QoS-Aware Component for Cloud Computing

Inès Ayadi, Nöemie Simoni
Engineering School Telecom ParisTech
Paris, France
(ines.ayadi; noemie.simoni)@telecom-paristech.fr

Gladys Diaz
L2TI, Université Paris 13, Sorbonne Paris Cité
Villetaneuse, France
gladys.diaz@univ-paris13.fr

***Abstract*—Providing dedicated cloud services that ensure user's QoS requirements is a big challenge in cloud computing. Currently, cloud services are provisioned according to resources availability without ensuring the expected performances. The cloud provider should happen to evolve its ecosystem in order to meet QoS-awareness requirements of each cloud component. In this paper, we consider two important points which reflect the complexity introduced by the Cloud management: QoS-aware and self-management aspects. QoS-aware aspect involves the capacity of a service to be aware of its behavior. Self-management implies the fact that the service is able to react itself with its environment. In this paper we propose to integrate a QoS agent in each cloud component in order to control and inform the system about its current behavior.**

Keywords-Cloud computing; QoS-aware; Autonomic components; self management; Fractal ADL

I. INTRODUCTION

Cloud computing is a new trend that enables resources (Infrastructure, Platform or Software) to be exposed as services. These resources are offered using a pay-as-you-use pricing plan. The final service offered to the user consists in a set of components, which may be offered by different providers. To satisfy the request of customer, the final service must be provided in accordance with the required level of QoS (Quality of Service). QoS management must be considered to provide the attended end-to-end (E2E) QoS level. In current solutions, a degradation of a component can produce the degradation of the global service. Thus, one of the major challenges in the current cloud solutions is to provide the required services according to the QoS level expected by the user.

Cloud users expect the system to guarantee the required QoS services regardless of any unforeseeable events. Consumers needs vary unpredictably depending on their types (developer, service provider, end user) and their strategies (QoS requirement, cost effective, optimization, etc.). These unstable demands can result in SLA (Service Level Agreement) violation due to the QoS degradation of the cloud services. While cloud providers can ensure the elasticity, the high availability and the reliability of services, the QoS expectations of users are not achieved. QoS is a very important aspect that must be considered in the different phases of life-cycle of Cloud solutions. Thus, the QoS aspects should be considered from the design phase of cloud components in order to achieve the expected QoS requirements. Consequently, we think that QoS-aware is a good approach to be implemented in future dynamic cloud environments. The QoS-aware approach implies the knowledge of significant QoS information related to each life-cycle phase. In fact, Cloud environments require a dynamic configuration and management of services and resources at run time in order to ensure the expected QoS. This dynamicity and adaptability is only possible if the system is able to use the pertinent QoS information in order to predict the suitable consummation of resources needed by the applications.

Complementarily, self-management approach must be introduced to guarantee the E2E behavior of the global service. The self-management implies the ability of each service component to manage itself its behavior.

We propose in this paper the modeling of both aspects: the QoS-awareness and the autonomic management in the cloud. We suggest the modeling a QoS self-managed cloud component using the Fractal ADL. Our work is part of the OpenCloudware project [2].

This paper is organized as follows. Motivations are presented in Section II. The related work for QoS-aware and autonomic components aspects in cloud is described in Section III. Section IV gives a brief review about our QoS generic model. Our propositions for a QoS-aware component in cloud are presented in Section V. We give the modeling description of our proposed component and a use case in Section VI. Finally, in conclusion, we exhibit the advantages of our approach in Cloud Computing and the future works.

II. MOTIVATIONS

The present work is a continuation of the previous research studies performed by the UBIS project [5]. This project focuses on the user-centric approach and it aims at providing personalized services anytime, anywhere and anyhow. The goal of this project has been to ensure E2E QoS requirements of the services demanded by the end-user. For this purpose, the QoS management is handled within different layers: service, network and equipment. The resources in different layers are conceived as a set of service elements (building blocks) in order to have a fine grained QoS description. A generic QoS model [1] is proposed for define the service elements behavior. We suggest that this model could be used to deal with cloud computing issues that are related to self-management and QoS-awareness. Consequently, we apply our generic QoS model to design cloud components by considering the expected QoS requirements in the whole life-cycle. In fact, in our point of view, the cloud environment is described as a set of distributed components. A Cloud component is an independent element that provides a well known

 ISBN: 978-1-61208-257-8

functionality and a QoS level. This notion of component is generic and can be applied at different cloud levels. An IaaS (Infrastructure as a Service) component can be for example the CPU (MIPS), the VM, the network switch, etc. In the PaaS (Platform as a Service) level, a component can be an application container, a routing stack (number of packet sent, delay of treatment of each packet), etc. The SaaS (Software as a Service) component is the final service (processing time, requests number, etc.) such as a financial service, a Web service, etc. We also consider that management, control, monitoring and security functions are conceived as cloud components. This fine-grained modeling of components allows composing applications by considering E2E QoS requirements.

To more explain this, consider a user that demands a 3-tier application from the cloud provider. This application is composed by an Apache Web server, a Jonas application server and a MySql database server. By applying our model in Jonas for example, we consider this tier as a composition of several Cloud components including: CPU, memory, VM (where the Jonas server is hosted), the software of the server, the network binding, etc. Each component is described by its behavior. This allows the deployment of the Jonas software in the adequate virtual environment that is in turn be hosted in a suitable physical environment. In order to satisfy the E2E QoS requirements (SLA), the cloud provider should deploy the 3-tier application by composing adequate cloud components. For example, the Jonas server should be put up in a large VM instance (in terms of CPU and memory) in order to process requests in less than 0.1s. Consequently, the self-acknowledgement of the component's behavior allows the mapping of each virtual application (Vapp) in the suitable environment.

Furthermore, our model deals with the unified control of QoS aspects, and provides an answer to guarantee the required QoS services regardless of any unforeseeable events. *To do this, a QoS-agent will be integrated into cloud components in order to perform dynamic adaptation according to QoS requirements.*

III. RELATED WORK

Providing a QoS-aware solution requires autonomic capabilities in the cloud components. In this section, we review some approaches treating the two aspects: QoS-aware and self-management components in cloud computing.

A. QoS-Aware cloud

S. Ferretti et al. [6] proposes a QoS-aware cloud architecture that aims to satisfy QoS requirements of the application. The principal function of this architecture is the efficient resources management of the virtual execution environment associated to the application. This architecture includes features that eliminate resources over-provisioning by changing and configuring the amount of resources dynamically. But how can describe the behavior of an application?

R. Nathuji et al. [7] propose "Q-Cloud", a QoS-aware control framework that manages resources allocation in order to alleviate consolidated workload interference problem. The principal aim of this framework is to dynamically allocate resources of co-hosting applications based on QoS requirements. Q-States (QoS states) notion is proposed in order to assign additional QoS levels to the application. The goal of these states is to offer additional flexibility to the user in order to easily improve his application-specific QoS level. But QoS levels generate different behaviors, then, can we talk about the same service?

That is why, QoS monitoring feature presented in [9] according to "as a service" paradigm is interesting. This facility ensures a continuous control of QoS attributes in order to avoid SLA violation.

H. Nguyen Van et al. [8] attempt to manage autonomic virtual resources for hosting cloud services. It proposes a two-level architecture that separates application's QoS specifications from the allocation and provision of resources. An application-specific local decision module is proposed within each application in order to analyze the QoS requirements of the hosted service. This module determines a high-level performance goal in order to make the best decision in allocation and provision phases.

While the cited solutions aim to satisfy QoS requirements of applications, the management is still resource-based by adapting resource reservation to QoS requirements. The QoS-aware aspects are not addressed. A QoS-aware component should provide the same expected service and the same intended QoS level. In others words, Cloud Services must maintain the same behavior even if the environment conditions are changed. Thus, we propose, through this paper, a QoS-aware Cloud component that can itself control its behavior.

B. Self-managed components in cloud

An autonomic cloud components should intrinsically integrates the dynamic adaptation and self-management capabilities in order to meet the non-functional requirements. We present in this section a review of some references of this context.

F. Zambonelli et al. [10] propose the management of service components that are able to adapt dynamically their behavior according to the changes perceived in their environment. The research issues include the identification mechanisms, to enable components to self-express the most suitable adaptation scheme and acquiring the proper degree of self-awareness to enable putting in action self-adaptation and self-expression schemes. The rule execution model provides mechanisms to dynamically detect and handle rule conflicts for both, behavior and interaction rules.

H. Liu [11] et al. propose an "Accord framework" that enables the development of autonomic elements and their autonomic composition. They provide rules and mechanisms for reconciliation among manager instances, which is required to ensure consistent adaptations. For example, in parallel Single Component Multiple Data (SCMD) each processing node may independently propose different and possible conflicting adaptation behaviors based on its local state and execution context. Several others research papers have been related on the self-management of cloud services such as [12], [13], and [14]. They implement the loop MAPE

 ISBN: 978-1-61208-257-8

principles (Monitoring, Analysis, Planning, and Execution) in order to maintain QoS requirements and reduce the probability of SLA violations.

These approaches suggest some paradigms to create autonomic cloud components, but they still modeled according to the monolithic approach. However, cloud computing system requires a new approach where Cloud Services are conceived as a set of independent building blocks. Moreover, these proposals focus on self-management treated by the implementation of loop MAPE principles. But, in an environment as heterogeneous as the cloud, these solutions can be difficult to implement. Is that self-management would not be a more appropriate solution?

In our approach, we consider the self-management of each service component based on QoS control. The QoS control helps to inform and prevent the degradation of component's behavior. It provides a new approach to preserve and guarantee the services of the whole System. To do this, we propose to integrate a QoS agent into the cloud component in order to allow it to manage his behavior. In fact, the integration of a QoS agent in each component allows implementing the self-management capabilities, since the agent is able to indicate whether or not the component performs its work in the normal conditions. For example, if the current values (at runtime) of a QoS criterion is exceeded over its thresholds values, the QoS-agent sends an "out-contract" to indicate that there is a problem and that it must be replaced by another ubiquitous component. To complete the self-management mechanism, we assume the existence of ubiquitous components in the Cloud environment. These ubiquitous components provide the same service and have the same QoS level.

IV. QoS Generic model overview

The principal objective of our generic QoS model is to design components by taking into account not only the processing aspects but also the management ones. For this purpose, an informational model is proposed to manage the non-functional aspects of each resource (see Fig. 1). This model is generic and unified. It applies to all layers (equipment, network, service). Resources in each layer are conceived as a set of service elements. The informational model describes the QoS criteria of each service element.

Our QoS model introduces the definition of autonomic components throughout a QoS agent. The QoS agent manages and monitors the component behavior by using QoS criteria types (see Table I).

TABLE I. QoS criteria types

	QoS Criteria			
	Availability	***Capacity***	***Delay***	***Reliability***
Description	is the portion of time that a service component makes the requested service without failure	is the processing capacity of service component during a unit of time	is the total time taken by a service component to fulfill its functions	is the compliance rate of the rendered service compared to the demanded one

Resources of each component must be allocated dynamically in accordance with their current QoS. The management of the sharing service's resources and capabilities is performed through a queue integrated in each component. The acceptance of a new request in the queue is done according to the current values of QoS criteria [1]. The QoS criteria are evaluated through three types of measurable values: conception, current and thresholds values [3] (see Table II).

TABLE II. QoS criteria values

	Value types of QoS criteria		
	Conception	***Threshold***	***Current***
Description	is the maximum capacity of the service component processing	is the limits values not to be exceeded by a service component in order to ensure a normal behavio	is the real current value of a QoS criteria in instant t. It is used to supervise the behaviour of the service component. This value would be compared with the threshold values to control the non-violation of the service capacities

The Fig. 1 shows our information QoS model that defines all these concepts. The first level represents the QoS criteria (availability, reliability, delay, and capacity) of each component. The second one shows the measured values of each criterion. Finally, the third level depicts necessary parameters to do measurement.

Our QoS model is based on four criteria: Availability, Delay, Capacity and Reliability. We briefly justify the choice of the four criteria. Our logic is how evaluate the behaviour of a given environment without being linked to its dependencies such as location, time, network type, the delivered service, the terminal, etc. Therefore, our objective is to determine QoS criteria that achieve the End-to-End transparency. We expose the transparency in four dimensions:

- Temporal transparency: a given information can be delivered anytime. This transparency dimension is associated with the availability criteria in order to evaluate how long the system (middleware, network, etc.) is in operation during the transfer.
- Distance transparency: a given information can be delivered regardless of the distance between the end-nodes. A delay criterion is associated to this dimension in order to evaluate the processing and transfer time.
- Spatial transparency: a given information can be delivered regardless of its volume. The Capacity criterion is associated to this dimension in order to evaluate the system capabilities to treat any volume of information.
- Semantic transparency: a given information can be delivered without alteration of its content. The reliability criterion is associated to this dimension in order to evaluate how the system can treat correctly the information.

 ISBN: 978-1-61208-257-8

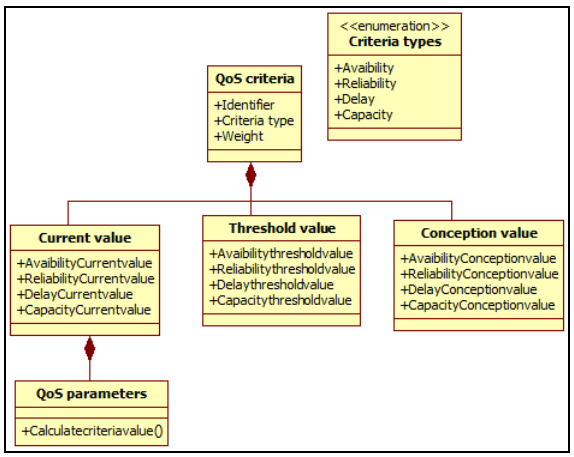

Figure 1. Information QoS model

Our QoS generic model is associated to multiple profiles [4] for managing QoS measured parameters, which are solicited during the different phases (design, deployment, operational) of the life-cycle (see Table III).

TABLE III. PROFILES IN LIFE-CYCLE

	Life-cycle phases		
	Design	***Deployment***	***Operational***
Profiles	Resource profile	Resource Usage Profile	Real Time Profile
Values of QoS criteria	Conception values	Threshold values	Current values

V. QOS-AWARE COMPONENT PROPOSITION

Our contribution consists of adding a new feature to the Cloud Service Component (CSC) in order to cope with the QoS management throughout the life-cycle. Our extension of Fractal component is an integration of a QoS component that represents the QoS agent presented in Sections II and IV. This "QoSComponent" allows managing the cloud component's behavior and enables to send notifications in the case of SLA violation or QoS degradation. We describe in this section our proposed QoS-Aware component.

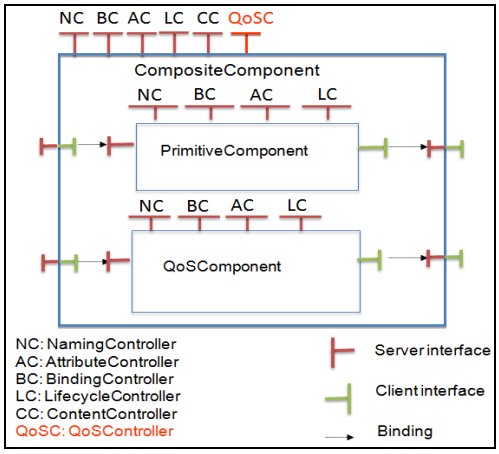

Figure 2. QoS-aware Cloud Service Component with Fractal

A. *QoS-aware Component Model Description*

Our proposition consists in an extension of the Fractal component by adding non-functional features that manage dynamically the offered QoS, like is shown in Fig. 2. The goal is to add a QoSComponent within each Fractal component. A new Fractal control interface (QoSC) is proposed to monitor and manage the QoS criteria.

B. *Characteristics of of the CSC*

The CSC are characterized by the following proprieties:

- Mutualisation: the CSC is a multi-tenant service element. Several users can share it in the same time. A CSC is stateless in order to offer the same service to all simultaneous demands.
- QoS Self-management: a CSC can self-control and self-monitor his behavior. The QoSComponent is implemented in each CSC in order to monitor the QoS criteria and to generate accurate notifications in the case of QoS degradation.
- Exposability: a CSC has a business value. Users can custom their services thanks to a portal catalogue.

C. *Functionnals aspect of the CSC*

The functional aspects represent the implementation of the offered service, including the content and the management controllers. The proposed CSC uses the following native Fractal controllers:

- Attribute Controller (AC): it manages (get, set, and update) the configuration attributes of the component. In the reconfiguration phase, this controller can modify the values of these attributes.
- Binding Controller (BC): it manages interconnections between client and server interfaces. Binding channels can be deactivated or activated depending on Fractal component life-cycle.
- Content Controller (CC): this controller manages the hierarchic architecture of the Fractal component. It adds, removes or substitutes Fractal sub-components.
- Life-cycle controller (LC): it allows the start-up and the shutdown of a Fractal component. As the QoS management is performed at many phases of the life-cycle, we propose new functionalities to this controller that will be described in the section IV.4.
- Naming controller (NC): it manages the Fractal component identification.

To ensure the QoS self-management of the cloud component, we propose a new QoS Controller (QoSC). The QoSC controller manages the CSC's behavior. This controller allows sending QoS notifications that indicate if the component maintains its behavior. These notifications are essential to take reactive decisions in the case of failures or QoS degradation.

D. *The QoS management in the Life-cycle*

The native life-cycle controller of a Fractal component allows managing the runtime phase. It handles the start-up and the shutdown states. However, the autonomic

 ISBN: 978-1-61208-257-8

management of the component's behavior requires more operations in others life-cycle phases. In fact, the QoS management is not only performed in runtime phase but also in the design and the deployment phases.

The life-cycle controller is needed to check which is the current life-cycle stage of the component and what are the constraints associated with each phase. For this purpose, we propose new functionalities of the life-cycle controller in order to maintain QoS measurements according to life-cycle phases. We define six life-cycle phases of a cloud component: design, development, deployment, runtime, billing and retirement. In the present subsection, we focus on life-cycle phases in which QoS management is performed (design, the deployment and the runtime phases).

- Design phase: represents the modeling phase of a cloud component. In this phase, each QoS criteria has conception values that determine the maximal cloud service processing capacities (e.g., server memory, CPU cores, maximal transaction per second, etc.). These values are static and unchangeable during the whole life-cycle. The life-cycle controller creates the "resource profile" containing these values.
- Deployment phase: represents the integration stage of a cloud service within the execution environment. In this phase, the life-cycle controller determines constraints of the execution environment and creates the "resource usage profile" with the threshold values of each criterion. These values show the limited capacity beyond which the cloud service's behavior becomes abnormal (e.g., the limit CPU of the virtual machine in which the cloud service is deployed).
- Runtime phase: represents the processing phase of a cloud component. The "current values" of each QoS criteria are dynamically determined throughout this stage (e.g., free disk space, current load network, requests number in the queue, etc.). These values are measured and updated by the QoSComponent and containing in the "Real-time profile".

In this stage, the life-cycle controller manages the cloud services states that are presented as follows:

- Unavailable: this state is equivalent to the shut-down state performed by the native life-cycle controller.
- Available: this state indicates that the CSC is not reserved and it is able to be used.
- Activable: in this state, the CSC is awaiting for additional information (example: login/password) to begin the execution.
- Activated: is equivalent to the start-up state. In this phase, the CSC is already in use.

E. QoS Component description

In our proposition, a QoS component is integrated in each CSC in order to allow the QoS self-management. The QoSComponent controls the CSC's behavior throughout the QoSC interface. The QoS component has two main functions: cloud service control and QoS notifications.

a. Cloud service control

The QoS component controls the behavior of a cloud service element based on QoS profiles (Section III.B). In fact, each cloud service possesses a set of QoS profiles that include criteria QoS values (conception, current and threshold). The QoSComponent performs the following functions by means of the QoSC controller.

- Update current values of the QoS criteria: the QoS component interrogates a "Monitoring module" in order to have current values of the QoS metrics. Then, it evaluates and updates these values in the "real time profile". The Monitoring module is an entity that gives different metrics others than QoS metrics. This module is provided as a service (MaaS: Monitoring as a Service) and integrated in each cloud component. The description of this module is out of the scope of the present paper.
- QoS degradation: the QoS component has to detect degradation behavior of the cloud service. In fact, there are many causes that can bring to QoS degradation such as network congestion, increasing processing time, etc.
- Processing a new request: as the cloud service element can be shared by several users in the same time, new users' requests can be received. This component uses the QoS profiles in order to make an accurate decision about the possibility to treat a new request.

b. QoS notification (In/Out Contract)

The QoSComponent notifies permanently if the service retains his behavior during the run-time. It sends to the Cloud system an "In contract" message if the intended comportment is maintained (Figure 3). The second possible notification is an "Out contract" message. This notification indicates that the CSC does not maintain the correct behavior. This notification helps to prevent the occurrence of anomalies or failures.

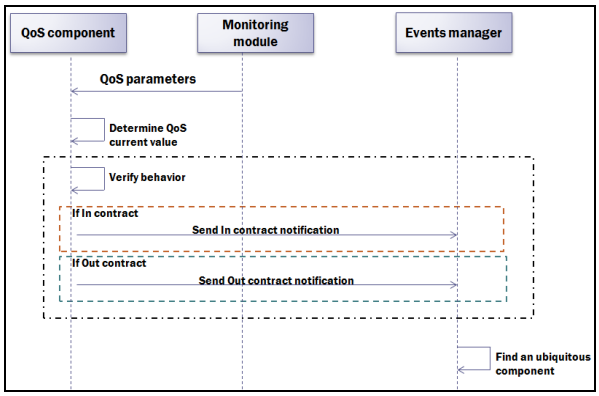

Figure 3. QoSComponent notifications

If the events manager receives an Out contract. One solution may be performed is to replace the degraded CSC component by a ubiquitous one. In fact, two CSC components are ubiquitous if they have the same function

 ISBN: 978-1-61208-257-8

and an equivalent QoS level. Many others reactions can be performed by the events manager. However, the management reactions will be treated in our future work.

VI. SPECIFICATION AND USE CASE

In this Section, we describe the CSC specification with the Document Type Definition (DTD) grammar.

A. *Specification of the extended DTD*

To describe distributed cloud component in accordance with our models we need to specify new formal notions with a DTD grammar. To do this, we use the basic DTD grammar of the ADL fractal language. We propose an extension of this grammar in order to depict the proposed notions such as QoS component, QoSC controller, etc. All implementations presented in this paper have been made by using Fractal ADL plug-in [15] in the eclipse IDE (Integrated Development Environment) [16]. This Section shows the extended DTD grammar that describes the notions corresponding to our model.

The basic DTD notions that define the native Fractal component specification are the following:

- component: can be primitive or composite
- interface: is the point of access to the component
- binding: it allows components communication
- content: represents the content of the component
- attributes: are described by a name/value pair. They are used to (re)configure the component
- controller: represents the membrane of a component.

As it is shown in Fig. 2, an Autonomic Cloud Service component is essentially composed of three parts: the interfaces, the primitive fractal component and the QoS component. Based on the basic DTD description, we add new notions to describe our proposed component. These notions are the following:

- CompositeComponent: represents our proposed component (the Cloud Service Component).
- PrimitiveComponent: represents a non-composite Fractal component.
- QoSComponent: manages the non-functional aspects of the cloud component (Section IV).
- interface-QoSC: is the QoS notifications controller.

The remainder of this Section describes examples of the extended DTD grammar. We describe the Cloud Service Component with the extended DTD grammar as follows:

```
<!ELEMENT CompositeComponent (PrimitiveComponent+, QoSComponent+, NC+, BC+, AC+, LC+, CC+, interface-QoSC+)>
<!ATTLIST CompositeComponent
  name CDATA #REQUIRED >
```

The DTD of the QoSComponent is described through three notions: QoSCriteria and QoSParameter. The needed controllers of this element are: NC, BC, AC, LC. A QoSComponent has a name and a role (client, server).

```
<!ELEMENT QoSComponent (QoSCriteria+, QoSParameter, NC+, BC+, AC+, LC+)>
<!ATTLIST QoSComponent
  name CDATA #REQUIRED
  role (client | server)>
```

As previously mentioned, the interface-QoSC is a new controller added to the CompositeComponent. Through this interface the QoS component communicates with the component to send management messages: In/Out contract. The QoS component can have different roles (passive, active, proactive and inter-active).

```
<!ELEMENT interface-QoSC (#PCDATA)>
<!ATTLIST interface-QoSC
  name CDATA #REQUIRED
  role (passive | active | proactive | inter-active)
  signature CDATA #REQUIRED >
```

Four QoS criteria (Availability, Delay, Capacity, Reliability) define the type of parameters to be measured. Our model defines three values types of these criteria: (Conception | Threshold | Current). The DTD specification of the QoSCriteria is as follows:

```
<!ELEMENT QoSCriteria (#PCDATA)>
<!ATTLIST QoSCriteria
  Criteriatype (Availability | Delay | Capacity | Reliability)
   ValueType CDATA #REQUIRED
  roleValueType (Conception | Threshold | Current)>
```

B. *Use case description*

In order to show a proof of concept of our propositions, we describe the example indicated in Section II. The goal is to apply our model to describe a simple distributed application in the context of cloud computing. The use case in Fig.4 represents a requested PaaS service defined by three software components: Apache, Jonas and MySQL. Each component is installed in a different VM (ApacheVM, JonasVM and MySQLVM). These VMs are interconnected throughout two links: AJP and JDBC (Figure 4).

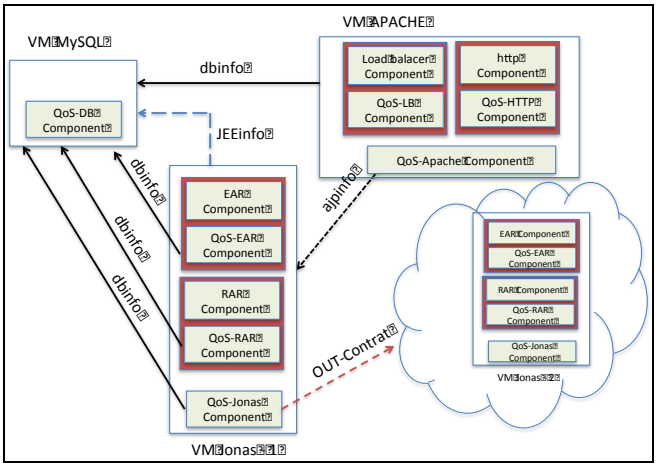

Figure 4. Use Case: Springoo

These components are in their turn composed of others sub-components. The Apache Server is composed of two sub-components: Load Balanced and HTTP Server. The JEE applications are deployed in each Jonas component. They are

composed by EAR and RAR components. According to our model each component is described by its functional and non-functional (QoS-component) aspects. For example each EAR component is described by the EAR component (functional part) and the associated QoS-component (non-functional part). The QoS component manages the behavior of each component by controlling the QoS criteria. We also suppose two ubiquitous Jonas components located in different cloud environments (VM Jonas-1 and VM Jones-2).

We propose the following two scenarios to show the QoS-awareness and self-management of the Jonas component:

- The Jonas component receives more requests than it is able to treat (peak load). In this case, the QoS component notifies that the current capacity is exceeded and sends an Out contract notification. Then, the Jonas component rejects the request.
- If the Jonas component is not available, the QoSC interface sends an Out contract. To deal with this, a ubiquitous Jonas component (VM Jones-2) is demanded to replace the failed one. We do not describe in this paper the mechanism how to choose the new component, it is out of the scope of this paper.

VII. CONCLUSION AND FUTURE WORK

Cloud computing is a new paradigm that provides on-demand services over the Internet. Cloud services are viewed as a composition of distributed components. These components have multiple types: infrastructure (hardware, storage, network), platform or software. On-demand, flexibility and availability of computing components are behind the great success of cloud computing. However, the QoS requirements of a cloud user are still not guaranteed.

To ensure the QoS requirements, the cloud services must be able to adapt its behavior dynamically. In this paper, we considered two important points to reflect the complexity introduced by the QoS cloud management: the self-management and the QoS awareness. We present the mapping of our generic QoS model to deal with these two aspects. Two principal propositions are presented:

- an integration of our QoS model to conceive a new QoS-aware cloud component (CSC).
- an extension of a DTD basic grammar in order to describe our QoS model specification. This DTD is used to describe the CSC component through the Fractal ADL language.

We are based on our generic QoS model to propose an autonomic cloud component that is able to manage its non-functional aspects. We use our informational model to maintain the QoS self-management aspects. We present a QoS component, which is integrated in each cloud component. This component uses the QoS criteria to control the current behavior of the CSC component and to inform the system about the current component's state ("IN/OUT contract").

The present work represents the first step to study how the self-management of a QoS-aware cloud component behavior is achieved. In the further work, we will present some mechanisms to maintain the autonomic loop principles in our proposed component.

ACKNOWLEDGMENT

This work is supported by the OpenCloudware project. OpenCloudware is funded by the French FSN (Fonds national pour la Société Numérique), and is supported by Pôles Minalogic, Systematic and SCS.

REFERENCES

[1] ETSI TR 102 805-3 V1.1.1 (2010-04). Part 3: QoS informational structure. On-line report: http://www.etsi.org/deliver/etsi_tr/102800_102899/10280503/01.01.01_60/tr_10280503v010101p.pdf [retrieved: February, 2013]

[2] http://www.opencloudware.org/bin/view/About/ProjectInfo [retrieved: January, 2013].

[3] ETSI TR 102 805-1 V1.1.1, Part 1: User's E2E QoS – Analysis of the NGN," On-line report : http://www.etsi.org/deliver/etsi_tr/102800_102899/10280501/01.01.01_60/tr_10280501v010101p.pdf, [retrieved: February, 2013].

[4] N. Simoni, C. Yin, and G. Du Chén, "An intelligent user centric middleware for NGN: Infosphere and ambient grid," in Proc. Communication Systems Software and Middleware and Workshops, COMSWARE 2008, pp. 599-606.

[5] G. Diaz, K. Chen, N. Simoni, and N. Ornelas, "Spécifications des composants fonctionnelles de la session UBIS," May 2010, on-line report: http://www-l2ti.univ-paris13.fr/~ubis/UBIS-SITE/FichiersPDF/SP31.pdf [retrieved:February, 2013].

[6] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, "Qos–aware clouds," in Proc. Cloud Computing, CLOUD 2010, pp. 321–328.

[7] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds", in Proc. European Conference on Computer Systems, EUROSYS 2010, pp. 237–250.

[8] H. Nguyen Van, F. Dang Tran, and J.M Menaud, "Autonomic virtual resource management for service hosting platforms," in Proc. Workshop on Software Engineering Challenges of Cloud Computing, ICSE 2009, pp. 1–8.

[9] L. Romano, D. De Mari, Z. Jerzak, and C. Fetzer, "A novel approach to qos monitoring in the cloud," in Proc. Data Compression Communications and Processing, CCP 2011, pp. 45-51.

[10] F.Zambonelli, N.Bicocchi, G.Cabri, L.Leonardi, and M.Puvianil, "On Self-adaptation, Self-expression, and Self-awareness in Autonomic Service Component Ensembles," in Proc. Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2011, pp. 108-113.

[11] H.Liu, M.Parashar, and S. Hariri "A component based programming model for autonomic applications," in Proc. International Conference on Autonomic Computing, ICAC 2004, pp. 10-17.

[12] I. Brandic, "Towards self-manageable cloud services," in Proc. Computer Software and Applications Conference, COMPSAC 2009, Vol 2, pp. 128–133.

[13] M. Maurer, I. Brandic, V. Emeakaroha, and S. Dustdar, "Towards knowledge management in self-adaptable clouds," in Proc. Services, SERVICES 2010, pp. 527–534.

[14] R. Ranjan and R. Buyya, Special section on Autonomic cloud computing: technologies, services, and applications, in Proc. Concurrency and Computation: Practice and Experience, 2012, Vol 24, pp. 935-1034.

[15] http://fractal.ow2.org/f4e/, [retrieved: January, 2013].

[16] http://onjava.com/pub/a/onjava/2002/12/11/eclipse.htm, [retrieved: January, 2013].

 ISBN: 978-1-61208-257-8