastructure and software platform, moving this system to TOAS IaaS was a promising way to enhance the system performance and reduce infrastructure cost.	SpringTrader to this new cloud platform because it will enable (i) users to access real-time stock market data and more interactivity with the system, and (ii) the individual scaling up/down of SpringTrader's components (called micro services) and their maintainability.

Within-case analysis: InformIt case

The following paragraphs describe how the constructs in the process model are instantiated to represent activities, carried out by a development team in *InformaIT* project (Rabetski 2012). The 43-page secondary document of this project was carefully reviewed. Figure 2 represent the instance of the enacted process in *InformIT*.

As one of the first tasks, the developers performed *Preliminary Analysis* to identify benefits and challenges of migration to the cloud in terms of privacy, vendor lock-in, and environmental limitations. This activity is an instantiation of the *Analyse Migration Feasibility* in the metamodel. Additionally, a task which was called *Current DC Implementation* was performed to identify the current deployment model of DC. The model revealed that DC's customers have to take care of the infrastructure and provision of technical expertise to maintain it locally. The process model supports this activity through an instantiation of the *Recover Legacy Application Knowledge* defined in the *Plan Phase*.

The developers estimated the cost of DC migration to the cloud on the basis of server instances, storage, data transfer, storage transaction, cache, and database. They realised that the cost of DC could be down by 40 percent; that is $764.99 in the cloud model compared to $1264.99 in the legacy model when leveraging elastic scalability. The abovementioned cost analysis in *InformaIT* can be derived from the *Analyse Migration Cost* in the metamodel which is a subclass of *Analyse Context*.

Once the cloud migration was perceived as a viable solution to empower DC, the developers performed a task named *Choosing a Cloud Provider* in order to analyse three existing public cloud platforms, Amazon Web Services, Google App Engine, and Microsoft Azure. Each of these platforms could affect the cost, the quality of the architecture solution, and the required legacy code changes. The developers found that Google App Engine could not be a suitable candidate for DC since it did not support .NET applications as opposed to the Amazon AWS and Microsoft Azure that both provided such a support. Given a further analysis, the developers preferred Windows Azure platform to Amazon AWS for three reasons: (i) it would require less configuration effort, (ii) it would offer a faster deployment model, and (iii) developers had a consistent development experience for systems that were based on Microsoft technologies. *Choosing a Cloud Provider* in *InformaIT* conforms to the metamodel's construct of *Choose Cloud Platform/Provider* in the *Design Phase*.

In *InformaIT* scenario the development team performed a task called *Cloud DC Architecture* indicating how the existing legacy application components are mapped to the Microsoft Azure platform. For example, the legacy version of DC's database, which was a Microsoft SQL Server database, was replaced with SQL Azure. In the metamodel, *Cloud DC*

Architecture in *InformaIT* can be instantiated from the construct *Design Cloud Solution* in *Design Phase* of the metamodel.

Although Microsoft Azure was well suited as a target platform for the migration, the developers identified some incompatibility issues that implicated required changes in the current legacy implementation, referred to as *Identified Compatibility Issues*. Subsequently, the migration process proceeded with some changes in the legacy DC. As an example, the available Blob Storage and Queue Storage by Microsoft Azure were not compatible with regular APIs that were currently used by DC. Accordingly, legacy codes were changed for access Microsoft Azure database. As another example, DC had been developed using Microsoft .Net 2.0 that was not supported by Microsoft Azure. The action to resolve this was to update DC's framework to Microsoft .Net 3.5/4. Other incompatibility issues were session management and registration of legacy components in the cloud. The classes *Identify Incompatibilities* and *Refactor Codes* in the metamodel represent the above modifications to the DC in *InformaIT* case.

Some changes to DC were in the form of applying design principles proposed by the construct *Apply Design Principles* in the metamodel. For instance, DC was required to be portable between the local network and the cloud. To address this, the data and business layers of DC were decoupled by adding a new intermediate data assess layer in order to increase the portability of DC. That is, instead of directly data access, the business logic layer calls a data access layer interface. In *InformaIT* project, this construct was referred to as *Separate Data Layer from Business Logic Layer* which can be derived from the construct *Decouple Application Parts* as a subclass of the *Apply Design Principles* in the metamodel. As another example, DC stored megabytes of data per session which was a big overhead. Such a session size required more time for serialisation and de-serialisation. Developers applied a principle called *Becoming as Stateless as Possible* to make DC cloud-enabled. This construct is an instantiation of the principle *Make Application Stateless* in the metamodel.

It was likely that the performance of DC in the cloud was going to decrease due to latencies and unknown hardware infrastructure. In *InformaIT* project the task *Performance Experiment* was performed to execute CPU heavy code for the document processing in order to compare the execution and response time of running DC in the cloud (using a small Azure compute instance) and on-premise environment (using a local server). This experiment could identify potential performance bottlenecks in the cloud when heavy computational jobs such as digital document rendering are running. This activity was conducted in North Europe deployment location of Microsoft Azure because it was the closest geographical location to the project testing environment, located in Gothenburg, Sweden. The experiment illustrated that a proper deployment location can reduce interaction latencies and consequently, the performance. In addition, the experiment revealed that the performance of DC in the cloud is less than the local server and nine more instances of DC in the cloud are required to achieve the expected throughput. The abovementioned test in this scenario, i.e. *Performance Experiment*, conforms to the construct *Test Performance* in the *Enable Phase* of the metamodel. In *InformaIT* project, the developers felt there is no need for running other kinds of tests.

The suitability of the DC migration to the cloud also was analysed from a cost perspective. Developers built a prototype to analyse three real life scenarios that could describe how DC could benefit from the cloud services. The cost of each migration scenario was estimated on the basis of the pricing model of Microsoft Azure and cost parameters such as compute instance, relational database, storage, storage transaction, data transfer, and cache. The prototyping helped developers to make a final decision regarding the DC cloud enablement. Prototyping in this scenario can be generated as a result of performing the task *Make Prototype* in the metamodel. Regarding DP3, the analysing *InformaIT* confirmed some

relationships between the constructs defined in the metamodel. Table 3 shows the list of relationships among the metamodel's constructs that were instantiated in this migration scenario.

With-in case analysis confirmed that almost all of its accommodated constructs can be derived from the metamodel constructs, except for a new construct *Extensively Use Logging* which was not covered by any constructs in the metamodel. It was found that the metamodel has a deficiency to support this construct. According to the finding in this migration exercise, since cloud environments are a-synchronous, debugging and tracing an application in the cloud might be problematic (Rabetski 2012). Applying a logging mechanism in the architecture of the application facilitates tracing of the behaviour of the application, resource utilisation, and identifying reasons for failures in the cloud. Therefore, the *InformaIT* case refined the metamodel construct *Apply Design Principles* by adding a new subclass construct and was named by *Use Logging*. In Figure 2, this new construct is the class *Apply Design Principles*. The following definition was used to define this new construct: Use the logging mechanism to facilitate the application debug and resource monitoring when running in the cloud.

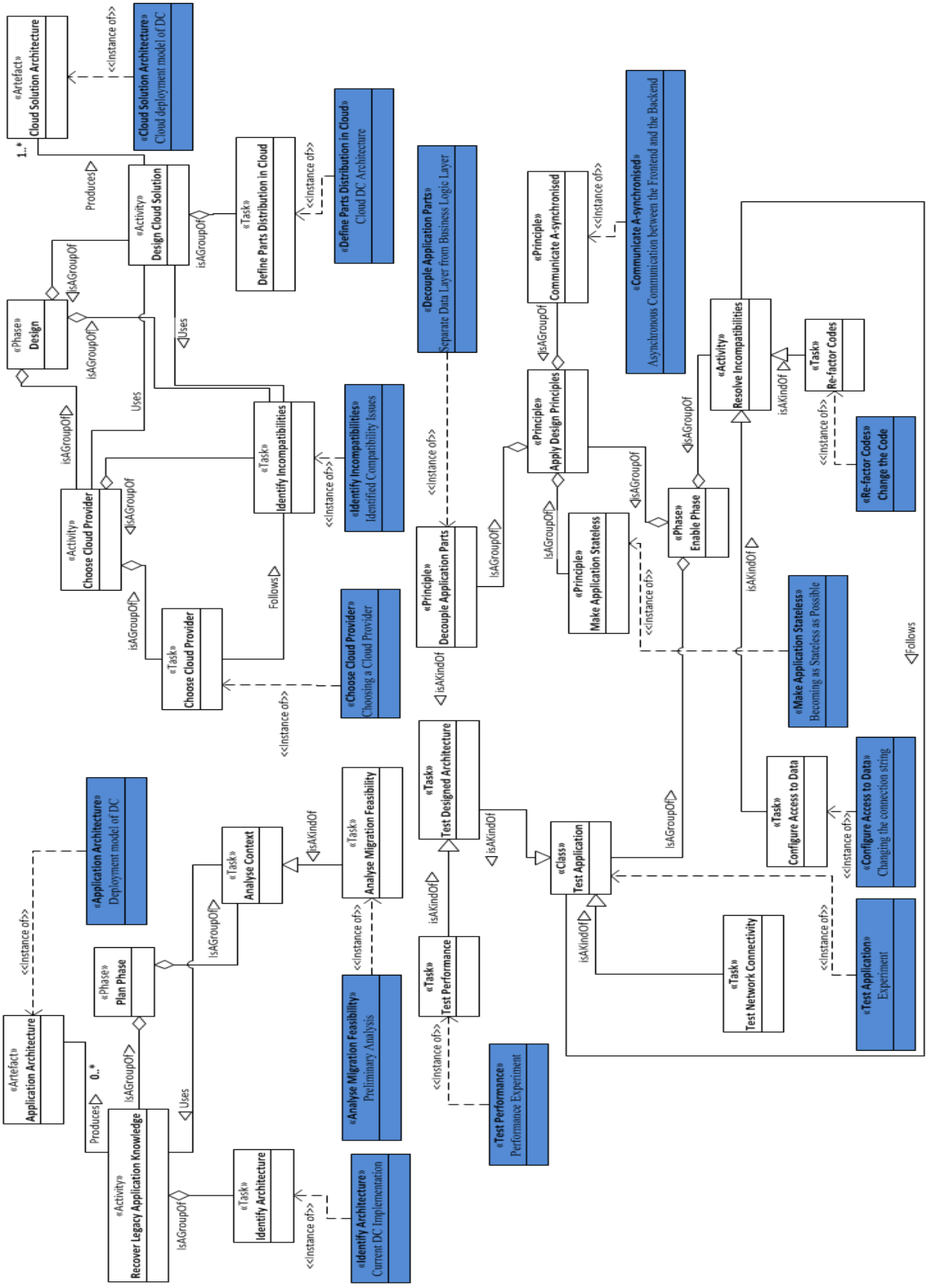

Figure 3 *InformIT* model as an instantiation of the metamodel

Cross analysis

Next, our cross-analysis compares and contrasts three cases of cloud migration process in terms of the metamodel adherence to DP1 and DP3. Table 3 shows results related to the metamodel completeness and its expressivity for producing constructs of *InformIT*, *TOAS* and *SpringTrader* migration scenarios and Table 4 shows the instantiations of some relationships in the metamodel in these cases. According to these tables, only a slice of the metamodel is required to represent domain-specific constructs which is described in the following.

As for DP1, the review of the cases *InformIT*, *TOAS*, and *SpringTrader* shows that four metamodel constructs *Recover Legacy Application Knowledge*, *Design Cloud Solution*, *Identify Incompatibilities,* and *Decouple Application Parts* were commonly instantiated in their mainstream process, to make legacy systems cloud-enabled. For example, the construct *Design Cloud Solution* defined in the *Design phase* of the metamodel was instantiated in three different ways in the scenarios. In *InformIT*, the decision on the selection and deployment of legacy system components on cloud servers was basically a mapping between Microsoft-based legacy components into their counterparts in Microsoft Azure cloud platforms. In TOAS, the legacy components were classified into two logical groups of platforms on the basis of similar functional behaviours. In *SpringTrader*, those components that provided financial and market functionalities/data were selected for the migration purpose. These are an instance of *Design Cloud Solution* defined in the *Design phase* of the metamodel.

Migration scenarios were performed very differently and therefore they were not same in the instantiation of the metamodel constructs. Except for *InformIT*, in both *TOAS* and *SpringTrader* scenarios activities related to handling incompatibility issues are performed. In *TOAS* case, developers implemented run-time adaptors to hide incompatibilities of message formats and API's support between the legacy and TOAS platform. Comparably, in *SpringTrader* case, developers implemented wrappers to separate incompatibilities between micro service and the legacy system. These techniques are subsumed under the construct *Develop Integrators* defined in the *Enable phase* of the metamodel. Additionally, unlike the instantiation occurrence of the construct *Choose Cloud Provider* in the scenario *InformIT* where developers decided to use Windows Azure cloud platform due to their experience with Microsoft-based programming platforms, the target cloud platform in both scenario *TOAS* and *SpringTrader* was a pre-chosen private cloud platform and therefore there was no need for the instantiation of the construct *Choose Cloud Provider*.

Furthermore, as for DP3, the case studies confirmed some relationships between the constructs defined in the metamodel. With respect to DP3, **Error! Reference source not found.** shows the list of relationships among the metamodel's constructs that were instantiated in the cases. As shown in **Error! Reference source not found.** in each case reviewed, only a slice of the metamodel was required to instantiate to represent relationships. **Error! Reference source not found.** shows the list of relationships that were identified during the case analysis.

The second and third case studies did not lead any refinements to the metamodel and all their constructs were producible using the metamodel constructs. As we progress through the case analysis, the coverage of the metamodel on enacted process models is a strong indicator of the metamodel adherence to DP1 and DP3. Nevertheless, the metamodel adherence to DP1 and DP3 cannot be statistically generalized based on the results of case analysis and hence there is a possibility of extending the metamodel to new constructs or relationships if more case studies are performed. This will be discussed further in this paper.

Table 3 Support of constructs in the migration scenarios by the metamodel

Metamodel Construct	InformaIT	TOAS	SpringTrader
Recover Legacy Application Knowledge	A distributed deployment model of DC was identified. DC included five components namely frontend web, application, backend engine, distributed cache, database, and v) shared file store. InformaIT rents several virtual private servers. However, the servers could not be scaled dynamically and they became underutilized most of the time.	The legacy system architecture recovered and documented in order to identify its hardware requirements, running components on middle wares and servers, their settings and computational requirements. In addition, legacy dependencies, features such as hardware requirements, running components on middle wares and servers, their settings and computational requirements were identified.	It was found that the legacy system architecture was fairly simple with a front end that includes the Web layer talking to a set of HTTP/JSON-based services where stock quotes and portfolios could be viewed, and stock trade orders may be submitted, and a back end that fulfils orders. The communication between the front and back ends was asynchronous with the front end delivering orders to a message queue and the back end consuming from that queue. Both the front end services and the back end also access a shared relational database.
Choose Cloud Provider	Three existing public cloud platforms namely Amazon Web Services, Google App Engine, and Microsoft Azure compared on the basis of required cost for changing in DC and architecture quality. Standard rate of cloud platform alternatives such as price for computation, virtual network, storage, content delivery network, caching, service bus, data transfer, and access control were identified and analysed. Windows Azure was chosen as it needed less configuration effort, faster deployment model, less training effort.	Not instantiated (pre-chosen private cloud, i.e. TOAS platform)	Not instantiated (pre-chosen private cloud, i.e. Pivotal Cloud Foundry)
Design Cloud Solution	Microsoft-based DC's components (e.g. database) were mapped to their counterpart in Microsoft Azure platform and accordingly modified to Azure cloud services.	The legacy components were classified on the basis of their similar functional behaviours into two logical groups of platforms.	Those components that provided financial and market functionalities/data were selected for the migration purpose. Also, micro service architecture was used to distribute legacy components in Cloud Foundry.
Identify Incompatibilities	There were some incompatibilities between current technologies used in DC and legacy and Microsoft Azure such as different versioning between platforms, APIs, and session management.	Legacy components were based on the technologies belonging to middle of 2000 and constitute a lot of legacy codes. It was identified that are different versioning in legacy APIs and TOAS.	As SpringTrader was written for JDK6 and Spring 3 and the current version of Cloud Foundry PaaS was JDK8 and Spring 4, there were some incompatibilities in the libraries of these two environments. These include

			the bytecode manipulation used by the annotation processing, and the use of some obsolete JSON libraries. To resolve incompatibilities, developers upgraded the application's libraries.
Decouple Application Parts	The legacy was needed to be portable across on premise and cloud platform, it was decided to separate the business logic and data layer of DC. Decoupling also facilitated using on premise file system and Azure Storage depending on the chosen deployment environment.	Loose coupling was performed to facilitate component scaling up/down and fault management. Decoupling was applied by replacing all remote method invocation (RMI) based communications in the legacy with XML-based service in TOAS cloud.	*Micro-service architecture design* was used to decouple legacy components. In addition, a service discovery mechanism was implemented to enable the legacy system and developers to locate micro services by name at a known catalogue endpoint and look them up dynamically at runtime.
Adapt Data	DC used a Microsoft SQL Server database. It was replaced with SQL Azure. In most cases switching to SQL Azure was not a big task but it was required to update connection setting to the new database.	Not instantiated.	Different types of cloud database solutions were used in this project such as MySQL, MongoDB, and relational SQL database. In order to address incompatibilities between these cloud services, the notion of boundary context was used in the sense that transition data were packed and unpacked during executing transactions.
Develop Integrators	Not instantiated.	Different kinds of run time adaptors were developed to hide incompatibilities of message formats and API's support between the legacy and TOAS platform.	Warpers were implemented to hide incompatibilities between micro service and the legacy system.
Refactor Codes	Not instantiated.	Not instantiated.	Quote simulation functionality was refactored from the SpringTrader legacy in order to be exposed as a new micro service, i.e. Quote Web-Service. It was a simple service that used the public Yahoo Finance APIs to provide real-time market data. Such refactoring freed the developers to choose any technologies that could make sense on basis of requirements regardless of ripple effect in the existing legacy codes. To refactor the legacy code some steps such as locate the code, identify a service interface, use the proxy pattern, create an

			implementation of the interface, and point the monolith to the new service were performed.
Re-configure Network	Not instantiated.	Firewall rules and subsequently application endpoints were reconfigured. Firewall configuration was a burdening task and included finding, setting up, testing, and maintaining the required firewall rules.	Not instantiated.

Table 4 instantiations of metamodel relationships in the scenarios

Relationship Name	Construct 1	Construct 2	Migration scenario		
			InformaIT	TOAS	SpringTrader
Uses	Design Cloud Solution	Analyse Migration Requirements	Not instantiated	√	Not instantiated
Uses	Design Cloud Solution	Identify Incompatibilities	√	√	√
Uses	Design Cloud Solution	Choose Cloud Provider	√	√	√
Uses	Refactor Codes	Identify Incompatibilities	√	-	√
Uses	Design Cloud Solution	Recover Legacy Application Knowledge	Not instantiated	Not instantiated	√
Uses	Refactor Codes	Design Cloud Solution	√	Not instantiated	√
Uses	Migrate Database	Refactor Codes	√	-	-
Uses	Test Application	Design Cloud Solution	√	√	
Follows	Plan Migration	Design Phase	√	√	√
Follows	Design Phase	Enable Phase	√	√	√
Follows	Choose Cloud Provider	Identify Incompatibilities	√	√	√

Evaluation

kkkk

Step 1. Domain experts feedback

The metamodel was qualitatively examined by a panel of four domain experts regarding DP1, DP2, and DP3. The experts are denoted by E1, E2, E3, and E4 in this study. The overall experts' feedback was promising and valuable. The list of questionnaire form and details of the feedback from the experts are available in the Supplementary document Appendix G and H, respectively. The usefulness of the metamodel was stated by the words such as "education and high-level guidance" (E1), "good communication vehicle" and "more comprehensive list of concerns" (E2). E2 stated that "the model is clearly valuable in conveying the important concerns of a migration and how they are related. The detailed semantics help to clearly understand dependencies and possibly resulting decisions and trade-offs to be considered". A similar opinion was expressed by E3. He said "this model can make a good impact to increase the confidence of success factor of the migration process and decrease some uncertainty. Also, this model can be used as a checklist of success migration and this reference model makes an overall picture of migration phase and clears the roadmap for

audiences to do the migration with less stress and concerns". The advantage of the metamodel against existing migration process models was stated by E4 "I have mostly used the classical reengineering model for legacy migration. In comparison to the model by SEI, the proposed model is more detailed in terms of underlying process and activities for migration". Experts provided some suggestions for the improvement of the metamodel in relation to the design principles. The following is an explanation of the metamodel refinements as a consequence of each expert's feedback.

Regarding DP1, an area of concern raised by E2 was that he believed "determining licensing issues of legacies should be made more visible in the metamodel as it can turn out to be a major task in the migration process". In cloud environments, multiple instances of a system, which is encapsulated into a virtual machine, might be created by a server based on the workload or rules are triggered to run resource scaling. This may cause an unintended violation of the licensing agreement that has been made between the owner and user of the system. The above comment by E2's has been partially covered in the definition of the construct *Analyse Migration Cost* in the initial metamodel. However, it has not been considered as an individual construct in the metamodel. Utilising the knowledge source prepared in the phase one of design science process, a new construct named *Resolve Licensing Issues* as a special construct was added in *Design Phase* of the metamodel to explicitly represent this construct (Figure 2). It is defined as follow: *Define and monitor a pay-as-you-go licensing model to handle unintended license agreement violations due to automatic scaling.*

E3 explained that the metamodel lacks a construct called "roll-back: I have observed that migration process model should contain a construct to show rollback for the migration process". To address this concern, the metamodel was refined by extending the construct *Define Plan* to *Define Roll-Back Plan* and defining a new relationship in the metamodel (Figure 2). A definition for this construct regarding the knowledge source was decided as "Define roll-back, as a B plan, to an in-house version of the legacy application in the case of occurrence of any significant risk or new application fails during the migration process. This reduces the risk and exposure to the business".

The experts provided some comments related to DP2. From E2's point of view, the notations and visualisation used to represent the metamodel were found unclear: "UML is not used by all stakeholders". Likewise, E4 mentioned "a unified high-level block diagram for the reference model (unifying all those three different phases) must be presented for better illustration or reflection of the model". As a responded to the above comment, a preliminary version of the metamodel was made using simple block diagrams. However, such a representation could be used simply for process documentation purposes. If the metamodel is going to be an integral part of model-driven development and OMG metamodeling framework (Atkinson and Kuhne 2003), a semi-formal representation of the metamodel becomes important when the migration scale is large. In this spirit, UML is a de-facto standard for the conceptual representation of a particular domain in terms of organising constructs, their relationships, and decidable reasoning. Furthermore, UML is used only to represent the metamodel for the purpose of illustration and its graphical representation is presented later in the prototype system (next section). With respect to DP3, there was no major comment made by the experts.

Step 2. Prototype System

In this step, a prototype system of the refined metamodel from the previous step was implemented in order to appraise the metamodel efficacy with respect to the design principles. Through this prototype it was also possible to examine the adherence of the

metamodel to DP4, i.e. creating situation method for a given cloud migration scenario through instantiation, reusing, and configuration of the generic constructing in the metamodel. The prototype was built on the guidelines proposed in (Ralyté, Deneckère et al. 2003) for the situational method engineering approach. The prototype comprises two components: (i) a *repository* which stored the metamodel constructs, their definitions, relationships among them, and relation to migration types and (ii) a generic three-step *procedure* for the metamodel instantiation and customisation. The input to the procedure are parameters of a migration scenario mainly selected cloud service delivery model (migration types) and migration phases such as *Plan*, *Define*, and *Enable*. The sourced input parameters, provided by a method engineer, are used to select relevant constructs of the metamodel which are stored and retrieve from the system repository.

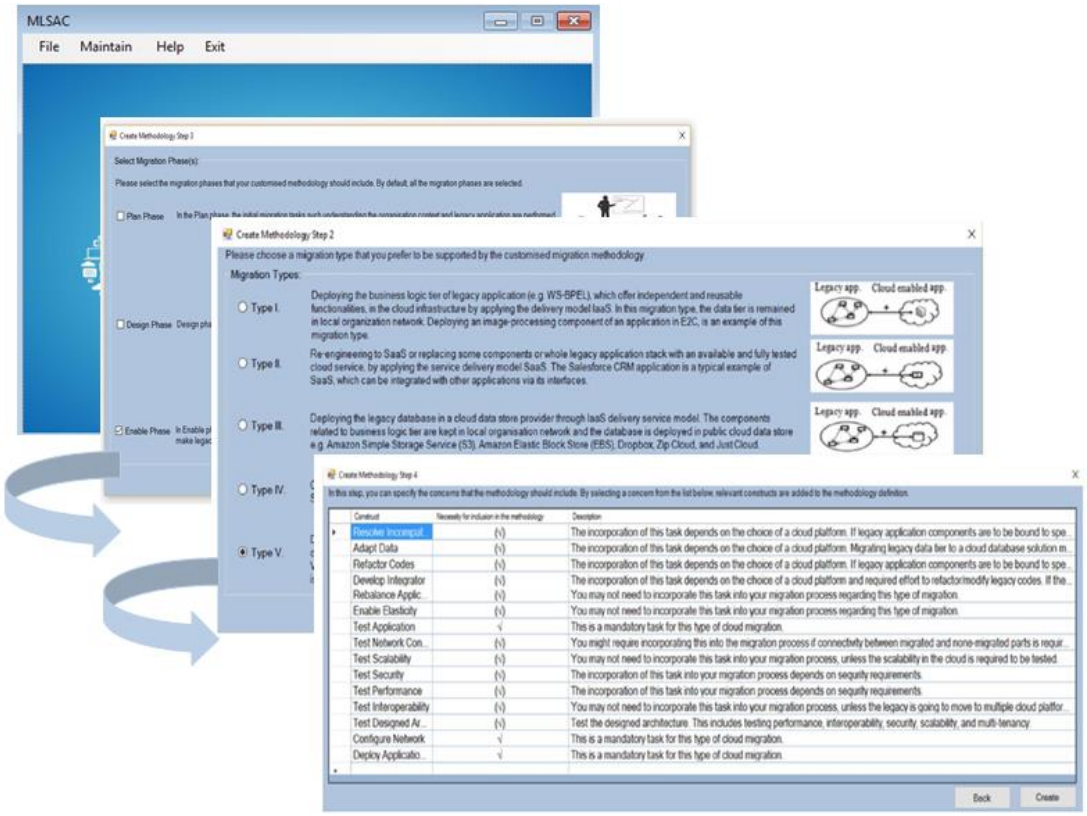

Figure 4 choosing a migration type for a target method

Once the initial method is created, the system shows constructs which are mandatory, situational, or unnecessary to be carried out in the method (the fourth form in Figure 6). Figure 7 shows a snapshot of a created sample method. The method engineer can browse through the method, which contains a set of relevant constructs and their definitions reused from the metamodel. The graphical user interface in Figure 7 has three main sections. The upper part contains the method name (in this fictitious example LegacyMigrationtoAmazonEC2), the migration type, and a general description of the method. The bottom-left part shows the constructs of the method reused from the metamodel. The method is rendered using a Microsoft .Net Tree View Control which is a common control to visualise complex data structures. The bottom-right part contains the information about a construct once the method engineer clicks on it in the tree view. Different icons are used in the prototype system to illustrate the classification of a method's constructs such as phases, tasks, and work-products.

Different functions may be performed to accommodate migration method needs such as (i) adding new constructs to the created method in the cases where the pre-existing constructs in

the in the repository are insufficient for the representation of a new particular construct for a given migration scenario (ii) extending existing constructs with new sub-constructs through the notion of inheritance in object-oriented software design, (iii) adding alternative techniques to guide developer in how to operationalise the abstract constructs, and (iv) defining a specific flow among constructs in the method to show how they are sequenced.

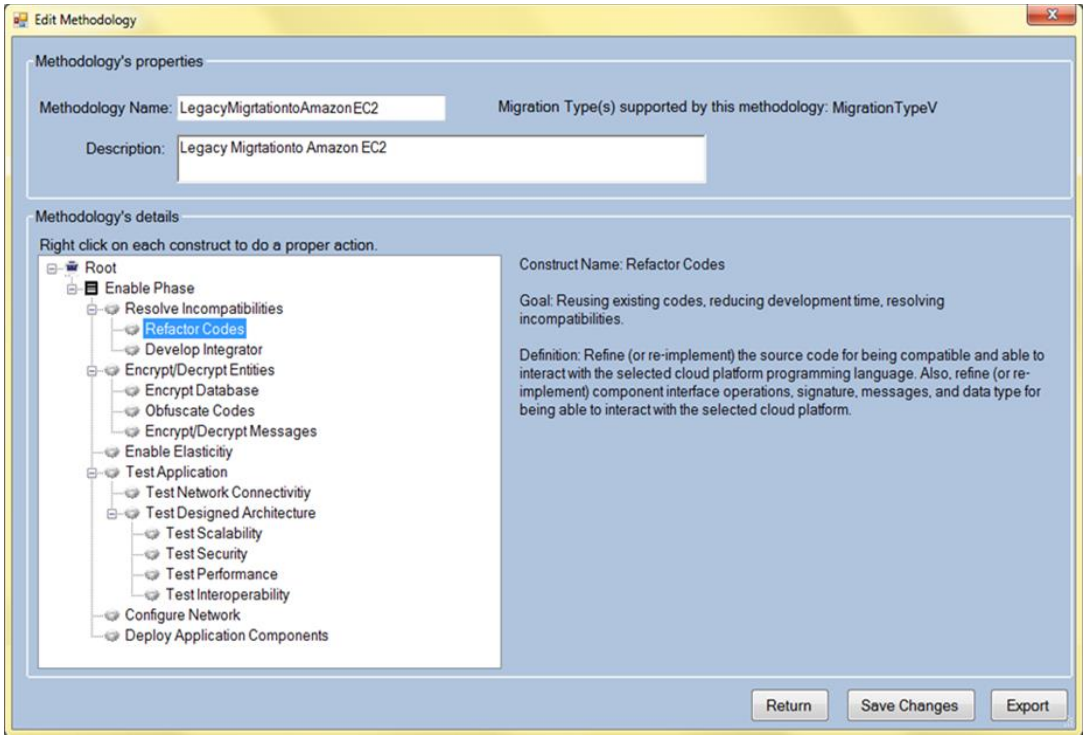

Figure 5 a created method for enable phase of migration type

The last optional step is to export the method as an XML document (Mendling and Nüttgens 2006). Using the XML document in the prototype facilitates computer-readability and interoperability of produced methods across modelling tools. Figure 8 shows the conceptual structure and corresponding XML representation of a created method. Later on, developers can import this base method, reuse, and tailor it for a given migration scenario. The developers can specialise the method through new constructs or operationalisation techniques using complementing method or previous cloud migration experience.

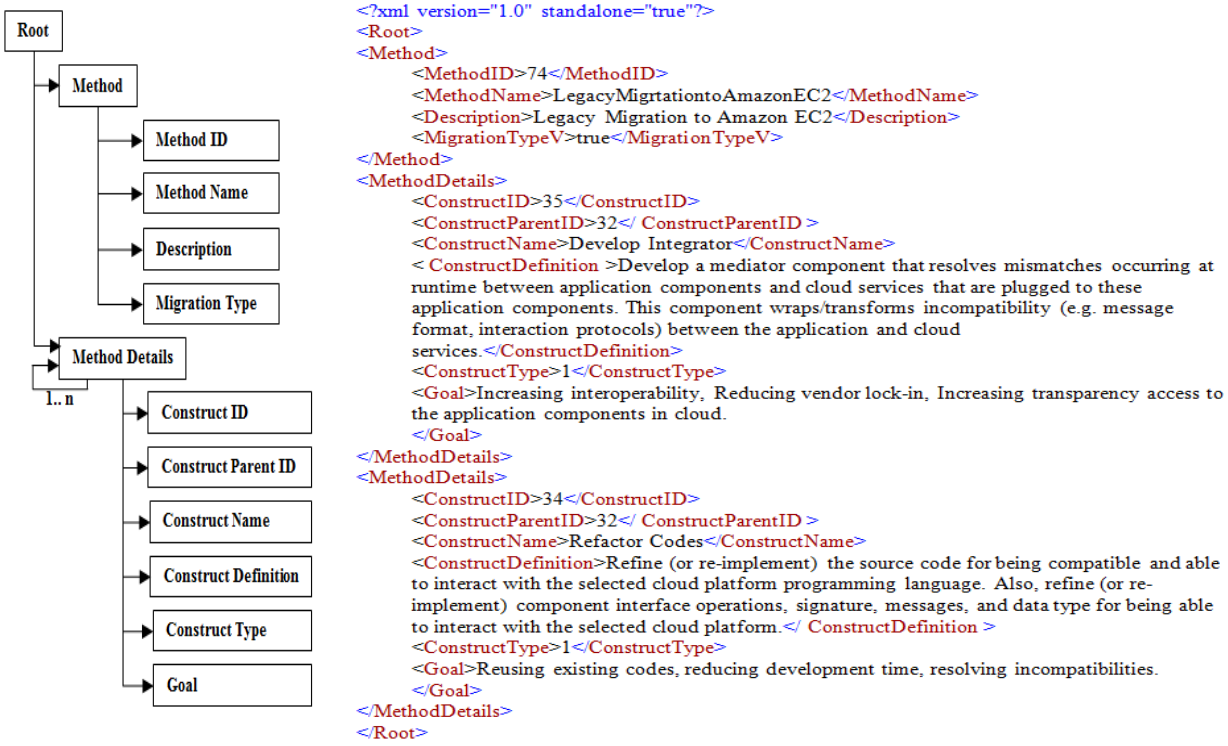

Figure 6 an excerpt of a created method described in an XML format

The prototype was examined by two experts. They worked with prototype, ran three-step tailoring procedure, and expressed their opinions about the suitability of the metamodel regarding DP1, DP2, DP3, and DP4. Two cloud computing experts, denoted by E5 and E6, one project manager and one technical lead were recruited and asked to provide feedback on the prototype. They have had experience in moving geosocial networking and finance applications to cloud environments, respectively. Each expert was asked to nominate one cloud migration scenario in which they already had actively participated.

The overall feedback from the experts was positive along with some suggestions for further improvement of the metamodel. Both experts primarily mentioned that providing a rich repository of constructs and flexibility for extending them with new ones are excellent features of the metamodel. They believed the metamodel will have a positive effect on the quality of the cloud migration process. E6 highlighted that the framework is helpful for practitioners who may not be familiar with the cloud migration concepts. They also mentioned that the metamodel mitigates missing constructs that are important for consideration during the migration process. Furthermore, E6 stated that the metamodel's ability in moving from text-based to a standard representation of methods facilitates method integration with other process modelling tools. Regarding the adherence to the DP1, experts alike agreed that the metamodel covers major constructs relevant to the cloud migration process.

The experts provided some suggestions for improving the user interface design of the prototype. For example, E5 suggested adding a drag and drop feature for moving constructs among phases. Both experts acknowledged the clarity of names and definitions (DP2) and classification of constructs based on names and the migration types (DP3). The experts also provided positive feedback about DP4 and confirm the suitability of the metamodel tailorability for a given scenario; however, they suggested adding pre-built reusable templates relevant to particular cloud migration domains such as mobile computing, finance, or insurance in order to create immediate methods through the process of method tailoring for

specific domains. Additionally, the experts uttered improving the prototype to allow multiple users concurrently work on the method under enactment and maintenance, and if a user changes the method content, the framework can integrate this change into new a version of the method. E6 suggested adding warning messages when users define an illogical sequence among activities. However, adhering to some of the above suggestions are trivial to the main objectives of the current research and constitute as the future work.

Discussion

Implications for research

Aimed at alleviating the problems afflicting the process aspect of the cloud migration, this research contributes to the cloud computing literature in several ways. Firstly, researchers have attempted to develop intellectual models to demystify multifaceted yet ambiguous nature of cloud computing technology from perspectives architecture of cloud computing (Hamdaqa and Tahvildari 2012; Zimmermann, Pretz et al. 2013), green cloud computing (Procaccianti, Lago et al. 2014), quality aspects of cloud services (Nunez, Fernandez-Gago et al. 2013), eases code refactoring (Ardagna, Nitto et al. 2012), reducing risk and compliance efforts for cloud computing (Martens and Teuteberg 2011; Keller and König 2014). Compared to these studies and in response to call made by previous research for engaging scholar to enhance the methodological aspect of cloud migration (Jamshidi, Ahmad et al. 2013; Fahmideh, Daneshgar et al. 2016), a key theoretical contribution of this research is to shed light into the cloud adoption from the process perspective by developing an overarching, yet customizable, metamodel of moving legacy systems to cloud platforms. The proposed metamodel hides away the multifaceted and dispersed area of cloud computing migration from operationalisation details that are only relevant to the domain-specific cloud computing adoption by providing an abstract representation of core constructs in the domain. It facilitates the understanding of cloud migration process for both IS researchers and practitioners since it represents a single and unified model instead of looking for such a model in the existing sporadic and fragmented literature. It can also educate newcomers to the cloud computing field to envision the way through which organisation can move their legacy systems to cloud platforms.

Secondly, the proposed metamodel can be viewed as a knowledge sharing platform to assist consistent communication between scholars since they can use the metamodel as a reference process model, allowing effective knowledge transfer across the community, which has been the concern of the current literature in the field (Hamdaqa and Tahvildari 2012). The metamodel will fertile theoretical grounding for future research and is as a potential candidate for addressing the need for future cloud standardization as raised by (Dillon, Wu et al. 2010; Ortiz Jr 2011).

Thirdly, previous research largely assumes that the cloud migration process is monomorphic, i.e. a single migration method is enough. When the methods reviewed in section Related Work are viewed collectively, they define a set of fixed activities for reengineering and integrating existing legacy systems with cloud services, though they may differ in their scope, operationalization details, and application domain. While there are merits to adopting these methods, some research has started to argue that cloud migration methods need to be tailored prior to their enactment based on a chosen cloud platform, migration type (e.g. IaaS, PaaS, or SaaS), reusability and quality of legacy system source code, system scalability and security requirements (Mohagheghi, Berre et al. 2010; Menychtas, Santzaridou et al. 2013). For instance, a method might be a better fit for process-intensive and distributed workloads from legacy data centers to public IaaS whilst another method maybe an adequate option for

making a legacy system SaaS-enabled. While such characteristics circumscribe the method suitability of a method for a given scenario, this research integrated extant migration solutions into a generic metamodel, supporting a pluralist view of the cloud migration process and yet customizable and extensible.

Implications for practice

Pertaining to practical values of this research IS practitioners may benefits from the results of this research in the following ways. Previous research (Yang and Tate 2012; Fahmideh 2016) state an urgent demand for model explaining cloud computing technologies in simple and friendly language since existing migration methods mostly focus on specific technical details of utilization of cloud services but do not offer practical knowledge to IS executives who may find difficult to fully comprehend, digest, synthesize, and put this voluminous and unstructured cloud migration body of knowledge for a specific migration purpose. A better understanding and articulation of the cloud migration process can facilitate identification of appropriate activities, anticipation, cost, and expected outcomes of a migration exercise. In response to this issue, the current research extends the literature by proposing a metamodel acting as an abstract view of cloud migration domain and flexible for extension and helping IS executives to gain the knowledge of what important business and technical activities that should be incorporated during moving legacy systems to cloud platforms.

Secondly, an IT-based organisation may have its own arbitrary off-the-shelf or in-house method for a technology shift but it does not applicable for cloud computing migration. An important application of results in this research is that metamodel syntheses commonly encountered constructs to provide a rich source that can be integrated with existing methods, as an extension, to enhance their capability to support cloud migration.

Thirdly, as is the case in any IS projects, method designers need to select appropriate methods that fit characteristics of a given cloud migration scenario. This effort cannot be facilitated unless by adopting a suitable tool to analyse and evaluate existing methods in terms of their features, shortcomings, strengths, similarities, and differences. According to Siau and Rossi (Siau and Rossi 1998), one effective way of comparing family-related methods, herein cloud migration process, is to use metamodels as a basis for analysis because they take place at one level of abstraction and capture information about methods. As mentioned earlier, cloud migration literature points to a dozen of methods for cloud adoption such as Chauhan's Method (Chauhan and Babar 2012), REMICS (Mohagheghi 2011), Tran's Method (Tran, Keung et al. 2011), Cloud-RMM (Jamshidi, Ahmad et al. 2013), Strauch's Method (S. Strauch 2014), Zhang's Methodology (Zhang, Chung et al. 2004), Oracle Method (Laszewski and Nauduri 2011), ARTIST Method (Menychtas, Santzaridou et al. 2013), Amazon Method (Varia 2010), Legacy-to-Cloud Migration Horseshoe (Ahmad and Babar 2014), IVI Cloud Computing Life Cycle (Conway and Curry 2013), and MILAS (Huru 2009). As the applicability of these methods is limited by characteristics of a migration scenario, project managers can use the list of constructs in the metamodel as a checklist to examine and select an appropriate existing method to match a given migration scenario.

And finally, the metamodel is a useful source used by project managers to estimate required development effort to make legacy systems cloud-enabled. The metamodel constructs provides heuristics that facilitates analysing required migration effort and, accordingly, it can be used as an input for the studies by (Tran, Keung et al. 2011) and (Quang Hieu and Asal 2012) suggesting a migration cost estimation based on development tasks are required to be performed.

Limitations of the study

The current study has three limitations. Firstly, a potential weakness of the case studies is their specificity to a context which limits generalisability of the results to other applications and contexts (Benbasat, Goldstein et al. 1987). Although the applicability of metamodel was illustrated in three idiosyncratic cases with different characteristics, the complete satisfaction of the DP1 and DP3 can still be subject to some arguments. The metamodel is only a representative of the constructs of the three case studies and there is a possibility of being extended by introducing new constructs or relationships if more case studies are performed. In this regard, this research acknowledges that to increase the generalisability of the metamodel as a domain language for cloud computing adoption, it should be appraised with more cases in a variety of cloud migration contexts. By the word *context*, this research implies different industries, migration scale (partial or full), and organisation size (.i.e. small start-up, medium-sized organisation, and big organisations) which help identifying possible improvements of the framework components.

Secondly, the current study acknowledges the limitation of retrospective studies (Hess 2004). Since it was not possible to have a detailed record of each activity of migration scenario, this research relied on the accuracy of the written documents about the cloud migration scenarios, which might not have documented some valuable constructs that were related to the cloud migration. Hence, examining the adherence of the metamodel to the design principles has been subjected to the quality of available documents of the migration scenarios. There is a possibility of missing some constructs that could be added to the metamodel to increase its domain coverage. This threatens the examining of the metamodel adherence to DP1 and DP3. To alleviate this issue, we conducted follow up communications with interviewees to confirm the validity of the secondary documents of the case studies and provide any missing information.

Thirdly, the refinements to the metamodel have been based on the opinions from six experts which might have been biased and confined by their own experience and knowledge in relation to the cloud migration. Therefore, the satisfaction of design principles might have been affected by this threat. Receiving feedback from a larger number of experts in the cloud migration area in future will reduce this threat. Finally, although the steps for the metamodel tailorability was showed through the prototype system, there is no claim regarding adherence to the DP4 as it can be extended with new steps as briefly pointed out in the next section.

Conclusion and future research

This study was justified with the lack of an integrated and abstract domain language for efficient creation, configuration, standardisation and sharing knowledge of cloud migration processes that cater to specific cloud migration scenarios. In addressing this gap in extant literature, a generic, domain-independent, and tunable metamodel was developed and evaluated that constitutes reusable domain constructs incorporated into the cloud migration process. It was performed by case studies as the context of data collection, domain experts as source of data, and a prototype system as the research tool. The identified limitations of the current study lead to several directions for further extension of this work. Some of future studies are described in the following.

One is to augment the metamodel to encompass new constructs relevant to post-migration and during the system maintenance in the cloud, for example continuous integration and delivery. Similarly, the metamodel can be extended to other particular domains of cloud computing. An example of that is *mobile cloud systems* which is a growing area of the cloud computing field (Giurgiu, Riva et al. 2009; Dinh, Lee et al. 2013). Mobile cloud applications

are run on mobile devices, utilise cloud services, and characterised with challenges such as battery life, bandwidth, heterogeneity, and privacy in mobile environments. The metamodel can be extended with constructs related to the development of such system. The UML formalism used in this study for representation of the metamodel facilitates inclusion and representation of new further constructs in a structured way.

Secondly, depending on the characteristics of a cloud migration scenario, the creation of situational cloud migration methods may involve other factors that have not been covered by the current version of the metamodel and prototype system. Factors such as code refactoring cost, the choice of a target cloud platform, the pricing model of cloud providers, the capability of the development team, and time to market which may influence the steps of method tailoring procedure. In addition, a method tailoring effort may involve making trade-offs among different factors or cloud migration goals which may be in contradiction or have dependencies with each other. The trade-off analysis of alternative migration methods can be performed on the basis of the results of an aggregated analysis of methods. As a further work, one can utilise the idea of goal-driven situational method tailoring suggested in (Cesar and Paolo 2009; Karlsson and Ågerfalk 2011) as a baseline in order to strengthen the metamodel tailorability for addressing such complex situations during a tailoring effort.

Given the inclination of IT-based organizations towards empowering their legacy systems with cloud computing services, this study strives to facilitate consistent knowledge sharing and exchange about cloud migration processes as the proposed metamodel generalizes common domain constructs that are typically incorporated into a cloud migration process. It also enables method engineers to create and share new customised cloud migration methods on the basis of selecting constructs from the metamodel and characteristics of a migration scenario. We expect that this research will motivate other researchers to further explore new cloud migration approaches which systematically simplify the cloud migration process.

References

Ahmad, A. and M. A. Babar (2014). A framework for architecture-driven migration of legacy systems to cloud-enabled software. Proceedings of the WICSA 2014 Companion Volume. Sydney, Australia, ACM**:** 1-8.

Ambler, S. W. (2005). The Elements of UML (TM) 2.0 Style, Cambridge University Press.

Antkiewicz, M., K. Czarnecki, et al. (2009). "Engineering of framework-specific modeling languages." IEEE Transactions on Software Engineering **35**(6): 795-824.

Antkiewicz, M., K. Czarnecki, et al. (2009). "Engineering of framework-specific modeling languages." Software Engineering, IEEE Transactions on **35**(6): 795-824.

Ardagna, D., E. Di Nitto, et al. (2012). Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. Modeling in Software Engineering (MISE), 2012 ICSE Workshop on, IEEE.

Ardagna, D., E. D. Nitto, et al. (2012). MODAClouds: a model-driven approach for the design and execution of applications on multiple clouds. Proceedings of the 4th International Workshop on Modeling in Software Engineering. Zurich, Switzerland, IEEE Press**:** 50-56.

Armbrust, M., A. Fox, et al. (2010). "A view of cloud computing." Communications of the ACM **53**(4): 50-58.

Atkinson, C. and T. Kuhne (2003). "Model-driven development: a metamodeling foundation." Software, IEEE **20**(5): 36-41.

Bain, A., J. Mitchell, et al. (2011). A domain-specific language for computing on encrypted data (invited talk). LIPIcs-Leibniz International Proceedings in Informatics, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Benbasat, I., D. K. Goldstein, et al. (1987). "The Case Research Strategy in Studies of Information Systems." MIS quarterly **11**(3): 369-386.
Beydoun, G., G. Low, et al. (2009). "FAML: a generic metamodel for MAS development." Software Engineering, IEEE Transactions on **35**(6): 841-863.
Brandic, I., S. Dustdar, et al. (2010). Compliant cloud computing (c3): Architecture and language support for user-driven compliance management in clouds. Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, IEEE.
Cesar, G. P. and G. Paolo (2009). "Method construction by goal analysis."
Chauhan, M. A. and M. A. Babar (2012). Towards Process Support for Migrating Applications to Cloud Computing. Cloud and Service Computing (CSC), 2012 International Conference on.
Cimato, S., E. Damiani, et al. (2013). Towards the certification of cloud services. Services (SERVICES), 203 IEEE Ninth World Congress on, IEEE.
Conway, G. and E. Curry (2013). The IVI Cloud Computing Life Cycle. Cloud Computing and Services Science, Springer**:** 183-199.
Cuadrado, J. S. and J. G. Molina (2009). "A model-based approach to families of embedded domain-specific languages." IEEE Transactions on Software Engineering **35**(6): 825-840.
Dillon, T., C. Wu, et al. (2010). Cloud computing: Issues and challenges. Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, Ieee.
Dinh, H. T., C. Lee, et al. (2013). "A survey of mobile cloud computing: architecture, applications, and approaches." Wireless communications and mobile computing **13**(18): 1587-1611.
Dougherty, B., J. White, et al. (2012). "Model-driven auto-scaling of green cloud computing infrastructure." Future Generation Computer Systems **28**(2): 371-378.
Fahmideh, M., F. Daneshgar, et al. (2016). "Cloud migration process—A survey, evaluation framework, and open challenges." Journal of Systems and Software **120**: 31-69.
Fahmideh, M., F. Daneshgar, et al. (2016). "Cloud Computing Adoption: An Effective Tailoring Approach." The 27th Australasian Conference on Information Systems (ACIS) **In press**.
Fahmideh, M., Low Graham, Ghassan Beydoun (2016). "Conceptualising Cloud Migration Process." Twenty-Fourth European Conference on Information Systems (ECIS), İstanbul,Turkey, 2016 **1**: 0.
Fehling, C., F. Leymann, et al. (2012). "Pattern-based development and management of cloud applications." Future Internet **4**(1): 110-141.
Fehling, C. and R. Retter (2011). Cloud computing patterns, URL: http://cloudcomputingpatterns. org.
Giurgiu, I., O. Riva, et al. (2009). Calling the cloud: enabling mobile phones as interfaces to cloud applications. Middleware 2009, Springer**:** 83-102.
Gonzalez-Perez, C. and B. Henderson-Sellers (2008). Metamodelling for software engineering, Wiley Publishing.
Gregor, S. and A. R. Hevner (2013). "Positioning and presenting design science research for maximum impact." MIS Quarterly **37**(2): 337-356.
Hamdaqa, M., T. Livogiannis, et al. (2011). A Reference Model for Developing Cloud Applications. CLOSER, Citeseer.
Hamdaqa, M., Livogiannis, T., Tahvildari, L. (2011). A Reference Model for Developing Cloud Applications, In: Proceedings of CLOSER 2011.
Hamdaqa, M. and L. Tahvildari (2012). "Cloud computing uncovered: a research landscape." Advances in Computers **86**: 41-85.
Harmsen, A. F., J. Brinkkemper, et al. (1994). Situational method engineering for information system project approaches, University of Twente, Department of Computer Science.
Hess, D. R. (2004). "Retrospective studies and chart reviews." Respiratory care **49**(10): 1171-1174.
Huru, H. A. (2009). "MILAS: ModernIzing Legtacy Applications towards Service Oriented Architecture (SOA) and Software as a Service (SaaS)."

Isard, M. and Y. Yu (2009). Distributed data-parallel computing using a high-level programming language. Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM.

Jamshidi, P., A. Ahmad, et al. (2013). "Cloud Migration Research: A Systematic Review." Cloud Computing, IEEE Transactions on **PP**(99): 1-1.

Jonkers, H., M. Stroucken, et al. (2006). Bootstrapping Domain-Specific Model-Driven Software Development within Philips. 6th OOPSLA Workshop on Domain Specific Modeling (DSM 2006), Citeseer.

Karlsson, F. and P. J. Ågerfalk (2011). "Towards Structured Flexibility in Information Systems Development: Devising." Theoretical and Practical Advances in Information Systems Development: Emerging Trends and Approaches: Emerging Trends and Approaches: 214.

Karlsson, F. and P. J. Ågerfalk (2012). "MC Sandbox: Devising a tool for method-user-centered method configuration." Information and software technology **54**(5): 501-516.

Keller, R. and C. König (2014). "A Reference Model to Support Risk Identification in Cloud Networks."

Kelly, S. and R. Pohjonen (2009). "Worst practices for domain-specific modeling." Software, IEEE **26**(4): 22-29.

Kitchenham, B., O. Pearl Brereton, et al. (2009). "Systematic literature reviews in software engineering – A systematic literature review." Information and software technology **51**(1): 7-15.

Kopp, O., T. Binz, et al. (2012). BPMN4TOSCA: A domain-specific language to model management plans for composite applications, Springer.

La, H. J. and S. D. Kim (2009). A systematic process for developing high quality saas cloud services. Cloud Computing, Springer**:** 278-289.

Laszewski, T. and P. Nauduri (2011). Migrating to the Cloud: Oracle Client/Server Modernization, Elsevier.

Leymann, F. (2011). "Cloud computing." it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik **53**(4): 163-164.

Lindland, O. I., G. Sindre, et al. (1994). "Understanding quality in conceptual modeling." Software, IEEE **11**(2): 42-49.

Liu, F., J. Tong, et al. (2011). "NIST cloud computing reference architecture." NIST special publication **500**: 292.

Loutas, N., E. Kamateri, et al. (2011). Cloud computing interoperability: the state of play. Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, IEEE.

Martens, B. and F. Teuteberg (2011). "Risk and compliance management for cloud computing services: Designing a reference model."

Mendling, J. and M. Nüttgens (2006). "XML interchange formats for business process management." Information Systems and E-Business Management **4**(3): 217-220.

Menychtas, A., C. Santzaridou, et al. (2013). ARTIST Methodology and Framework: A novel approach for the migration of legacy software on the Cloud. Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on, IEEE.

MLSAC (2016). https://www.dropbox.com/sh/u1im5321t8hdjbq/AAA5p6fWGfbedTPresVfCfNBa?dl=0, Prentice Hall PTR.

Mohagheghi, P. (2011). Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project. Services (SERVICES), 2011 IEEE World Congress on.

Mohagheghi, P., A. Berre, et al. (2010). REMICS- REuse and Migration of Legacy Applications to Interoperable Cloud Services. Towards a Service-Based Internet. E. Nitto and R. Yahyapour, Springer Berlin Heidelberg. **6481:** 195-196.

Moody, D. L. (1998). Metrics for evaluating the quality of entity relationship models. Conceptual Modeling–ER'98, Springer**:** 211-225.
Moody, D. L. (2005). "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions." Data & Knowledge Engineering **55**(3): 243-276.
Nowak, A., U. Breitenbücher, et al. (2014). Automating Green Patterns to Compensate CO2 Emissions of Cloud-based Business Processes. ADVCOMP 2014, The Eighth International Conference on Advanced Engineering Computing and Applications in Sciences.
Nunez, D., C. Fernandez-Gago, et al. (2013). A metamodel for measuring accountability attributes in the cloud. Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on, IEEE.
Ortiz Jr, S. (2011). "The problem with cloud-computing standardization." Computer **44**(7): 13-16.
Othman, S. H. and G. Beydoun (2013). "Model-driven disaster management." Information & Management **50**(5): 218-228.
Othman, S. H., G. Beydoun, et al. (2014). "Development and validation of a Disaster Management Metamodel (DMM)." Information Processing & Management **50**(2): 235-271.
Paige, R. F., P. J. Brooke, et al. (2007). "Metamodel-based model conformance and multiview consistency checking." ACM Transactions on Software Engineering and Methodology (TOSEM) **16**(3): 11.
Peffers, K., T. Tuunanen, et al. (2008). "A design science research methodology for information systems research." Journal of management information systems **24**(3): 45-77.
Procaccianti, G., P. Lago, et al. (2014). "Green Architectural Tactics for the Cloud."
Quang Hieu, V. and R. Asal (2012). Legacy Application Migration to the Cloud: Practicability and Methodology. Services (SERVICES), 2012 IEEE Eighth World Congress on.
Rabetski, P. (2012). "Migration of an on-premise application to the cloud."
Ralyté, J., R. Deneckère, et al. (2003). Towards a Generic Model for Situational Method Engineering. Advanced Information Systems Engineering. J. Eder and M. Missikoff, Springer Berlin Heidelberg. **2681:** 95-110.
Ranabahu, A. H., E. M. Maximilien, et al. (2011). A domain specific language for enterprise grade cloud-mobile hybrid applications. Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11, ACM.
Ried S, K. H. (2011). "Sizing the cloud—A BT futures report." Forrester Research, Inc., Cambridge, MA.
Rossi, M. and S. Brinkkemper (1996). "Complexity metrics for systems development methods and techniques." Information Systems **21**(2): 209-227.
Rossi, M., B. Ramesh, et al. (2004). "Managing evolutionary method engineering by method rationale." Journal of the Association for Information Systems **5**(9): 356-391.
S. Strauch, V. A., D. Karastoyanova, F. Leymann (2014). "Migrating Enterprise Applications to the Cloud: Methodology and Evaluation." International Journal of Big Data Intelligence.
Sargent, R. G. (2005). Verification and validation of simulation models. Proceedings of the 37th conference on Winter simulation, Winter Simulation Conference.
Siau, K. and M. Rossi (1998). Evaluation of information modeling methods-a review. System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on, IEEE.
Sledziewski, K., B. Bordbar, et al. (2010). A DSL-based approach to software development and deployment on cloud. Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, IEEE.
Stamper, R. (1996). "Signs, norms, and information systems." Signs at work. Walter de Gruyter, Berlin: 349-397.
Tran, V., J. Keung, et al. (2011). Application migration to cloud: a taxonomy of critical factors. Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, ACM.
UML, O. (2004). "2.0 Superstructure Specification." OMG, Needham.

Varia, J. (2010). "Migrating your existing applications to the aws cloud: A Phase-driven Approach to Cloud Migration."

Weimer, M., T. Condie, et al. (2011). Machine learning in ScalOps, a higher order cloud computing language. NIPS 2011 Workshop on parallel and large-scale machine learning (BigLearn).

Wettinger, J., M. Behrendt, et al. (2013). Integrating Configuration Management with Model-driven Cloud Management based on TOSCA. CLOSER.

Yang, H. and M. Tate (2012). "A descriptive literature review and classification of cloud computing research." Communications of the Association for Information systems **31**(2): 35-60.

Z.Mahmood (2013). Cloud Computing Methods and Practical Approaches, Springer-Verlag London 64.

Zech, P., M. Felderer, et al. (2012). Cloud risk analysis by textual models. Proceedings of the 1st International Workshop on Model-Driven Engineering for High Performance and CLoud computing. Innsbruck, Austria, ACM**:** 1-6.

Zhang, J., J.-Y. Chung, et al. (2004). Migration to web services oriented architecture: a case study. Proceedings of the 2004 ACM symposium on Applied computing, ACM.

Zhang, L.-J. and Q. Zhou (2009). CCOA: Cloud computing open architecture. Web Services, 2009. ICWS 2009. IEEE International Conference on, Ieee.

Zimmermann, A., M. Pretz, et al. (2013). Towards Service-Oriented Enterprise Architectures for Big Data Applications in the Cloud. Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International, IEEE.

Zimmermann, O., C. Miksovic, et al. (2012). "Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services." Journal of Systems and Software **85**(9): 2014-2033.