Architecture-Driven Engineering of Cloud-Based Applications

Everton Cavalcante

DIMAp – Department of Informatics and Applied Mathematics
UFRN – Federal University of Rio Grande do Norte
Natal, Brazil
evertonrsc@ppgsc.ufrn.br

Abstract. In this paper we give an overview of a Ph.D. work that involves the specification, implementation, and evaluation of an architecture-driven environment for cloud applications that consists of: (i) a *cloud-based architecture description language* (ADL) endowed with constructs to represent cloud services, contracts in terms of quality metadata and pricing models, and dynamic reconfiguration statements, and; (ii) a *cloud middleware* that enables the use of underlying cloud platforms and also offers services for monitoring cloud services, composing, and adapting cloud applications.

Keywords: Cloud applications, Architecture description languages, Cloud-ADL, Dynamic reconfiguration, Specification-to-deployment environment

1 Introduction

Cloud Computing is a new computing paradigm that enables the ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage facilities, applications and services) typically referred as a *cloud* [1]. These resources can be rapidly provisioned and released with minimal management effort or interaction with the service provider and are offered in a pay-per-use model. Although Cloud Computing provides benefits absent in current technologies, such as the pay-per-use model and the elasticity of the applications, the development of this paradigm is still in emergence and there are several challenges to be addressed in the Cloud Computing context.

One of the challenges brought by the Cloud Computing paradigm regarding Software Engineering is related to the methods, tools, and techniques to support developers to design, build, and deploy complex software systems that make use of the cloud technology. In fact, building cloud-based applications is a challenging task as they are significantly more complex due to the intrinsic complexity of using third-party cloud providers, thus encompassing issues such as the decision of which underlying cloud platforms to use and the need of tracking pricing policies of the provided services. Moreover, the particular nature of the cloud applications create specific requirements that also demand changes in terms of the development of such applications, thus encompassing methodologies and techniques for requirements elicitation, architecture, implementation, deployment, testing, and evolution of software [2, 3].

In terms of software architecture, architectures of service-oriented applications (such as cloud applications) differ from the architecture of traditional applications mainly regarding their basic architectural elements, the *services*, and their dynamic behavior, thus dealing with issues related to heterogeneity interoperability, and changing requirements [4, 5]. As most of the works addressing cloud-related challenges focus on the underlying infrastructure or on discussions about the cloud services provided by third-party providers, there is an important gap in terms of *architectural support* in order to enable the use of the cloud technology in a systematic way. For instance, it is reasonable to think that software architectures need to be described in a different way if they are deployed on the cloud, so that it is necessary to provide an architectural support to their project and means to model them and capture important elements regarding the Cloud Computing paradigm, e.g. the logical separation between the cloud platforms (service providers) and the applications (service consumers), the agreed contract between these players, and the modeling of the pricing model adopted by each service. Furthermore, due to the dynamics of the Cloud Computing environments, *dynamic reconfiguration* must be autonomously performed according to the changes in the environment in order to meet the application requirements, such as replacing a cloud service that is being used by another one in case of degradation of quality parameters or when new services arise in the environment.

In order to mitigate some of these issues, this Ph.D. work aims to propose, specify, implement, and evaluate an architecture-driven environment for developing cloud-based applications. In a higher abstraction level, such environment provides means to describe architectures of cloud applications by using a customized *architecture description language* (ADL) [6, 7] that addresses important elements related to the Cloud Computing context. Such environment is also intended to provide compositional mechanisms to design and dynamically reconfigure applications driven by their respective architectural descriptions combined with a cloud middleware. In this perspective, the infrastructure composed by the ADL and the middleware platform enables to manipulate (at both design time and runtime) the underlying services that are being used by the applications.

In summary, the novelty of this Ph.D. work consists of: (i) the proposal of a new ADL for describing cloud-based applications (and then specificities related to the Cloud Computing context) and specifying dynamic reconfiguration actions; (ii) the mapping of the constructors provided by such ADL to elements managed by a cloud-based middleware platform, which is able to monitor the execution of the application as well as the different cloud services that it uses, and; (iii) the support for dynamic reconfiguration of cloud applications driven by architecture and related constraints, so that the cloud-based middleware manages the dynamic reconfiguration actions specified in the architectural description. Therefore, the proposed environment is intended to provide a complete support not only for specifying architectures of applications, but also for their execution and autonomous adaptation under dynamic conditions. This is an essential issue for cloud applications that may use services from one or more providers according to QoS (Quality of Service) constraints and resources provisioning, and then the application can have a greater control over the quality of the services and their dynamic variability by exploring this capability.

This position paper is structured as follows. Section 2 presents details of the proposed environment. Section 3 discusses some related works. Finally, Section 4 contains final remarks and directions to ongoing and future works.

2 Architecting cloud-based applications

The environment proposed in this work is composed of three main elements that enable software architects to design cloud-based applications in terms of their description and adaptation in dynamic scenarios, as detailed in the following subsections. At the *architectural level*, architectures of cloud applications are described by using an ADL, which also provides constructors to specify dynamic reconfiguration actions that will take place at the *runtime level*. These levels are integrated through a *cloud-based middleware platform*, which is able to monitor the execution of the application and manage the dynamic reconfiguration actions specified at the architectural level. Fig. 1 gives an overview of the proposed environment for architecting Cloud Computing applications.

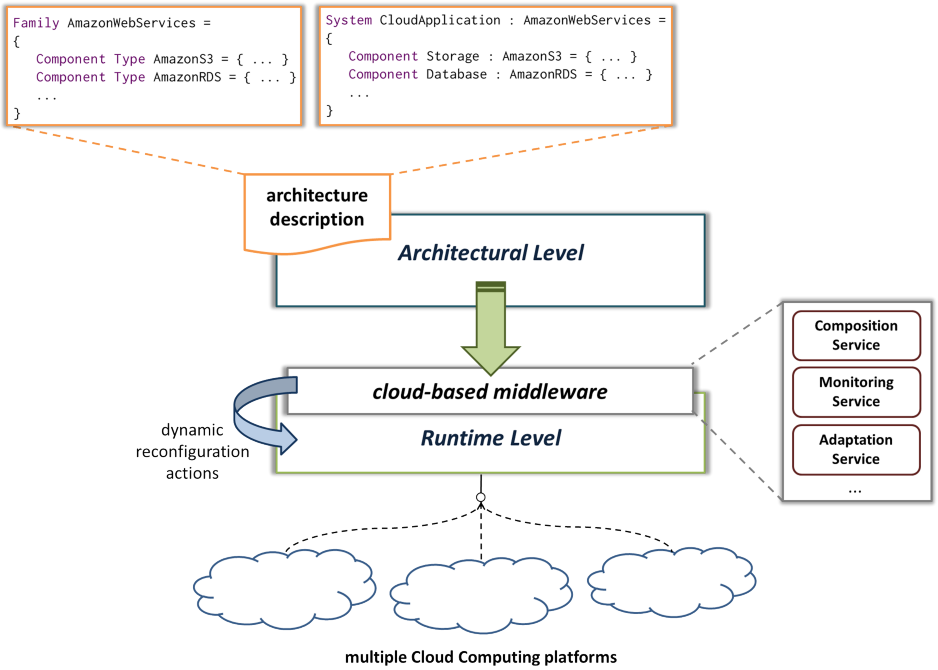

Fig. 1. Overview of the proposed architecture-driven cloud environment.

2.1 Architectural formalism

ADLs [6, 7] are a well-accepted approach to formally represent and analyze architectural designs and usually provide both a conceptual framework as a concrete syntactic

notation to characterize software architectures in terms of: (i) *components*, which are functional elements of the system; (ii) *connectors*, which are responsible for describing the communication mechanisms between the components, and; (iii) *configurations*, which describe the topology of the system through the relationship between components and connectors. Although there is a myriad of ADLs in the literature, the architectural features of cloud applications goes beyond what is currently supported by the existing ADLs, so that an ADL could provide a systematic and disciplined way of developing and managing cloud-based applications. Moreover, it is missing models that capture specific and important aspects always present in the Cloud Computing context, such as: (i) the logical separation between the cloud platforms and the applications that use the provided services; (ii) the agreed contract between these players, and; (iii) the QoS model for the services. As briefly discussed in Section 3, to the best of our knowledge, there are no proposals in the literature for modeling cloud-based applications, and despite of they are service-oriented, there are very few proposals for describing service-oriented software architectures [4, 5].

In this perspective, our recent work [8] introduced Cloud-ADL, a seamless extension of the ACME [9] general-purpose ADL for modeling Cloud Computing applications. The philosophy of Cloud-ADL is to take advantage of ACME elements to describe architectures of a cloud application, thus relying on existing abstractions and avoiding the addition of many new abstractions. Cloud-ADL enables to describe the architecture of the applications and: (i) the *services* provided by cloud platforms and that are used by the applications, thus making a clear and modular separation between the definition of the cloud services and the specification of the applications that use them; (ii) the agreed *contracts* between service providers and clients, in terms of quality metadata, pricing models, and constraints regarding the services; (iii) application-level *constraints*, and; (iv) *dynamic reconfiguration actions* according to QoS and resource provisioning constraints.

The complete specification of a cloud application in Cloud-ADL encompasses:

1) Definition of cloud services provided by Cloud Computing platforms and essential characteristics related to this context. In this step, the ACME *Family* element is customized to the cloud scenario for representing the set of services provided by a cloud platform, so that a new *Family* abstraction must be created for each cloud platform. In the *Family* element, each service provided by a cloud platform is represented by a *Component Type* element, as in the specification of component types in ACME. Similarly to ADLs for service-oriented architectures [4, 5], in the architectural description of the application, the component types that represent provided cloud services can be instantiated in order to indicate which services are being used by the application. As in ACME, the components (services) can be annotated with *Property* elements for expressing additional information about these components.

Architectural contracts [10] are used in some works in the literature [12, 13] to express static and/or dynamic non-functional features and describe, at design time, the resources that will be used by the application and acceptable variations regarding the availability of these resources, at runtime. Grounding on this idea, Cloud-ADL comes with a novel mandatory first-class primitive called *Contract* for specifying the agreed contract of service providers in terms of cloud both static and/or quality attributes and

constraints over them, and also the pricing model of the services. Within such element associated to a given service, there are four main elements: (i) the *QualityParameters* element, in which the properties regarding the quality parameters are specified; (ii) the *PricingModel* element, which defines details about the pricing model of the service; (iii) a *constraints* section specified in Armani [14], which determinate the minimal warranties offered by the service provider related to the quality parameters, and; (iv) a section in which the software architect can *annotate* any additional information using the conventional *Property* element defined in ACME. In addition, Cloud-ADL introduces the *Dynamic* clause associated to elements of the *Contract* that enables to specify monitorable parameters at runtime.

Fig. 2 shows an example of definition of the set of the services provided by the Amazon Web Services (AWS) platform [15], which represented by the *AmazonWebServices Family* element (line 1). Due to space restrictions, in Fig. 2 we present only one of the services provided by the AWS platform, the *Amazon S3* storage service, which is represented by the *AmazonS3 Component Type* (line 2). Similarly to ACME, the services can be annotated using *Property* elements, for instance, the *description* property (line 3), which describes the service. Moreover, within the *Contract* element at line 4, two QoS parameters, namely *response_time* (the response time to execute the service) and *availability* (the percentage in which the service is available in a given time interval), are defined in the *QualityParameters* element. Next, an Armani invariant specifies that the service provider warranties a minimal availability of 99% for this service according (line 10). Finally, the pricing model is detailed in the *PricingModel* element (lines 13 to 17).

```
1.   Family AmazonWebServices = {
2.      Component Type AmazonS3 = {
3.         Property description : String = "File storage service";
4.         Contract = {
5.            QualityParameters = {
6.               Dynamic Property response_time = Integer
7.                  << nature : String = "negative"; unit : String = "ms" >>;
8.               Dynamic Property availability = Float
9.                  << nature : String = "positive" >>;
10.              Invariant availability >= 0.99;
11.           };
12.
13.           PricingModel = {
14.              Property reference : String = "http://aws.amazon.com/s3/#pricing";
15.              Property unit : String = "$";
16.              Property billingMethod : String = "per GB";
17.           };
18.
19.           Property SLADetails : String = "http://aws.amazon.com/s3/s3-sla/";
20.        };
21.     };
22.     ...
23.  };
```

Fig. 2. Partial definition of the cloud services provided by the AWS platform.

2) Specification of a cloud application itself in Cloud-ADL. The specification of an application in Cloud-ADL is very similar to the ACME architectural description of systems by using the same basic architectural elements defined in such ADL. As the services are the basic functional entities (components) of the application, in a Cloud-ADL architectural description, cloud services (represented by *Component* elements) are instances of the services (represented by *Component Type* elements) defined in the *Family* elements that abstract the cloud platforms, thus following the ACME principle in which architectural elements defined within a *Family* element can be inherited by all systems that adhere to it. In this perspective, a *System* element that describes a cloud application must extend (adhere) the *Family* element that represent the cloud platforms that are providers of the services used by the application. For instance, in the statement *System A : B, C*, the application *A* uses services provided by the cloud platforms represented by the *B* and *C Family* elements. In Cloud-ADL, describing a component *x* in a *System* element as an instance of a component type *y* defined within a *Family* element means that the application component *x* uses the cloud service defined by the component type *y*.

As the *Dynamic* clause present in Cloud-ADL supports the specification of monitorable parameters at runtime and these dynamic attributes may change over time, a *reconfiguration action* is required in order to better satisfy the application needs. In this perspective, Cloud-ADL currently addresses dynamic reconfiguration in terms of architectural *programmed changes*, which can be foreseen at design time, thus following ideas proposed in the Plastik framework [16], which extends the ACME/Armani ADL to enable this dynamic reconfiguration support. In this context, Cloud-ADL also includes a *predicate–action* construct to specify the situations that trigger reconfigurations (*when*) and which reconfigurations (*what* should be changed) must take place. Such predicate is specified by using the standard Armani predicate syntax and refers to dynamic properties attached to the application components, e.g. regarding to quality parameters. In turn, the actions to be taken contains ACME-based statements that are instantiated when the predicate clause becomes true. Finally, another extension consists of the *detach* and *remove* elements, which specify the destruction of existing ACME elements. The *detach* element is used to remove an attachment between a port and a role, and the *remove* element is used to destroy an existing component, connector or representation. As the removal of a component is only possible when it is no longer involved in an attachment, it is firstly necessary to detach the component to be removed from other components and afterwards remove it from the architecture.

2.2 Cloud supporting mechanisms

There has been a good amount of research over past few years addressing the potential of coupling ADLs with underlying runtime environments to support systematic and integrated *specification-to-deployment* architectures [17] when architecting dependable systems. As a consequence, it is necessary to provide means to *causally connect* high-level architecture specifications and the underlying runtime level, which defines the deployment, execution, and reconfiguration environment of the applications. In this perspective, as we have mentioned in Section 2.1, Cloud-ADL provides constructors that enable to specify programmed dynamic reconfiguration actions to be taken at the runtime level. The integration between such levels is supported by a *causal connection*

between the architectural and runtime levels, more precisely in an ADL-to-runtime direction [18] through proper *mappings* from the architecture-level concepts to the runtime-level elements, similarly to the Plastik framework [16]. However, in Plastik, such mappings are directly performed from the architectural level to the runtime level; in our environment, the mappings are intermediated by the cloud-based middleware platform.

As also illustrated in Fig. 1, a *cloud-based middleware platform* is placed between the architectural and runtime levels in order to enable the manipulation (both at design time and runtime) of the underlying services provided by cloud platforms and to be used by the applications. Furthermore, such middleware platform provides transparent and automatic integration, thus enabling the use of services provided by different cloud platforms and abstracting away the specificities of each one in a heterogeneous cloud scenario. Considering the purposes of our environment, we have identified three minimal services that should be provided by such middleware platform:

1. a *composition service*, which represents a composition mechanism that enables applications to use services provided by different cloud platforms;
2. a *monitoring service*, which is responsible for the runtime assessing and periodic monitoring of the dynamic QoS parameters specified at the architectural level, as well as for detecting considerable variations in such parameters, and;
3. an *adaptation service* which manages the dynamic reconfiguration process at runtime in order to ensure the availability and quality of the applications under conditions that may affect their execution, e.g. in cases of service failures or quality degradation.

In our environment, we have chosen to use the *Cloud Integrator* service-oriented middleware platform [19], which is part of our previous work related to management of cloud services. Besides being in accordance with W3C open standards and languages for Web services, *Cloud Integrator* works as a mediator between the service providers and the applications (clients) and already provides the mentioned requirements, in terms of service composition [20], autonomous adaptation of cloud applications [21], and; (iii) integration with a monitoring strategy for cloud services [22]. It is very important to emphasize that the mentioned cloud supporting runtime mechanisms are already implemented, so that this Ph.D. work is focused on the architectural level and on the causal connection between such levels.

3 Related work

The work related of the subject of this Ph.D. thesis regards to three fundamental elements: (i) ADLs for describing service-oriented and/or cloud-based applications; (ii) dynamic reconfiguration of software architectures, and; (iii) architectural runtime support for cloud environments. As we have mentioned, to the best of our knowledge, there are no proposals in the literature for modeling Cloud Computing applications and that satisfactorily capture important and specific features in such context. Consequently, in this section we present some related works in terms of architectural support for service-oriented applications in two concerns: (i) the notion of *services* in architectural descriptions, and; (ii) the use of architectural *contracts* for expressing non-funcional

aspects and describing resources to be used by an application. As ongoing work, we are performing systematic reviews of the literature regarding the other two subjects.

Rennie and Miŝić [11] identify a set of requirements present in service-based applications. By analyzing some existing ADLs such as Darwin and C2, the authors state that these ADLs would not be suitable to describe service-based architectures despite covering aspects such as dynamicity, which is an essential characteristic in this context. Moreover, components, which represent the services in such kind of architecture, should contain information about the resources that they use, functional (e.g. inputs and outputs) and non-functional (e.g. quality parameters) constraints, pricing models (e.g. pay-per-use model or subscription-based payment), etc. Although the discussion presented by the authors gives directions towards an ADL for describing service-based architectures, they do not propose any specific solution.

Xie et al. [4] propose a new ADL for describing service-oriented architectures called SOADL, which is based upon the XML language. SOADL has three main elements: (i) *services*, represented by components; (ii) *connectors*, which define how components are interconnected and also the used protocol(s), and; (iii) *configuration*, a graph of interacting components. In a later work [5], the authors present a refinement of the previous version of SOADL, which now supports not only the basic definition of the elements that compose the architecture, but also new elements for specifying behavior (sequence of actions) and quality parameters of the services, which are defined as properties and may have applied constraints as invariants or heuristics, similar to ACME/Armani. This latest version of SOADL supports the dynamic reconfiguration of applications, in which software architects specify alternative configurations for the applications based on events (programmed reconfiguration). However, as the SOADL approach is based on proposing a new specific ADL rather than extending an existing ADL (as we do in this work), software architects may be burdened with many new abstractions, thus increasing the learning curve of the language.

In terms of using contracts in architectural descriptions through ADLs, Loques et al. [12] and Ansaloni et al. [13] use the notion of *architectural contracts* [10] to express non-functional requirements (static and/or dynamic) in the architectural description and to describe, at design time, the resources to be used by the application. In this perspective, these works propose a framework called CR-RIO (*Contractual Reflective-Reconfigurable Interconnectable Objects*) conceived for specifying and supporting QoS contracts associated with the architectural components of an application. CR-RIO has its own ADL called CBabel, which is used to describe not only the functional components of the application and their relationships, but also QoS non-functional aspects, such as processing capacity, transport, etc. In CR-RIO, QoS contract is basically composed by four elements: (i) *QoS categories*, which are described apart from components and used to aggregate properties related to non-functional requirements; (ii) *QoS profile*, which specifies quantified constraints over these parameters; (iii) the *services*, which define a set of non-functional requirements to be applied over the components of the application, and; (iv) a *negotiation clause*, which specifies alternative configurations for the application when a non-functional requirement is not fulfilled. Although this idea of using architectural contracts to express non-functional requirements has inspired the creation of the *Contract* element in Cloud-ADL to specify features related to the Cloud Computing context, the proposed *Contract* element is

more comprehensive, thus enabling to express not only QoS parameters and associated constraints, but also information related to the pricing models adopted by the services and the established agreements between service providers and consumers. In addition, the negotiation clause used by CR-RIO can be viewed as a mean of supporting reconfiguration of applications, but this reconfiguration is not performed based on dynamic monitorable parameters, which is a very relevant issue because cloud environments are intrinsically dynamic.

4 Final remarks

In this paper we presented our ongoing work on the specification and implementation of an architecture-driven environment for the development of cloud-based applications. The environment is built on two causally-connected levels, namely an architecture level and a runtime level, and also supports the dynamic adaptation of such applications according to their architectural requirements. At the architectural level we defined Cloud-ADL, a seamless extension of the ACME ADL to represent cloud services, contracts (including quality metadata and pricing), and dynamic reconfiguration constructs, whilst at the runtime level we use *Cloud Integrator*, a service-oriented middleware platform that supports service composition, monitoring of cloud services and autonomous adaptation of cloud applications.

The next activities of this Ph.D. work encompasses:

1. possible refinements in the Cloud-ADL metamodel, including the specification of the quality parameters to be monitored and the evolution of architectural descriptions (probably following existing proposals in terms of evolutive architectures, such as [23] and [24]);
2. mapping between the architectural level and the runtime level, encompassing the *Cloud Integrator* middleware elements;
3. the development of a software toolkit for describing and constructing cloud-based applications based on the adaptive composition of services in cloud environments;
4. development of complex applications (case studies) that will use services provided by different cloud platforms, and;
5. tests and quantitative/qualitative evaluations of the proposed environment.

References

1. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, USA (2011)
2. Chhabra, B. et al.: Software Engineering issues from the cloud application perspective. International Journal of Information Technology and Knowledge Management 2(2), pp. 669–673 (2010)
3. Sriram, I., Khajeh-Hosseini, A.: Research agenda in cloud technologies. Computing Research Repository, Cornell University, USA (2010)
4. Xie, D. et al.: An approach for describing SOA. In: 2006 IEEE International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4, IEEE Computer Society, USA (2006)

5. Jia, X. et al.: A new architecture description language for service-oriented architecture. In: 6th International Conference on Grid and Cooperative Computing, pp.96–103, IEEE Computer Society, USA (2007)
6. Clements, P.: A survey of architecture description languages. In: 8th International Workshop on Software Specification and Design, pp. 16–25, IEEE Computer Society, USA (1996)
7. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering 26(1), pp. 70–93 (1996)
8. Cavalcante, E. et al.: Describing architectures of cloud applications. In: 7th European Conference on Software Architecture, Springer-Verlag Berlin/Heidelberg, Germany, 320-323 (2013)
9. Garlan, D. et al.: ACME: An architecture description interchange language. In: 1997 Conference of the Centre for Advanced Studies on Collaborative Research, pp. 169–189, IBM Press, USA (1997)
10. Meyer, B.: Applying design-by-contract. Computer 25(10), pp. 40–51.
11. Rennie, M., Mišić, V.: Towards a service-based architecture description language. Technical report, University of Mantioba, Canada (2004)
12. Loques, O. et al.: A contract-based approach to describe and deploy non-functional adaptations in software architectures. Journal of the Brazilian Computer Society 10(1), pp. 5–20 (2004)
13. Ansaloni, S. et al.: Deploying QoS contracts in the architectural level. In: 2004 Workshop on Architecture Description Languages, pp. 19–34, Springer-Verlag, USA (2005)
14. Monroe, R.: Capturing software architecture expertise with Armani. Technical report, Carnegie Mellon University, USA (1998)
15. Amazon Web Services: `http://aws.amazon.com`
16. Batista, T. et al.: Managing dynamic reconfiguration in component-based systems. In: 2nd European Workshop on Software Architecture, LNCS, v. 3527, pp. 1–17, Springer-Verlag, Germany (2005)
17. Joolia, A. et al.: Mapping ADL specifications to an efficient and reconfigurable runtime component platform. In: 5th Working IEEE/IFIP Conference on Software Architecture, pp. 131–140, IEEE Computer Society, USA (2005)
18. Gomes, A.T.A. et al.: Architecting dynamic reconfiguration in dependable systems. In: Lemos, R. et al. (eds.) Architecting dependable systems IV, LNCS, v. 4615, pp. 237–261, Springer-Verlag, Germany (2007)
19. Cavalcante, E. et al.: *Cloud Integrator*: Building value-added services on the cloud. In: First International Symposium on Network Cloud Computing and Applications, pp. 135–142, IEEE Computer Society, USA (2011)
20. Cavalcante, E. et al.: In: 2012 IEEE Latin America Conference on Cloud Computing and Communications, pp. 31–36, IEEE Computer Society, USA (2012)
21. Cavalcante, E. et al.: Autonomous adaptation of cloud applications. In: 13th International IFIP Conference on Distributed Applications and Interoperable Systems, LNCS, v. 7891, pp. 175–180, IFIP/Springer-Verlag, Germany (2013)
22. Almeida, A. et al.: Towards a SPL-based monitoring middleware strategy for Cloud Computing applications. In: 10th International Workshop on Middleware for Grids, Clouds and e-Science, ACM, USA (2012)
23. Oreizy, P. et al.: Architecture-based runtime software evolution. In: 20th International Conference on Software Engineering, pp. 177-186, IEEE Computer Society, USA (1998)
24. Georgiadis, I. et al.: Self-organising software architectures for distributed systems. In: First Workshop on Self-Healing Systems, pp. 33-38, ACM, USA (2002)