Towards a Model-Driven Dynamic Architecture Reconfiguration Process for Cloud Services Integration

Miguel Zuñiga-Prieto, Javier Gonzalez-Huerta, Silvia Abrahao, Emilio Insfran

ISSI Research Group, Department of Information Systems and Computation
Universitàt Politècnica de Valencia
Camino de Vera, s/n, 46022, Valencia, Spain
{mzuniga, jagonzalez, sabrahao, einsfran}@dsic.upv.es

Abstract. Cloud computing is a paradigm that is transforming the computing industry and is receiving more attention from the research community. The incremental deployment of cloud services is particularly important in agile development of cloud services, where successive cloud service increments must be integrated into existing cloud service architectures. This requires dynamic reconfiguration of software architectures, especially in cloud environments where services cannot be stopped in order to apply reconfiguration changes. This paper presents a model-driven dynamic architecture reconfiguration process to support the integration of cloud services. Models are used to represent high-level architecture reconfiguration operations as well as adaptation patterns. Adaptation patterns allow us to describe reconfiguration operations independently of a specific cloud platform technology. On the other hand, model transformations are used: i) to support compatibility checking of increments; ii) to generate software adaptors that solve incompatibilities between architectures; and iii) to generate reconfiguration plans specific of cloud provider, that include reconfiguration actions to be applied on cloud service instances at runtime. The proposed process is illustrated with a dealer network system development example, where cloud services are deployed in an incremental way.

Keywords: Model Driven Development, Model Transformations, Cloud Computing, Dynamic Reconfiguration, Model Based Evolution

1 Introduction

Cloud computing is a software engineering paradigm that has the potential of change large part of the IT industry; becoming a research topic with innovative proposals to design, develop and deploy cloud-based systems [1]. Cloud applications are delivered as services over the Internet. Among distinguishable characteristics of cloud computing parading are measured service and rapid elasticity and scalability [2]. The former allows billing based on real usage of resources. The later allows acquiring more resources during a peak of demand and releasing them once they are no longer required. In addition, services can be redeployed on different provider-specific platforms depending on Quality of Service (QoS), Service Level Agreement (*SLA*) or other business criterion.

Service-oriented architecture approach is a way of designing, developing and deploying loosely coupled distributed applications using coarse-grained services. Developing service-oriented applications (such as cloud services) facilitates reconfiguration of software architectures at runtime, what is known as dynamic architecture reconfiguration. Organizations that adopt this approach will be able to i) manage business evolution and/or upgraded services can be introduced with minimum impact on existing systems, and ii) implement loosely-coupled integration approaches [3]. As stated before, cloud services could be deployed in different provider-specific platforms; which often leads to tight coupling of developed cloud services to a specific cloud provider technology. In order to avoid the dependence on cloud providers, the cloud service architectural design must facilitate the use of different environments for execution [4].

Model-Driven Development (MDD) is an approach for developing software systems that promotes a new form of building systems based on the construction and maintenance of models at different levels of abstraction to drive the development process. In this approach, a software system is developed by refining models and it is implemented through model to text transformations.

Software adaptation patterns represent generic and repeatable solutions to manage change in recurring architectural adaptation problems, and prescribe the steps needed to dynamically adapt a software system at runtime from one configuration to another [5]. The use of adaptation patterns is a trend to support reuse in evolution for dynamic adaptive software architecture [6]. Adaptation of software architectures is not only supported by change management proposals, but also by proposals for solving the problems that arise when the interacting entities do not match properly. Software adaptation promotes generation of software adaptors to bridge incompatibilities among services (e.g., different names of methods and services, different message ordering, etc.) in an nonintrusive way [7,8, 9]. Generation techniques for software adaptors are beginning to be used in cloud environments [10].

Cloud applications integrate and compose different cloud services. The cloud services to be integrated may come from the delivering of a software increment in an incremental development approach, or just may be product of maintenance/evolution phases. The integration/update of increments may trigger the dynamic reconfiguration of the existing cloud service architecture. Dynamic reconfiguration creates and destroys architectural elements instances at runtime; being particularly important for cloud services be able to manage instances in different cloud platforms and continue working while reconfiguration takes place. However almost no or little attention has been paid on supporting this reconfiguration at runtime, and only in recent years software engineering research started focusing on these issues [11]. In addition, as far as we know, the incremental and dynamic deployment of cloud services into existing services in the cloud has not been studied yet.

In this paper, we introduce a process to support the dynamic reconfiguration of cloud service architectures due to the integration of software increments. This process will allow software developers to specify how the integration of the architecture of a software increment affects the current cloud service architecture. Additionally, after applying model-driven techniques, software developers will obtain the software artifacts needed to dynamically reconfigure the current cloud service architecture. We define the

architecture of a software increment as a portion of an architecture that corresponds to the architectural description of the increment whose integration into current architecture triggers the current architecture reconfiguration.

The remainder of the paper is structured as follows. Section II discusses related work on proposals to support the dynamic architecture reconfiguration. Section III presents our dynamic architectural reconfiguration approach. Finally, Section IV presents our conclusions and future work.

2 Related work

Software evolution based on reconfiguration of software architectures is an active area of research; however, there are gaps that still need to be covered. Some of these gaps where identified in a systematic literature review performed by Jamshidi et al. [6]. The authors took into account the stage of the software lifecycle where evolution mechanisms were active; findings showed lack of support during the integration and provisioning stage, but also during deployment stage. In our work, we give support to the dynamic reconfiguration of software architectures at the deployment stage of the software life cycle.

In this section, we analyze how researchers and practitioners address the dynamic reconfiguration of software architectures to support the development of cloud/service applications. The most relevant works [11, 12, 13] we have found are analyzed below.

Baresi et al.[12] propose a methodology for deriving service-oriented architectures from high-level business-oriented architecture descriptions. They use formal representations to describe both application specific types as well as runtime configurations of concrete instances. They also, use graph transformations rules and define refinement relation from a generic style of component-based systems to the SOA style.

MOdel-based SElf-adaptation of SOA systems (MOSES) [13], proposes a methodology aimed at driving the self-adaptation of a SOA system to fulfill non-functional QoS requirements. This framework uses linear programing to formulate the identification of the most suitable adaptation according to the detected changes in the environment.

Self-architecting Software Systems (SASSY) [14] uses the application requirements captured by domain experts to derive automatically a base software architecture. Then, SASSY derives an optimized architecture from the base architecture by selecting the most suitable service providers and by applying QoS architectural patterns. In addition, for each QoS architectural pattern, they apply adaptation patterns that specify how the system self-adapts to incorporate the pattern into the configuration. Unlike previous cited approaches, SASSY deploys the coordination logic.

All the works described above i) take into account structural and behavioral aspects for reconfiguration; ii) use SLA or QoS negotiation to discover and select the most suitable service implementation (instance); and iii) apply dynamic binding for reconfiguration. This means that reconfiguration improves non-functional qualities through perfective changes. However, adaptive changes (e.g., software increments due to new functionalities) that require architecture reconfiguration are not taken into account.

They abstract models of business requirements or derive high level architectures ; however, they do not take into account the importance of architectural aspects in agile/incremental development processes [15]. Despite the fact that cited approaches propose consistency or compatibility checking task, they do not provide solutions to support the deployment in different cloud platforms.

In summary, as far as we know, there is a lack of support to incremental and dynamic deployment of cloud services into existing cloud service architecture. Our approach allows incremental reconfiguration of software architectures, and promotes compatibility between the architecture of software increments with the existing cloud architecture, according to cloud specific deployment platform.

3 A Process for Dynamic Architecture Reconfiguration of Cloud Services

In this section, we introduce our motivation example and continue with the reconfiguration process description.

3.1 Motivating Example

The proposed motivation example is based on "*Acme Manufacturing*" dealer network system scenario [16]. *Acme* is a manufacturer company, which wants to improve service to its dealers and partners. With this purpose in mind, Acme considers building and deploying cloud services in an incremental way. The first increment aims to do a better job of fulfilling dealers' orders and provides cloud services for dealers to place and manage their orders. This allows a direct interaction between customer's I.T. systems and Acme's systems. Acme also needs to improve its shipping process to increase delivery speed, and thus, in a second increment provides its local transport partner with cloud services. The transport partner uses cloud services to retrieve orders that need to be shipped as well as to inform dealers about shipping status. This second increment also updates the cloud services deployed in the first increment, providing dealers with information about last bought of items included in the order. Finally, after the third increment Acme needs to be able to manage international deliveries. However, since the international partner has its own custom systems based on exposed web services *Acme* uses the partner's web service to make shipping requests.

3.2 Reconfiguration Process

The model-driven Dynamic Incremental Architectural Reconfiguration (*DIARy*) process has been defined using model-driven and adaptation techniques. This process aims to support software developers during the deployment phase, on activities related to integration of software increments into existing services in the cloud. We support the integration process from an architectural point of view. *DIARy* proposes activities to support the management of dynamic reconfiguration of existing cloud services architectures, produced due to the integration of architectural elements. The main activities

of *DIARy* process are: i) *Specify Increments*; ii) *Check Increment Compatibility*; and iii) *Reconfigure Architecture*. Fig. 1 shows the activities of the *DIARy* process.

Specify Increments. *DIARy* may be incorporated into existing development processes and this activity serves as a glue that allows its incorporation. *Software Architects* perform this activity not only to specify the architecture that corresponds to the architectural description of the increment to be deployed, but also the impact that the integration of the increment has on current architecture. The latter is specified describing how the elements of the *architecture of the software increment* collaborate to reconfigure the current architecture in order to provide the required functionality. This activity generates as output the Increment's Architecture Model; uses as inputs artifacts the Current Architecture, *Design Artifacts*, and *SLA*s. Additionally, uses Increment Description Guidelines as guide and an architecture description language to describe both the current architecture as the *architecture of the software increment*. Each of these artifacts is explained below:

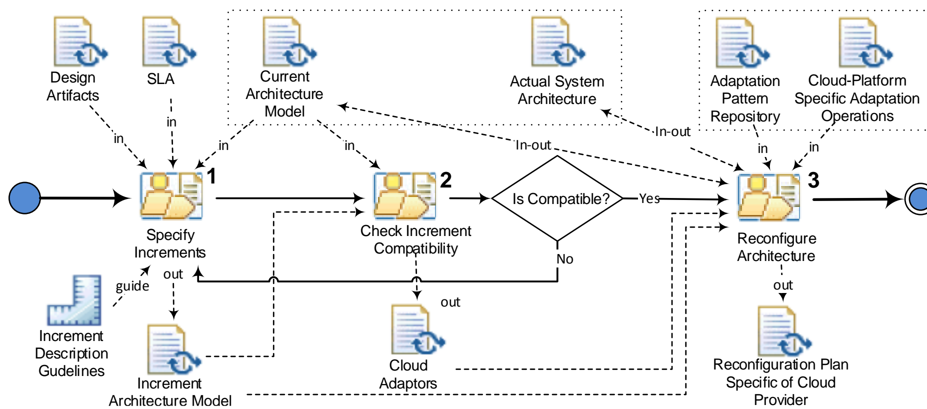

Fig. 1. Overview of the *DIARy* approach

1. *Current Architecture Model (CurAM):* This model allows representing the current architecture (i.e., before increment deployment) using design artifacts. *CurAM* includes information about services, connectors, configuration as well as cloud software architecture related information. *CurAM* evolves after each increment integration; however, in this activity it is used only as input, helping *Software Architects* to identify elements of the current architecture to be affected by the integration.
2. *Design Artifacts:* This input artifact represents design artifacts generated during the development process. Depending on the development process to which *DIARy* is applied, this artifact could be i) the original system architecture designed during the development process; ii) any form of architecture description that describes the increment; iii) Architectural backlogs generated during an agile development process. This artifact helps *Software Architects* to identify the elements of the current architecture that will be affected by the integration.

3. *Service Level Agreements (SLA):* This artifact contains the conditions and parameters that compromise the service provider to meet certain levels of quality. *Software Architects* use it to take design decision during specification.
4. *Increment's Architecture Model (IAM)*: Software Architects participate in generating this output artifact**.** *IAM* allows representing the *architecture of the software increment* and includes information about services, connectors, and configuration. Furthermore, *IAM* allows represent how the elements of the *architecture of the software increment* collaborate to reconfigure the current architecture. To do this, *IAM* includes references to *CurAM* elements. *Software Architects* use references to point out the elements of current architecture affected by the increment (elements added, updated or deleted) as well as the elements used as Integration Points (*IP*). We call *IP* to the interfaces of the current architecture elements that interact with interfaces of the elements of the *architecture of a software increment* in order to allow interaction and provide the required functionality. Finally, in order to support reconfiguration on cloud environments, *IAM* includes information related aspects of Cloud Software Architectures [17]. For instance, the *IAM* associated to the second increment of motivating example (see section 3.1) will include:
   (a) Information about Shipping Request Service, Ship Status Service and connections that need to be added.
   (b) References to the interfaces of Place Order service and its interaction protocol (i.e., both elements need to be updated in order to satisfy the new requirement that establish that: the Place Order service must provide information about last bought of items included in order).
   (c) References to the service interfaces required and provided by/to the current architecture that will serve as *IP*.
   (d) Information related to cloud software architecture such as interaction pattern between dealer and transport partner (e.g., publish/subscribe connector, request/response connector).
5. *Increment Description Guidelines:* Help *Software Architects*: i) to identify impact of increment integration on current architecture and; ii) to take design decisions. These guidelines give support about how to specify increments using *IAM* and *CurAM*. In addition, we have begun to work in an *Increment Description Language* (*IDL*). This language will allow *Software Architects* to use high-level architecture reconfiguration operations to specify impact of the integration on current cloud service architecture. Service Oriented Architecture Modeling Language (SoaML) [18] leverages Model Driven Architecture (MDA) and provides a UML profile and meta-model for the specification of services. However, SoaML does not allow to represent how the architecture of a software increment affects the existing cloud architecture nor to specify information related to cloud software architectures. *IAM* and *CurAM* are part of this *IDL* and their meta-models will extend the SoaML meta-model. *IAM* and *CurAM* artifacts are input for the next activity, which is described below.

Check Increment Compatibility. This activity helps to verify whether the *architecture of a software increment* can be integrated into the current architecture. Its main objective is to reduce the risk of incompatibilities between service interfaces that could

avoid integration (e.g., different names of methods and services, different message ordering, etc.). Despite the fact that in previous activity *Software Architects* specified the impact of the increment integration on current architecture, in practice, we cannot expect that any given software component perfectly matches the needs of a system nor that the components being assembled perfectly fit one another [7]. The same may happen during the integration of the increment, where incompatibilities may exist between *IAM* and *CurAM* elements.

We will apply software adaptation techniques to correct incompatibilities, generating software adaptors when needed. We have chosen to follow Cámara et al. approach [19] because i) it is a model driven approach; ii) it gives support the automatic creation of adaptors from abstract specifications; iii) and, it provides tools that fully support the adaptation approach from start to finish (including compatibility checking). To follow this approach we need to provide service interfaces described by signatures (operation names and types) and interaction protocol. The former must be described as a WSDL representation and the latter as an Abstract BPEL (ABPEL) representation. *Integration Designers* will specify model transformations to obtain these representations from the increment's architecture (*IAM*) as well as from the current architecture (*CurAM*).

This activity results in generation of software adaptors to be used in a specific cloud platform (*CloudAdaptors*) using *CurAM* and *IAM* as input. *CloudAdaptors* allow correcting incompatibilities between services interaction protocols (i.e., incompatibilities among *IP* operations). If discrepancies exist, *Software Architects* apply model-to-text (M2T) transformations to generate skeletons of *CloudAdaptors*. Then, *Software Developers* complete *CloudAdaptors* skeletons, implementing code to solve discrepancies according to deployment platform. Depending on cloud platform, *CloudAdaptors* may be scripts, configuration files, packages, services, or any cloud platform specific artifact. For instance, regarding to our motivating example (see Section 3.1), in order to allow interaction with web services provided by international transport partner, the integration of the third increment will require i) compatibility verification of interfaces requested by manufacturer and interfaces provided by international transport partner. The first interfaces are already deployed and belong to *CurAM*; whereas the latter, that are going to be deployed, are described in *IAM*; ii) generation of *CloudAdaptors*. Assuming that the deployment platform is Windows Azure, the generated *CloudAdaptors* will be a cloud service Worker Role.

Reconfigure Architecture. This last activity supports the execution of the integration operations, resulting in incorporation of the *architecture of the software increment* into current architecture and the corresponding dynamic architecture reconfiguration (see Fig. 2). This activity is composed of the following main steps:

Select Adaptation Pattern. In this step, *Software Architects* participate in the selection of the adaptation patterns best suited to integrate the *architecture of the software increment* into the current architecture. This step results in the generation of a List of Patterns Model, using *CurAM* and *IAM* information to select patterns from Adaptation Pattern Repository Model. The Output artifact and input *AdaPRepM* are explained below:

1. *Adaptation Pattern Repository Model* (*AdaPRepM*): *Integration Designers* use this to represent prescriptions at a high level of abstraction of steps required to integrate architectural elements into current architecture. *Integration Designers* define adaptation patters for possible integration scenarios. We consider scenarios where the elements of the *architecture of a software increment*: i) do not need interconnection with any current architecture element; ii) require establish interconnection with current architecture elements without updating them; and iii) require establish interconnections and update current architecture elements. Adaptation patterns is a research field by itself, in our work we will extend current proposals to define the *AdaPRepM* meta-model. To be specific, we will extend the Meta-model for Adaptation Pattern Composition proposed by Ahmad et al. [20].

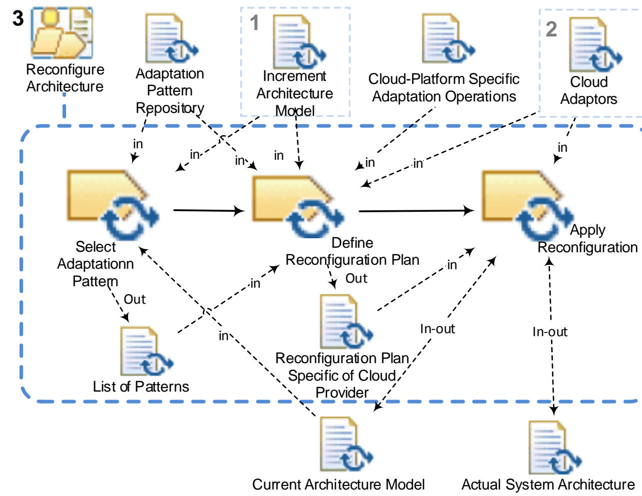

Fig. 2. Reconfigure Architecture Activity

2. *List of Patterns Model* (*LisPatM*): This output provides a list with the most suited adaptation patterns that must be applied to integrate the *elements of the architecture of the software increment* into current architecture.

Define Reconfiguration Plan. This activity aims to generate a plan with the sequence of reconfiguration operations needed to integrate the elements of the *architecture of the software increment* into the current architecture. For doing this, a two-step models transformation strategy must be applied. On the first step, *Integration Designers* specify M2M transformations that generate a Reconfiguration Plan Independent of Cloud Provider technology. This plan includes high-level reconfiguration actions needed to change cloud service architectures. In the second step, *Integration Designers* specify M2T transformations to operationalize reconfiguration actions into Reconfiguration Plans Specific of Cloud Provider. *Software Architects* execute these transformations and *Software Developers* complete the generated plans if required. This activity has as inputs *IAM*, *AdaPRepM*, *LisPatM* and *Platform Specific Adaptation Operations Model*.

1. *Cloud-Platform Specific Adaptation Operations Model*: This model represents at a high level of abstraction cloud artifacts and reconfiguration operations independently of a specific cloud platform technology. This model and *LisPatM* are used to generate a Reconfiguration Plan Independent of Cloud Provider.
2. *Reconfiguration Plan Specific of Cloud Provider*: This artifact is specific of a cloud provider technology. This artifact includes sequence of commands that create, update, or destroy architectural elements instances and their links at runtime. Examples are scripts, packages, configuration files and so on.

Apply reconfiguration. In the last step, the *Cloud Specialist*, expert in deployment, integrates the increment into the current architecture by deploying *CloudAdaptors* and by using dedicated services to apply the *Reconfiguration Plan Specific of Cloud Provider* artifacts in the corresponding cloud platform. Integration dynamically reconfigures instances of the running *Actual System Architecture*.

4 Conclusions and Future Work

We introduced the *DIARy* process to support the dynamic software architecture reconfiguration triggered by the deployment of new cloud services. *DIARy* uses model-driven and adaptation techniques to allow integration of cloud services into current architecture at runtime. We believe this process provides a solution to cover the lack of support to incremental and dynamic deployment of cloud services into existing cloud service architecture. *DIARy* shows the steps that software developers must follow to specify how the *architecture of a software increment* will affect the existing cloud architecture. In addition, model transformations are used for: i) promoting the compatibility between the architecture of the increment with the existing cloud architecture; and ii) generating cloud-platform specific reconfiguration plans that apply adaptation patterns to reconfigure existing cloud architecture. Activities and artifacts included in *DIARy* were described, and a motivation example was used to illustrate some related aspects.

At this moment, we experimented with several small examples to test the viability of the approach. As further work, we plan to empirically validate DIARy through controlled experiments and case studies with medium-sized real-world projects. We are also working on: i) the definition of an Increment Description Language to specify increment's architectures and their impact on actual system architecture; ii) the definition of a reference architecture to support the reconfiguration process, and iii) the implementation of different model transformations to automate the DIARy process.

Acknowledgments. This research was supported by the Value@Cloud project (MICINN TIN2013-46300-R); the Scholarship Program Senescyt, Ecuador; the Faculty of Engineering, University of Cuenca, Ecuador; and the ValI+D program (ACIF/2011/235), Generalitat Valenciana.

5 References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A View of Cloud Computing. Commun. ACM. 53, 50–58 (2010).
2. Motta, G., Sfondrini, N., Sacco, D.: Cloud Computing: An Architectural and Technological Overview. Int. Joint Conf. on Service Sciences. pp. 23–27. IEEE, Shanghai (2012).
3. Bastida, L., Berreteaga, A., Cañadas, I.: Adopting Service Oriented Architectures Made Simple. Springer, London (2008).
4. Fehling, C., Leymann, F., Retter, R.: An Architectural Pattern Language of Cloud-Based Applications. 18th Conf. on Pattern Languages of Programs. pp. 1–11. ACM Press, New York (2011).
5. Gomaa, H., Hashimoto, K., Kim, M., Malek, S., Menascé, D.: Software Adaptation Patterns for Service-Oriented Architectures. ACM Symposium on Applied Computing. pp. 462–469. ACM, New York (2010).
6. Jamshidi, P., Ghafari, M., Ahmad, A., Pahl, C.: A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research. 17th European Conference on Software Maintenance and Reengineering. pp. 305–314. IEEE, Genova (2013).
7. Canal, C., Poizat, P., Salaun, G.: Model-Based Adaptation of Behavioral Mismatching Components. Softw. Eng. IEEE Trans. 34, 546–563 (2008).
8. Yellin, D.M., Strom, R.E.: Protocol Specifications and Component Adaptors. ACM Trans. Program. Lang. Syst. 19, 292–333 (1997).
9. Becker, S., Brogi, A., Gorton, I., Overhage, S., Romanovsky, A., Tivoli, M.: Towards an Engineering Approach to Component Adaptation. Springer Berlin Heidelberg (2006).
10. Miranda, J., Guillen, J., Murillo, J.M., Canal, C.: Assisting Cloud Service Migration Using Software Adaptation Techniques. 6th Int. Conf. on Cloud Computing. pp. 573–580 (2013).
11. Baresi, L., Ghezzi, C.: The Disappearing Boundary Between Development-time and Run-time. FSE/SDP Workshop on Future of software Engineering Research. pp. 17–21 (2010).
12. Baresi, L., Heckel, R., Thöne, S., Varr´o, D´.: Style-Based Modeling and Refinement of Service-Oriented Architectures. Softw. Syst. Model. 5, 187–207 (2006).
13. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: QoS-Driven Runtime Adaptation of Service Oriented Architectures. Proc. 7th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. 131–140 (2009).
14. Menascé, D.A., Gomaa, H., Malek, S., Sousa, J.P.: SASSY: A Framework for Self-Architecting Service-Oriented Systems. Software, IEEE. 28, 78–85 (2011).
15. Babar, M.A., Brown, A.W., Mistrik, I.: Making Software Architecture and Agile Approaches Work Together: Foundations and Approaches. Agile Software Architecture: Aligning Agile Processes and Software Architectures. pp. 1–22. Morgan Kaufmann (2013).
16. Casanave, C.: Enterprise Service Oriented Architecture Using the OMG SoaML Standard. Model Driven Solut. Inc., White Pap. 1–21 (2009).
17. Hamdaqa, M., Livogiannis, T., Tahvildari, L.: A Reference Model for Developing Cloud Applications. CLOSER. pp. 98–103. Citeseer (2011).
18. Object Management Group: Service Oriented Architecture Modeling Language (SoaML), http://www.omg.org/spec/SoaML/.
19. Cámara, J., Martin, J.A., Salaun, G., Cubo, J., Ouederni, M., Canal, C., Pimentel, E.: Itaca: An Integrated Toolbox for the Automatic Composition and Adaptation of Web Services. 31st Int. Conf. on Software Engineering. pp. 627 – 630. IEEE, Vancouver, BC (2009).
20. Ahmad, A., Babar, M.A.: Towards a Pattern Language for Self-Adaptation of Cloud-Based Architectures. Proc. WICSA Companion Volume. pp. 1–6. ACM Press, New York (2014).