QoS-based Autonomic Service Component for Service Delivery

Houda Alaoui Soulimani, Philippe Coude
Department of Architecture and R&D
Société Française de Radiotéléphonie (SFR)
Paris-France
{houda.alaouisoulimani, philippe.coude}@sfr.com

Noëmie Simoni
Department INFRES
TELECOM ParisTech - LTCI - UMR 5141 CNRS
Paris - France
simoni@telecom-paristech.fr

***Abstract* - Nowadays, with the rising complexity of the service personalization in a heterogeneous and mobile context and the need to satisfy the End-to-End QoS, the service resources should be taken into account as a prominent resource as well as the network resources. Therefore, a high degree of self-sufficiency, self-management and automation is required in the service resource "service component" to enhance the service delivery. In this paper, we propose an autonomic service component "ASC" based on an integrated QoS–agent that self-controls and self-manages the service resources to dynamically adapt the service resources in response to changing situations during the user's session. To explain our proposal, we detail the mechanisms used to provision and monitor the service resource and to verify its conformity to the QoS contract established between the service providers and the customers during the exploitation. The issue of the ASC self-control and self-management is addressed according to the functional and non-functional (QoS) requirements in order to ensure the service continuity during the service delivery.**

Keywords - Autonomic Service Component; End-to-End QoS; QoS-agent; Mobility; Service Delivery.

I. INTRODUCTION

In the last years, with the fast evolution of the new generation networks and services (NGN/NGS), the user wants to access his personalized services while switching between different terminals or access networks. All these types of spatial mobility must be executed without impacting the End-to-End QoS. However, with the increasing demands on service delivery from customers, Providers are faced with the challenge of provisioning and managing their service resources in an efficient, cost-effective and flexible way. Currently, there are a set of mechanisms used to provision and monitor the network resources which caters to the specific QoS requirements of the applications at the transport network. These solutions permit to ensure and maintain the data delivery which guarantees the QoS of the transport network during the media session. These mechanisms do not take into account the QoS of the service resources knowing that there are some problems that arise during the user's session because of the services behavior in the platforms. Nowadays, QoS solutions which operate only on the network resources are no longer sufficient because the delivered QoS to a given client may be affected by many factors including the performance of the service component, the hosting platform or the underlying transport network. This is the reason why we have thought of broadening the QoS control and management in the service resource "service component" during the user's session. In fact, to guarantee the user satisfaction with regards to the service delivery in a heterogeneous and mobile context, we should provision and monitor the service resources as well as the network resources, and dynamically re-provision the service resources by benefiting from the ubiquitous services offered by different providers in the service platforms. To summarize, providing a service to the users guaranteeing the End-to-End QoS requires a horizontal QoS management at the service layer which is added to the existing QoS management at the access and transport layers.

However, the Service Oriented Architecture (SOA) [1] plays a principal role in allowing the creation of applications as a composition of independent service components offered by different providers. In addition, many available service components provide identical functionalities albeit with different quality of service capabilities. The following questions arise: How to rethink the service component to include the QoS control and management at the service level during the user's session? How a service component can enhance the service delivery? How to cover all the user's preferences and requirements (functional and non-functional QoS) and ensure a better performance in a heterogeneous and mobile context?

The purpose of this paper is to highlight the service component features that intend to decentralize and automate the QoS control and management for a flexible services composition with a better QoS performance. The benefit of this solution is to conceive a service component able to dynamically and autonomously react in real time to a change in the QoS contract during the user's session, such as availability. That's the reason why we propose in this paper an autonomic service component "ASC" based on a QoS-agent to monitor the QoS in real time during the processing. This QoS-agent triggers an event in the ASC environment in the case of a deterioration of the QoS. The receipt of an event activates the necessary mechanisms to change the current ASC used by another equivalent service component having the same functionality and QoS in order to maintain the QoS at the service level of the architecture and consequently to guarantee the service delivery.

The remainder of this paper is organized as follows: In Section II, we discuss some of the works related to the topic of this article. Section III details our proposal by explaining the service component features and mechanisms that are used to self-control and self-manage the QoS during the user's session. In Section IV, we present a scenario illustrating a utilization case of the ASC. In Section V, we evaluate the performance of the ASC through an implementation. Section VI concludes the paper.

II. RELATED WORK

In recent years, many researchers have focused their efforts on the service composition, the autonomic service component and the autonomic service architecture (ASA) especially in terms of providing the QoS to consumers in a dynamic environment.

Farha et al. [2] presented in their paper a generic Autonomic Service Architecture (ASA) to deliver applications and services over an all-IP infrastructure. Their solution is based around the concepts of SOA, virtualization and service delivery. The ASA proposes a generic framework to deal with the activation, provisioning, management and termination of network resources in an autonomic way. The ASA acts especially on the network resources to deliver a given service to customers. Cheng et al. [3] presented an approach to the autonomic service architecture (ASA) by proposing a framework for the automated management of internet services and their underlying network resources. This framework ensures the service delivery at the transport layer. The drawback of these approaches lies in the fact that they act on the network resources and do not cover the service resources which are essential to the service delivery.

Zhang et al. [4] proposed a framework to identify QoS problems in the SOA when a business process fails to deliver the quality of service. Their proposal is based on a set-covering algorithm which is used to select the locations of run-time service data collection or probes. The framework creates a dependency matrix to denote the relationships between the data recorded by probes and the service status. A diagnosis is then used to identify potential faulty services. Zhai et al. [5] presented a framework to repair failed services by replacing them with new services and ensuring that the new service process still meets the user specified QoS constraints. The drawback of these approaches lies in the fact that they present centralized solutions to identify failed services during a session. In opposition to the previous approaches, our proposition is based on a distributed self-management and self-control of the QoS to dynamically maintain the service session without impacting the end-to-end QoS.

Zambonelli et al. [6] proposed autonomic service components that are able to dynamically adapt their behavior in response to changing situations. They present mechanisms to enable the components to self-express the most suitable adaptation. The components acquire the proper degree of self-awareness to put into action the self-adaptation and the self-expression schemes. Liu and Parashar [7] presented a framework which enables the development of autonomic elements and the formulation of autonomic applications to have a dynamic composition of autonomic elements. They propose rules and mechanisms for the dynamic composition of autonomic components so that the computational behavior of the elements as well as their compositions and interactions can be managed at the runtime using dynamically injected rules. These approaches focus on the autonomic application creation paradigms and the behavior of the service component adaptation. However, these current solutions focus on self-adaptation and self-awareness of the service component and don't take into account the dynamic aspect of the session at the service level. In opposition to these approaches, we propose a distributed self-control and self-management of the QoS in each service component that is based on overcoming QoS violations without modifying any QoS parameter in the SLA. The QoS-based autonomic service component aims to enhance the service delivery in a dynamic, mobile and heterogeneous context.

III. PROPOSITION

In order to satisfy the SLA contract established between the customers and the providers, a more flexible and adapted service composition based QoS is desired to enhance the service delivery during the session mobility. Its principle task is to integrate the QoS control and management at the service level of the architecture. To do so, we should automate and distribute the QoS control and management at each service component which is involved in the delivery of a given service. That's the reason why we propose in this paper an ASC to fulfill such purpose. To explain our proposal, we firstly detail in section A the architectural and the functional aspects of the ASC. Secondly, we explain in section B our QoS model which is the basis of the service component self-management. Finally, we explain in section C the mechanism applied by the ASC during each operational step to self-manage and self-control its own resources for a dynamic reaction and adaptation during the user's session in order to maintain the service delivery with the required QoS.

A. Autonomic Service Component

Our proposal is based on two approaches: the SOA (Service Oriented Architecture (SOA) approach and the EDA (Event Driven Architecture) approach. The benefit of the SOA is the possibility to implement decentralized applications in distributed computing systems. One of the most important advantages of the SOA is its capacity to enable the rapid composition of the service components offered by various providers. The EDA complements the SOA because the service components can be activated by triggers fired on incoming events. To make the dynamic composition more effective in a heterogeneous and mobile context during the operations, we propose a novel view of the SOA (Service Oriented Architectures) based on an autonomic service component "ASC" that is generic, stateless, shareable, autonomous and self-manageable. The ASC is stateless because it performs the same processing (operations) for all the requests coming from different users without storing the state or the data related to each request. This is important in the situation where it should make a dynamic replacement of a service component in the case of a deterioration of the QoS during the user's session. The ASC is shareable because it is designed for the provisioning and the processing of several users' requests according to its capacities. In order to control and manage this resources sharing, we associate a queue in the usage plan for each service component which holds all the accepted users requests. The Service component is autonomic because it is both autonomous and self-manageable. The service component is autonomous because it is functionally independent, i.e., it is self-sufficient and does not need other service components to achieve its functionality. The aim of the functional independence between the various components of service is to facilitate the change of a service component by a ubiquitous one in the case of a QoS deterioration or a malfunctioning during the processing without any impact on the global service requested by the end-user. The service component is self-manageable because it monitors its own QoS and manages its states related to the use of resources during the user's session. The service component will have at any time "t", one of the following four states: *Unavailable*, *Available*, *Activable* and *Activated*. The state *Unavailable* means that the service component is temporary or permanently inaccessible. The state *Available* means that the service component is or can be accessible. The state *Activable* means that the service component is ready to be activated, and the state *Activated* means that the resource is being used. When the service component is activated, a QoS-agent inserted in its management plan, monitors and controls the QoS contract (In contract/Out contract) and communicates the service component resources state via notification events. We mention that the QoS agent is based on two functional elements: the IQM (Internal QoS Manager) and the EQM (External QoS Manager). The IQM is in charge of the control and the management of each service component QoS. It monitors the internal resources of the service component to determine the QoS contract state (In contract / Out contract) of the ASC during the exploitation phase. The main function of the EQM is the communication and the coordination of the QoS resources between the different service components. [8]

During the user's session, a QoS agent can have one of the four following roles: Passive, Active, Interactive or Proactive. The QoS agent has a passive role when it ensures the internal processing of the QoS and does not communicate with its environment. It has an active role when it notifies the QoS resources status of the service component (In Contract/Out Contract). It has an interactive role when it interacts with other QoS-agents to negotiate the QoS parameters. The service component has a proactive role when the QoS-agent has the knowledge and the rules that enable it to make decisions on its own and send notifications to its environment. To do so, a service component instantiates a QoS model allowing a real time management of the service resources and their possible deteriorations.

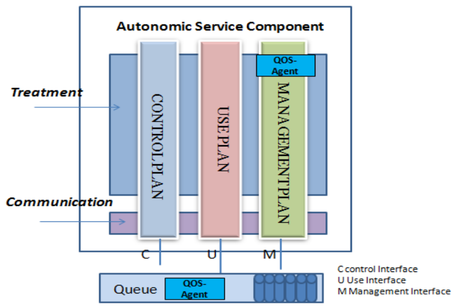

Figure 1: Structure of the Autonomic Service Component

B. *QoS Model*

In order to maintain the user Service Level Agreement (SLA), it is necessary to have a homogenous expression of the service component QoS to evaluate the End-to-End behavior. This is the reason why we propose a QoS model which represents the QoS in a uniform and homogeneous manner. The behavior of each component is reflected by measurable QoS parameters that can be categorized according to a vector of four criteria: *Availability, Reliability, Delay*, and *Capacity*.

Availability "$\mathcal{A}$": represents the ratio of accessibility for a service component. It indicates the number of times that the service component was accessible.

$$\mathcal{A} = 1 - \mathcal{U} / \mathcal{T}$$

$\mathcal{U}$ represents the number of times a service component has rejected a request because it was not available.

$\mathcal{T}$ represents the total number of requests sent over a period of time.

Reliability "$\mathcal{R}$": represents the ability of a service component to be executed without impacting the information. It indicates the percentage of successful invocations for a given period of measurement.

$$\mathcal{R} = 1 - \mathcal{F} / \mathcal{T}$$

$\mathcal{F}$ indicates the number of failed requests over a period of time.

Delay: represents the average time to process a request by a service component.

Capacity: represents the average number of requests processed by a service component during a unit of time.

The QoS criteria are necessary and sufficient for the self-control and self-management of the service component. These criteria are evaluated through three types of measurable values: *design values*, *current values* and *threshold values*.

Design values set the maximum processing capacity of a service component. *Current values* are used during the operation phase to monitor the behavior of the service component during the processing. *Threshold values* indicate the capacity limit that should not be exceeded by the service component in order to insure a normal treatment of the requests.

The QoS information helps to support the management, the treatment and the decision–making process of the service component. Hence, to make the right decisions during the user's session at the right time and the right place, it is necessary to have an efficient representation of the real world. Therefore, we need to have a uniform information structure containing both the description of the information of the service component as well as the knowledge of the behavioral aspects of the service component (such as the QoS). The informational model contains the different profiles that are going to be solicited during the different operations phases. For the provisioning phase, we have the resource profile which contains the QoS *design values*. During the consumption phase, the "Real Time Profile" is instantiated in real time to have a dynamic management of the QoS. It contains the *QoS current values* that will be compared with the *Threshold values* to allow the control of the service component behavior during the usage.

C. *QoS Management Mechanisms*

When the user demands a global service (GS) with an SLA that can be composed as a business process invoking a variety of available ASCs, it is necessary to apply a number of mechanisms to perform the service delivery during the user's session steps. During the pre-provisioning step, each ASC proceeds by a *QoS Admission Control* "*QAC*" to verify the service capacity required to treat a new user's request. The *QAC* allows the selection of the ASCs that suit the user's QoS requirements because of the ubiquitous characteristics of the ASCs offered by different providers. During the provisioning step, each ASC reserves the necessary resources to process the user's demand. Each ASC associates a queue (Figure 2) in the usage plan to hold all the accepted users requests because of its shareable characteristic. In order to control this resource sharing, we must apply a "*QAC*" to accept a user's query in the queue. The role of the *QAC* is to determine whether a new requestor can be accepted in the ASC queue, without violating the SLAs of the already accepted requestors. Finally, the QoS should be dynamically and continuously managed during the consumption step. Our self-management vision is based on the QoS-agent which is integrated in the management plan of each ASC to monitor the QoS related to the processing of the query as well as the QoS related to the ASC queue. We have already proposed ubiquitous services and queues communities in order to ensure a replacement of the ASC during the service delivery when a QoS agent detects a deterioration of the QoS. In fact, each ASC belongs to a Virtual Service Community (VSC) [9], which contains a set of ASCs having the same functionality and an equivalent QoS, each ASC's queue also belongs to a Virtual Queue Community (VQC) [10], which contains ASCs queues of equivalent QoS.

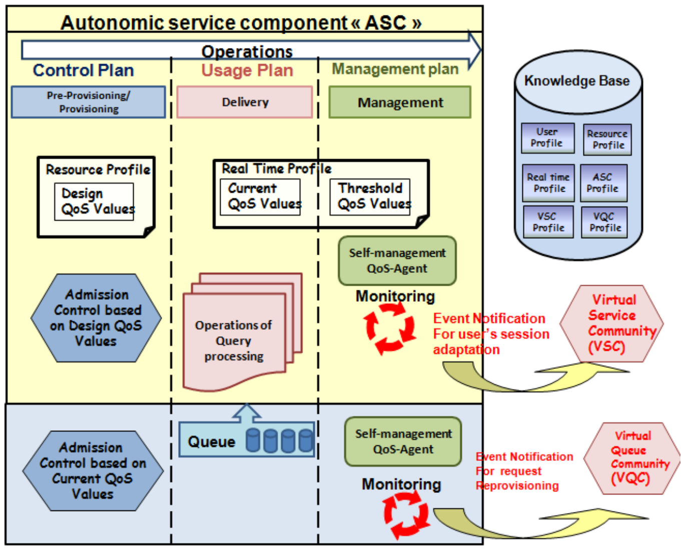
Autonomic service component « ASC »
Operations
Control Plan
Usage Plan
Management plan
Pre-Provisioning/ Provisioning
Delivery
Management
Knowledge Base
User Profile
Resource Profile
Real time Profile
ASC Profile
VSC Profile
VQC Profile
Resource Profile
Design QoS Values
Real Time Profile
Current QoS Values
Threshold QoS Values
Self-management QoS-Agent
Admission Control based on Design QoS Values
Operations of Query processing
Monitoring
Event Notification For user's session adaptation
Virtual Service Community (VSC)
Queue
Self-management QoS-Agent
Admission Control based on Current QoS Values
Monitoring
Event Notification For request Reprovisioning
Virtual Queue Community (VQC)

Figure 2: QoS Management Mechanisms applied by the ASC

The ASC is structured on three plans: the control, the usage and the management plan. The usage plan includes the main functions performed by the service component in order to process the requests. It includes all the mechanisms to process a request during the consumption phase. The control plan includes all the mechanisms used synchronously during the user's session to provision the service component resources. The management plan contains the self-management functions applied asynchronously on the data to control the service component resources. We explain in the following via an automaton the QoS management mechanisms used by the ASC in each plan during the different operational phases: the pre-provisioning, the provisioning, the delivery and the management phase.

During the pre-provisioning phase, the ASC receives in the control plan a signaling message to respond to a new user's request (Figure 3). First, it increments the "Attach" variable to "Attach+1" to keep the traceability of the users attached to a service component. Then, the QoS-agent applies a QAC mechanism to verify the statistical capabilities of its ASC. The statistical capabilities are based on the QoS *design values*. This admission control permits to determine if a component can be attached to a new user session to process its request. If the result is positive "OK", then the ASC changes its state to *Activable* in the case where it has not been previously attached to another user's session. Afterwards, it adds the user's session identifier to its profile in the knowledge base. If the result is negative "NOK", it should decrement the "Attach variable" to "Attach -1" and then the system searches in the VSC for another ASC having the same functionalities and an equivalent QoS to respond to the user's request.

During the provisioning step, the ASC receives the user's transaction in the control plan, it then increments the variable J to "J+1". Afterwards, the QoS-agent verifies the dynamic QoS based on the *QoS current values* of the queue in order to control the admission of the user's transaction in the queue. If the result is positive, it provisions the query in the queue and it changes its state from *Activable* to *Activated* if there is no query in the queue (i.e., J=1), else it decrements the variable J to"J-1".

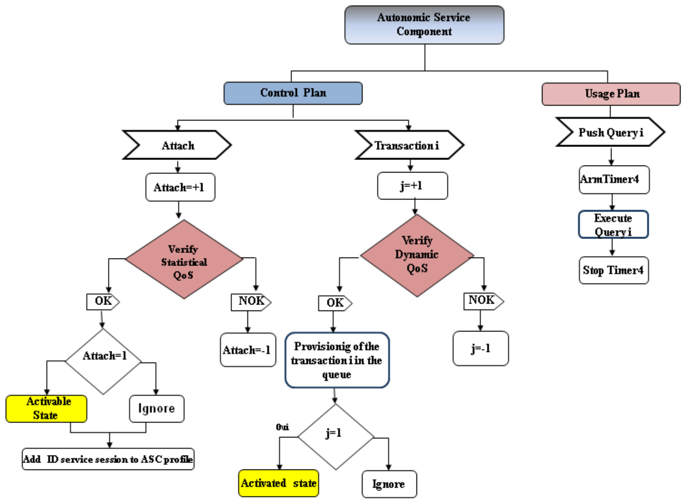

Figure 3 : Automaton of the ASC: Control and Usage Plans

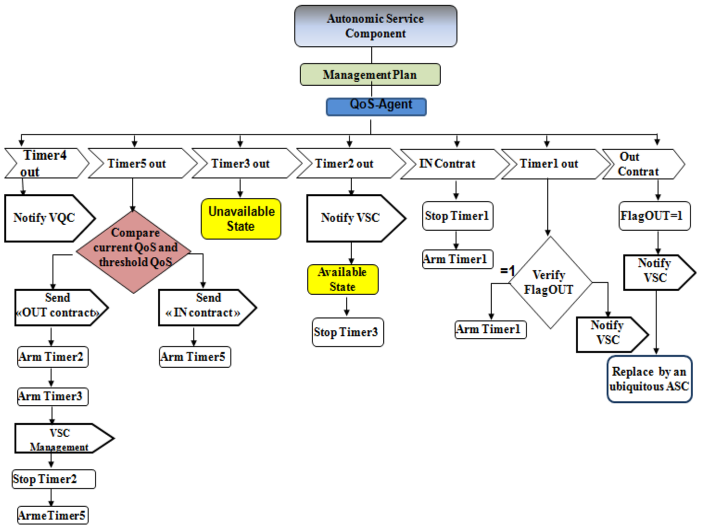

Figure 4: Automaton of the ASC: Management Plan

During the consumption step, an ASC that is in use through a user's session may not continue to function normally or to fulfill the user's QoS requirements. This is the reason why we solve such problem by proposing a number of mechanisms to monitor and manage the QoS at each ASC participating in the service delivery to the users. Our concept is based on a QoS agent integrated in the management plan of each service component. Figure 4 shows the different mechanisms used during the consumption step to maintain the service with the required QoS level. If the ASC takes a longer time to process a request (Timer 4 expired), the processing time of the requests that are provisioned in the ASC queue could be affected. The QoS-agent monitors the current QoS of the ASC queue. When it detects a QoS contract violation, it should notify the VQC to refer the request to another ubiquitous service component queue.

During the management step, The QoS–agent of each ASC composing the global service compares in real time the current QoS with the range of the QoS *threshold values*. If the result is positive, the QoS-agent sends a notification event "IN contract" to the VSC community to convey that it still respects the QoS contract. Otherwise, the QoS agent sends a notification event "Out contract" (Arm Timer 2 and Timer 3) to the VSC community in order to replace the current service component by a ubiquitous service (Stop Timer 2). At the end of Timer 3, the ASC changes its state from *available* to *unavailable* and leaves its community. Then, the ASC joins a new VSC community according to its current QoS and consequently changes its state from *unavailable* to *available*. All these operations are transparent to the end user and are done automatically by the system during the user's session.

Each ASC participates to the management of its VSC community; it shall notify regularly the other members on its QoS contract state (In Contract / Out Contract). If an ASC receives an Out contract, it sets its FlagOUT to 1 and notifies the VSC community to exclude the ASC that is out of the contract and to proceed to its replacement in order to maintain the VSC community. If the ASC receives an IN Contract, it will stop the Timer1 which has been armed in anticipation of receiving an IN Contract. In the case where Timer 1 expires (Timer 1 out) without receiving an IN Contract; the ASC should check first its FlagOut. If FlagOut=1, this means that the ASC has already received an Out Contract. Otherwise, it notifies the VSC about the defective communication of the ASC.

IV. SCENARIO

In this section, we use a case study to demonstrate the utility of the ASC during the user's session. The following scenario illustrates the case where the user wants to personalize his services to search for a property to buy in his geographical area. When the customer wants to search for a property, he selects two services from his catalog to obtain the required customized service. The first one is a global service named Search Property "SP" with an SLA. SP consists of three ASCs: FIND, LOCATION and GET. The second service is Google Maps.

The course of the scenario events is as follows: When the user is moving, he will first use "FIND" to search for properties according to his geographical position (Longitude, Latitude). The search is performed in the user's ambient zone

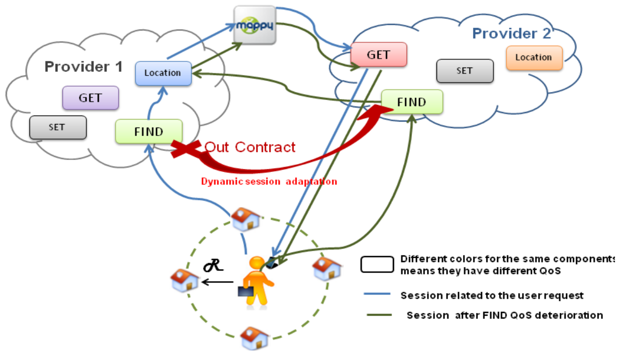
mappy
Provider 1
Location
GET
SET
FIND
Provider 2
GET
SET
Location
FIND
Out Contract
Dynamic session adaptation
Different colors for the same components means they have different QoS
Session related to the user request
Session after FIND QoS deterioration

Figure 5: Scenario

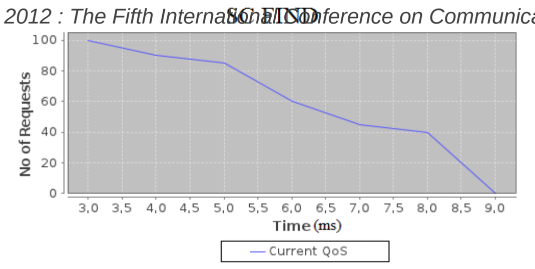

Figure 6: Basic Service component FIND

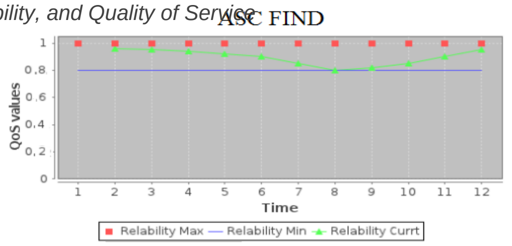

Figure 7: Autonomic Service Component FIND

which is limited by a radius R. Then, "Location" is used to obtain the GPS coordinates (Longitude, Latitude) for each property. Afterwards, he uses Google Maps to view the property on a map and finally he uses "GET" to see a detailed profile of each property (type, price and area). During the exploitation, the performance of "FIND" deteriorates and cannot provide the required QoS by the global service "Search Property". Therefore "FIND" sends an event notification to all the members of its VSC community to process its change. There are always some ubiquitous candidates to replace this component "FIND" and the selection algorithm used in the VSC community is essentially based on the user's geolocation in order to ensure the equivalent previous response time during the session. Once the VSC community finds a functional and QoS equivalent "FIND" on another platform, the connection is transferred from FIND@provider1 to FIND@provider2 (dynamic provisioning). Finally, the user's service session is maintained and the service delivery is performed. We should mention that all these operations are transparent to the user and are done dynamically and automatically by the system during the user's session.

V. IMPLEMENTATION & PERFORMANCE

In order to validate our proposal, we have used the language JAVA to develop our ASCs as independent EJBs and the JMS 1.1 queues to ensure communication between the ASCs. We have also used Oracle (V 10g) for the knowledge base and the JFreeChart API to plot the performance results.

In order to test the performance of our proposal, we have run the scenario that has been previously described with a basic service component (BSC) in one case and an ASC in the other. First case: we overload the BSC"FIND" and we notice that the data is lost because its QoS has deteriorated (Figure 6) and consequently the session is interrupted. Second case: we use ASCs with the following QoS: Find (QoS α): A_1= 0,99, R_1= 0,98, D_1=300ms, C_1=10; Location(QoS β): A_2=0,97, R_2=0,86, D_2=500ms, C_2=8; Get (QoS µ): A_4=0,98, R_4=0,90, D_4= 450ms, C_4=9. We note that if one of the four QoS criteria degrades during the user's session, it must proceed to change the service component because all the QoS criteria (A, R, D, and C) are necessary to maintain the overall QoS of the service component. During the user's session, the QoS-agent of the ASC "FIND" detects a QoS deterioration at the reliability criteria (R_1min = 0, 8 at T=8ms) (Figure 7). It invokes the VSC to change it by an equivalent ASC"FIND". We notice that the performance of the QoS is better thanks to the QoS-agent because it prohibits a complete QoS deterioration of the service component and maintains the service delivery.

VI. CONCLUSION

In order to enhance the service delivery, we have proposed in this paper an efficient solution based on an autonomic service component "ASC" which plays a key role in offering a service according to the consumer's requirements in a heterogeneous and mobile context. The ASC contains a QoS-agent to dynamically self-control and self-manage its own resources during the processing. Up to now, we have proposed a number of mechanisms to seamlessly adapt the user's session against any QoS deterioration caused during the operations. This solution is useful to resolve the problems of contemporary architectures that require more real time dynamicity, flexibility and responsiveness. Through experimentation, we have demonstrated that this proposal provides a better performance of the QoS and therefore a better service delivery to the consumers. In the future, we will try to find the best solution for the selection algorithm used in the virtual service community (VSC) and the virtual queue community (VQC).

ACKNOWLEDGMENT

The authors would like to thank all the participants in the UBIS project which is financed by the French ANR VERSO 2008.

REFERENCES

[1] "Portal for SOA and WS " www.service-architecture.com, April 2012.

[2] R. Farha, A. Leon-Garcia,"Blueprint for an Autonomic Service Architecture," Proc. IEEE. Autonomic and Autonomous Systems (ICAS 2006), Silicon Valley, CA, July. 2006, pp. 16-16.

[3] Y. Cheng, R. Farha, S.K. Myung, A. Leon-Garcia, and J.W.-K. Hong, "A Generic Architecture for Autonomic Service and Network Management," Computer Communications, November 2006, pp. 3691-3709.

[4] J. Zhang, Y. Chang, and K. Lin, "A Dependency Matrix Based Framework for QoS Diagnosis in SOA," Proc. IEEE. Service-Oriented Computing and Applications (SOCA 2009), Taipei, Taiwan , January 2009, pp. 1-8.

[5] Y. Zhai, J. Zhang, and K. Lin, "SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints," Proc. IEEE. International Conference on Web Services (ICWS 2009), Los Angeles, CA, July 2009, pp. 815-822.

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

[7] H. liu, and M. Parashar, "A component based programming framework for autonomic applications, " Systems, vol.36, May. 2006, pp. 341-352, doi:10.1109/TSMCC.2006.871577.

[8] H. Alaoui, P. Coude, and N. Simoni, "Modèle organisationnel pour le pilotage dynamique de la qualité de service de bout en bout d'une session user-centric," Gestion des réseaux et de services (Gres 2010), Montreal, Canada, Oct 2010, unpublished.

[9] H. Alaoui, N. Simoni, P. Coude, "User-centric and QoS-based service session," proc. IEEE. Asia-Pacific Services Computing Conference (APSCC 2011), Jeju, South Korea, Dec 2011, pp. 267-274.

[10] S Kessal, N Simoni, " QoS based Service Provisioning in NGN/NGS context," proc.IEEE. Network and Service Management (CNSM 2011), Paris, France, Oct 2011, pp.1-5.