User Centric Service Level Management in mOSAIC Applications[⋆]

Massimiliano Rak, Rocco Aversa,
Salvatore Venticinque, and Beniamino Di Martino

Dipartimento di Ingegneria dell'Informazione, Seconda Università di Napoli
massimiliano.rak@unina2.it

Abstract. Service Level Agreements (SLAs) aims at offering a simple and clear way to build up an agreement between the final users and the service provider in order to establish what is effectively granted by the cloud providers. In this paper we will show the SLA-related activities in mOSAIC, an european funded project that aims at exploiting a new programming model, which fully acquires the flexibility and dynamicity of the cloud environment, in order to build up a dedicated solution for SLA management. The key idea of SLA management in mOSAIC is that it is impossible to offer a single, static general purpose solution for SLA management of any kind of applications, but it is possible to offer a set of micro-functionalities that can be easily integrated among them in order to build up a dedicated solution for the application developer problem. Due to the mOSAIC API approach (which enable easy interoperability among moSAIC components) it will be possible to build up applications enriching them with user-oriented SLA management, from the very early development stages.

1 Introduction

Cloud Computing is the emerging paradigm in distributed environments. In cloud solutions everything from hardware to application layers are *delegated* to the network. Following this main idea many different technologies and solutions are born and assumed the *cloud hat.*

Cloud Computing technologies are moving along two orthogonal directions: on one side they aim at transforming existing datacenters into Cloud Providers, which offer everything in the form of a service (*as a service* paradigm); on the other side they aim at building applications and solutions that, as much as possible, fit with user's needing.

As an example, one of the more important effects of the Cloud Computing paradigm is the extreme flexibility in application and system re-configurability: it is incredibly simple to acquire new resources and to cut away them if they are not required. Due to this flexibility and capability of self-adaptation to the request, cloud applications become *User Centric.* It means that the role of the final user is central. The quality metrics (availability, performance, security) should aim at

[⋆] This research is partially supported by FP7-ICT-2009-5-256910 (mOSAIC).

improving the quality perceived from users, while resource administration and optimization assume the role of acquiring the right amount of resources, which are compliant with the needing of the final users, instead of optimizing the usage of already acquired ones.

Service Level Agreements (SLAs) aim at offering a simple and clear way to build up an agreement between the final users and the service provider in order to establish *what is effectively granted.* A *Service Level Agreement (SLA)* is an agreement between a Service Provider and a Customer, that describes the Service, documents Service Level Targets, and specifies the responsibilities of the Provider and the Customer.

From User point of view a Service Level Agreement is a contract that grants him about what he will effectively obtain from the service. From Application Developer point of view, SLAs are a way to have a clear and formal definition of the requirements that the application must respect.

But, in such a context, how it is possible for an application developer to take into account the quality perceived by *EACH* final user of his application? This problem is solved by the adoption of SLA templates (i.e. predefined agreements offered to final users). This means that developers must identify *at design time* the constraints to be fulfilled, the performance index to be monitored and the procedure to be activated in case of risky situations (i.e. when something happens that may lead to disrespect the agreement).

At the state of art many research efforts have been spent in order to define standards for SLA description (WS-Agreement [1], WSLA [6]) or operative framework for SLA management (SLA@SOI[10,2], WSAG4J[11]). As it is shown more in detail in the related work section 2, it is fully recognized the needing of SLA management in the cloud context, but there are, at the state of art, no clear proposals about an innovative approach to SLA management that takes into account the *User Centric* view, which is typical of the cloud environment.

In this context the mOSAIC project [4,7] proposes a new, enhanced programming paradigm that is able to exploit the cloud computing features, building applications which are able to adapt themselves as much as possible to the available resources and to acquire new ones when needed (more details in section 3). mOSAIC offers together an API for development of cloud applications which are flexible, scalable, fault tolerant, provider-independent and a framework for enabling their execution and access to different technologies.

The key idea of SLA management in mOSAIC is that it is impossible to offer a single, static, general purpose solution for SLA management of any kind of applications, but it is possible to offer a set of micro-functionalities that can be easily composed in order to build up a dedicated solution for the application developer problem. In other words, thanks to mOSAIC API approach (which enables easy interoperability between moSAIC components) it will be possible to build up applications with user-oriented SLA management features, from the very early development stages.

The reminder of this paper is organized as follows: next section (section 2) summarizes the state of art of SLA management solutions, while the following one briefly summarizes the main concepts related to mOSAIC API and how it is possible to develop applications using mOSAIC. Section 4 proposes the vision of the SLA problem in the context of cloud applications, which is detailed in the section dedicated to the architectural solution 5. A brief section dedicated to examples (section 6) shows how the approach has been applied in simple case studies. The paper ends with a section dedicated to the current status, future work and conclusions.

2 Related Work

As anticipated in the introduction a lot of research work exists on the problem of Service Level Management in cloud. The biggest part of this work focuses on building solutions which make possible to a cloud provider to offer (using the SLA) its services with a granted quality. In this direction the most interesting results come from SLA@SOI [10,2]. It is an European project that aims (together with other relevant goals) at offering an open source based SLA management framework. SLA@SOI results are extremely interesting and offer a clear starting basis for the SLA management in complex architectures. The main target of the project goes in the direction of enriching the Cloud Provider offer: they aims at developing a solution that can be integrated with Cloud technologies (like Open Nebula) in order to offer Cloud Services (mainly Infrastructures and image management services) trough an SLA-based approach. SLA@SOI offers solutions to design SLAs together with the offered services and to generate and manage them trough many different representations. The CONTRAIL project [3] proposes a complex architecture for building cloud providers for both Infrastructures and Platform as a services solutions. In their proposal they assumed an heavy role for the SLA management: each service request is enriched with a SLA, and service evolution in the architecture is traced together with its own SLA. They are reusing and applying many of the SLA@SOI technologies in order to build up their solutions. Even in the OPTIMIS project [8], which aims at building a Platform as a service solution on the basis of federated clouds (i.e they offer a complex dynamic system able to acquire virtual machines and storage resources from many different providers and deploy over them enterprise solutions), the role of SLA is very relevant. In Optimis, moreover, they have special care in managing the dynamicity and elasticity of acquired resources. OPTIMIS needing puts in evidence an additional aspect of the SLA management in cloud environment: the autonomicity, the needing of building solutions which are able to self-manage themselves. SLA and Autonomic approaches are strictly related each other, but partially in conflict: how to grant an agreement for a given specific service on resources that dynamically adapt their behavior due to the dynamic workload changes (how many requests, which kind of requests, ...) in a completely independent way? This topic is completely open and, at the best of author's knowledge there are no available solutions. mOSAIC Project assumes that these two requirements are complementary and equally needed: the application developer needs solutions able to self-adapt themselves and, at the same time, to have a clear and full notion of the state of the system allowing him to take decisions, which lead to offer the grants needed in SLA management.

3 mOSAIC API

In mOSAIC a Cloud Application is developed as a composition of inter-connected building blocks. A Cloud "Building Block" is any identifiable entity inside the cloud environment. It can be the abstraction of a cloud service or of a software component. It is controlled by user, configurable, exhibiting a well defined behavior, implementing functionalities and exposing them to other application components, and whose instances run in a cloud environment consuming cloud resources.

Simple examples of components are: a Java application runnable in a platform as a service environment; or a virtual machine, configured with its own operating system, its web server, its application server and a configured and customized e-commerce application on it. Components can be developed following any programming language, paradigm or *in-process* API. An instance of a cloud component is, in a cloud environment, what an instance of an object represents in an object oriented application.

Communication between cloud components takes place through cloud resources (like message queues – AMPQ, or Amazon SQS) or through non-cloud resources (like socket-based applications).

Cloudlets are the way offered to developers by mOSAIC API to create components. Cloudlet runs in a cloudlet container that is managed by the mOSAIC Software platform. A Cloudlet can have multiple instances, but it is impossible at runtime to distinguish between two cloudlet instances. When a message is directed to a cloudlet, it can be processed by anyone of the cloudlet instances. The number of instances is under control of the cloudlet container and is managed in order to grant scalability (respect to the cloudlet workload). Cloudlet instances are stateless.

Cloudlets use cloud resources trough connectors. Connectors are an abstraction of the access model of cloud resources (of any kind) and are technology independent. Connectors control the cloud resource trough technology-dependent *Drivers*. As an example, a cloudlet is able to access to Key-Value store systems trough a KVstore Connector, it uses an interoperability layer in order to control a Riak, or a MemBase KV driver.

Therefore a Cloud Application is a collection of cloudlets and cloud components interconnected trough communication resources.

Details about the mOSAIC programming model and about the Cloudlet concept, whose detailed description is out of the scope of this paper can be found in [4,5,7].

4 Vision and Approach

Consider a service developer who is using the mOSAIC project solution in order to build up a new cloud application that offers complex services to its own users.

Providers offer simple SLA, just granting the availability of the resources and (in some cases) the bandwidth and latency for the access network (see

Parameters). The cloud application will be accessed by a web interface, whose main functionalities are complex operations which may need some seconds to be performed. The developer would like to offer to its user a SLA, in order to grant average and maximum response time, together with a given availability. The SLA problem for a Cloud Application can be now described in the following terms: how can the developer ensure the user requirements in terms of SLA (parameters like average and maximum response time, service availability, ...) using resources which are not under its control and whose SLA grants different kind of service properties (e.g. availability of the VM)?. What the mOSAIC Platform should offer to developers in order to solve this problem?

The problem of Service Level Agreement in mOSAIC can be modeled by two layers. At each layer there is a different agreement: (1)Agreement between final users and Application (2) Agreement between Application and Cloud providers The agreement between users and the application (point 1) means that the application and the user shares an SLA. Inside the cloud application a negotiator concludes the agreement that will be shared with the final user. Moreover, from the developer's point of view, we need tools able to predict the application behavior, to monitor its evolution, to modify the application behavior and/or the cloud resources acquired in order to: (a) accept/refuse the agreements, (b) identify risky situations for stipulated agreements, (c) apply the actions needed to grant a stipulated agreements.

About the cloud resources to be acquired (point 2), the developer has to search and choose the Cloud Providers, negotiating with them (acting as a client) the SLA and monitoring the promised service levels. Note that the application is forced to search for a provider which allow to grant as much as possible the ones offered to the user. Moreover, at the state of the art, no Cloud Provider offers negotiable SLA, but they just offer a set of static SLAs. It is the mission of a federation framework (like mOSAIC) to build up a SLA negotiation system on the top of the existing ones. Agreement enforcement, instead, implies a large set of different problems: monitoring the state of the resources in order to identify risky situations, execution of recovery procedures in order to react to dangerous states and, last but not least, agreement mapping or delegation. Agreement mapping (or delegation), means definition of the SLA terms which can be offered to final users on the basis of the SLA offered by the providers. Agreement enforcement implies the capability of predicting the application behavior. This is strictly dependent on the application and can be performed only from the service developer, using techniques and tools application dependent (i.e. Petri Nets, simulation techniques or analytical models). From the mOSAIC point of view the above presented problem can be managed as follows: the first Agreement problem (user-application) should be solved at Software Platform Level, i.e. including in the API a set of components which let the final application to easily build up a negotiation system and using tools able to change the kind or the quantity of acquired cloud resources. The second Agreement problem (application-providers) is solved by the Cloud Agency.

5 General Overview of the SLA Architecture

The main assumptions in SLA management for mOSAIC applications can be summarized as follows:

- *microfunctionality approach*: mOSAIC will offer a set of cloudlets and components whose interface is defined in terms of exchanged messages. Application developers have to simply use the default components, connect them trough queues and develop the ones needed to build up his own custom solution.
- *User Centric Approach*, which means that it should be possible for an application developer to maintain information for each user, taking into account his requirements and what the application promised to him.
- Support for autonomicity of the components. In cloud environment a lot of efforts exists in development of applications, resources, services with elastic features, i.e. able to change their behavior due to execution conditions.

The key idea of the microfunctionalities approach is to provide a set of components, even for SLA management, to be composed together the functional ones according to the application requirements and to its logic. This choice is mandatory because there is not only one solution that solves the SLA general problem independently from the application.

mOSAIC global architecture is composed of: mOSAIC API, which includes the SLA components, the framework, which uses a provisioning system (Cloud Agency) and the tools needed to run the mOSAIC applications (cloudlet containers, application deployers, ..).

Negotiation of SLA with cloud providers on a federation basis is completely solved by the Cloud Agency [9]. Here the Cloud Agency is used as a black box that can book on user behalf the best set of cloud resources for his application, after that the developer has defined requirements of desired resources. Cloud Agency will be able to offer negotiable SLAs on the top of many different commercial cloud providers. The Cloud Agency solves the problem of Agreement between applications and Cloud Providers, offering the **Negotiator**, which implements the SLA negotiation towards a large set of different CPs and, once the SLA is approved, delivers the resources to the application.

SLA User Negotiation. This module contains all the cloudlets and components which enable interactions between user and the application in terms of SLA negotiation.

SLA Monitoring/Warning System. This module contains all the cloudlets and components needed to detect the *warning conditions* and generates alerts about the difficulty to fulfill the agreements. It should address both resource and application monitoring. It is connected with the Cloud Agency.

SLA Autonomic System. This module includes all the cloudlets and components needed to manage the elasticity of the application, and modules that are in charge of making decisions in order to grant the respect of the acquired needed to fulfill the agreements.

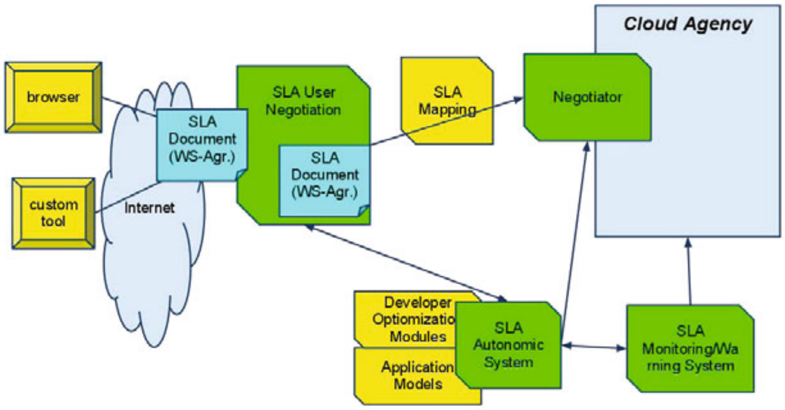

Fig. 1. mOSAIC SLA Module Organization

6 An Example of mOSAIC Approach

In the following we aim at showing in practice how the approach works on a simple example, derived from the mOSAIC Case studies. The mOSAIC Service Developer has an application developed on the top of a job submission system and aims at developing a cloud application, which offers SLAs and controls the job submission system, varying parameters like the number of virtual machines running in the cloud. User's behaviour can be easily described: each user has a job description that is filled in a User Request file, the user invokes the cloud application offering the file as a parameter. On the Service Provider portal the user is able to access the state and available results of the application execution.

The mosaic developer aims at building an application, which acquires virtual clusters from commercial cloud providers, configured in order to be able to execute the requested jobs. Moreover the developer will like to offer the same service at different quality levels (varying grants like exclusive use of machine or the maximum cpu consumption allowed), expressed trough SLAs. The developer needs the ability to monitor the SLA and to know what is granted to each different user.

Adopting an SLA based approach, the user's behaviour changes as follows: (a) User subscribes to the application signing an Agreement (SLA) (Agreement use case) (b) User submits his job request to the application (Submit use case) (c) User queries the portal about the status of his requests (Check).

In order to manage the SLAs, the application has the following duties:

– maintains the list of all the agreements signed;
– maintains the list of all the requests done;
– acquires and maintain resources from cloud provider in order to execute the requests of the users;
– monitors the SLA fullfillment;
– activates procedures in risky situations.

The complexity of the application depends on the grants offered by the SLA and on the kind of target application running on the top of the job submission system (as an example: is it easy to predict its response time?). The complexity due to the application behaviour (its predictability, the action to take in order to grant application dependent parameters, ..) cannot be defined in general. On the other side the management of the SLA toward the user (negotiation), the monitoring of resource status, the management of the SLA storage are common to all the applications. In the following we will focus on the components offered in mOSAIC for such requirements.

As a first step we design the SLA that the developer aims at offering to the users, in order to simplify the approach we model it just by two simple parameters: the maximum amount of Credits the final user wants to pay and the maximum number of requests the user is allowed to submit. Moreover the application assures that the services will be offered on dedicated resources (they will not sell the same resources to two users). This agreement is represented as a WS-Agreement template, some pieces of them are described in listing 1.1

Listing 1.1. Example of User SLA request in WS-Agreement

```
[..]
<wsag:VariableSet>
    <wsag:Variable wsag:Name="UserRequests" wsag:Metric="
        xs:integer">
        <wsag:Location>
            $this/wsag:Terms/wsag:All/
                wsag:ServiceDescriptionTerm
            [@wsag:Name = 'Term1']/mosaic:JobSubmission/
                MaxRequests
        </wsag:Location>
    </wsag:Variable>
        <wsag:Variable wsag:Name="UserCredit" wsag:Metric="
            xs:integer">
            [..]
<wsag:GuaranteeTerm wsag:Name="MaxCredit">
    <wsag:ServiceLevelObjective>
        <wsag:KPITarget>
            <wsag:KPIName>MaxCreditLevel</wsag:KPIName>
            <wsag:CustomServiceLevel>UserCredit GT 0</
                wsag:CustomServiceLevel>
        </wsag:KPITarget>
    </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>
<wsag:GuaranteeTerm wsag:Name="MaxRequests">
[..]
```

Note that the monitoring of the SLA can be done independently from the state of the acquired resources, but just by tracing the requests. This means that we will not use the autonomic and monitoring modules of the SLA architecture.

Once the SLA has been defined, we show briefly how to design the application, whose behaviour includes the SLA negotiation and agreement storing, once it has been signed. For each user's request the application evaluates the acquired resources, the available credit, eventually starts new resources and then submits the job to the acquired VC.

Following the microfunctionalities approach the application can be designed as in picture 2. The mOSAIC API offers a simple `SLAgw` component, which implements the WS-Agreement protocol (toward the final user) and sends messages on predefined queues in order to update the application. As a consequence the programmer has to develop few cloudlets: an **Agreement Policy Cloudlet**, which has the role to accept or not an SLA, a **Request Cloudlet**, which has the role of fowarding the user requests to the job suibmission system, and two cludlets,**Resource Policy Cloudlet** and **Guarantee Policy Cloudlet**, which have respectively the role of tracing the acquired resources and generate warning for risky conditions. Cloudlets cooperate only trough message exchange, coordinating their actions. As an example, Agreement Policy Cloudlet receives the messages from SLAgw each time a new SLA requests takes place. Moreover it sends messages to the SLAgw in order to update the agreement state and to query about the status of the Agreements. Messages data are represented in JSON (that helps when data need to be stored in a KV store). As an example the messages sent by the SLAgw are JSON representations of ServiceTypes and GuaranteeTerms extracted from the WS-Agreement. Note that they can be customized by the final user (WS-Agreement standard is open to this) and only final user knows how to represent them. The Monitoring cloudlet regularly checks the status of each user and eventually applies penalties (not reported in the WSAG for simplicity and sake of space).

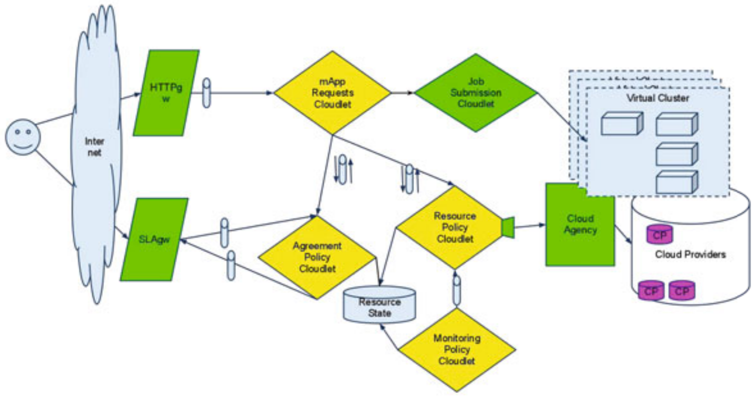

Fig. 2. Example of SLA-based application

7 Conclusions

As outlined in section 2, the management of Service level agreement is an hot topic in cloud environment. In the mOSAIC project, which aims at designing and developing a cloud provider independent API, we propose a set of features that should help the application developer to integrate SLA management in his aplications. In this paper we have outlined the vision and approach proposed in mOSAIC, the organization of the framework dedicated to SLA management and a simple example in which we have designed a SLA management system around a simple cloud application.

References

1. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement specification (ws-agreement). In: Global Grid Forum. The Global Grid Forum, GGF (2004)
2. Comuzzi, M., Kotsokalis, C., Rathfelder, C., Theilmann, W., Winkler, U., Zacco, G.: A Framework for Multi-level SLA Management. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 187–196. Springer, Heidelberg (2010), `http://dx.doi.org/10.1007/978-3-642-16132-2-18`, doi:10.1007/978-3-642-16132-2-18
3. CONTRAIL: Contrail: Open computing infrastructres for elastic computing (2010), `http://contrail-project.eu/`
4. Leymann, F., Ivanov, I., van Sinderen, M., Science, B.S.S., Publications, T. (eds.): Towards a cross platform Cloud API. Components for Cloud Federation (2011)
5. IEEE (ed.): Building an Interoperability API for Sky Computing (2011)
6. Keller, A., Ludwig, H.: The wsla framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management 11(1), 57–81 (2003)
7. mOSAIC: mosaic: Open source api and platform for multiple clouds (2010), `http://www.mosaic-cloud.eu`
8. optimis: Optimis: the clouds silver lining (2010), `http://www.optimis-project.eu/`
9. Venticinque, S., Aversa, R., Di Martino, B., Petcu, D.: Agent based cloud provisioning and management, design and prototypal implementation. In: Leymann, F., et al. (eds.) 1st Int. Conf. Cloud Computing and Services Science (CLOSER 2011), pp. 184–191. ScitePress (2011)
10. Theilmann, W., Yahyapour, R., Butler, J.: Multi-level SLA Management for Service-Oriented Infrastructures. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 324–335. Springer, Heidelberg (2008)
11. Waeldrich, O.: Wsag4j (2008), `https://packcs-e0.scai.fraunhofer.de/wsag4j/`