# End-To-End QoS-Aware Middleware Services

GIORGIA LODI*

Dept. of Computing Science, University of Bologna, 2002

**Abstract**

This paper describes the principal services that an Application Service Provider (ASP) must develop in order to support the execution of distributed applications with QoS and trust requirements to be respected. The article will present an approach to building an architecture that is QoS-aware and adaptable to the possible changes that can occur inside the system.

This work is supported by the European Union Project IST-2001-34069 called TAPAS, Trusted and QoS-Aware Provision of Application Services.

## 1 Introduction

In recent years service companies, that provide on a contractual basis, rentable or "pas-as-you-use" access to centrally managed applications are rising very quickly. We can define these service firms as APPLICATION SERVICE PROVIDERS (ASPs) that hold the promise of making available application hosting facilities on remotely managed servers to multiple users over Internet or other networks.

In the last years the users have changed their needs and requirements about the services provided by the Internet, becoming increasingly fickle and anything that impacts on service quality, even failed components, will affect their levels of satisfaction.

Hence, different notions of Quality of Service (QoS) have been defined in literature: "A set of quality requirements on the collective behaviour of one or more objects" [1], or, more precisely, "A set of quantitative and qualitative characteristics of a distributed system necessary to achieve the required functionality of an application" [2].

Bearing in mind these considerations, the Application Service Providers mentioned above, to work effectively, must guarantee the respect of certain Quality of Service parameters, security, provide resilience and SERVICE LEVEL AGREEMENTS (SLAs) [4] over commonly available infrastructures. Moreover, ASPs need to ensure that hosted applications are able to access to a wide range of

*Researcher in TAPAS project at the University of Bologna, email: lodig@tin.it

services irrespective of the underlying platforms or the organisations through which they are provided.

Currently ASPs lack tools and techniques for offering hosting facilities for advanced Internet based applications.

Taking into consideration these limits, our main goal is to develop methods, tools, algorithms and protocols that support the construction and provisioning of Internet application services with particular QoS requirements to be provided.

In order to achieve this principal objective, *QoS enabled middleware services*, that will enable components to be deployed and interact (in a secure way) across organisational boundaries, must be developed.

The original motivation for introducing such middleware platforms, has been to facilitate just the development of distributed applications by providing a collection of general-purpose facilities to the application designers.

Currently, commercially available middleware platforms, such as those based on CORBA are still limited to the support of best-effort Quality of Service (QoS) to applications. This constitutes an obstacle to the use of middleware systems in QoS critical applications, or in case services are offered in the scope of Service Level Agreements with strict QoS constraints [3].

Moreover, we should implement a middleware platform which is capable of supporting lots of different types of applications with different QoS requirements, which is capable of interacting and making use of different kinds of support subsystems (operating systems, communication resources), utilized to verify the availability of the resources in the overall system, and which is capable of adapting its behaviour to the changes in both the application environments and the available resources inside the system itself.

To achieve these goals, a middleware archictecture is being designed as par of the European Union Project TAPAS. The next section will describe the core QoS-aware services necessary to construct that architecture; Section 3 summarizes the architecture tiers wea re designing.

## 2 QoS-aware services

As briefly discussed above, ASPs provide resilience and Service Level Agreements (SLAs). The SLA is a legal contract that specifies deliverables, terms and conditions between service providers and service customers. It also identifies the services provided as well as the supported products, measurement criteria, reporting criteria, and quality standard of the service [4].

In this contract, every specification of QoS such as availability, performance and scalability characteristics of the components of the application level, as well as trust relationships have to be expressed.

In this way, the SLA becomes the point of start from which deriving the end-to-end QoS application requirements.

These QoS constraints can be grouped in a sort of *QoS Contract*, at the application level, which in turn will be used as a input of certain services designed to make the overall architecture QoS-aware.

This QoS contract is derived from the SLA specifications and represents the low level QoS requirements that have to be fulfilled in order to support the execution of the distributed applications.

We believe that, independently of the type of application, some basic QoS services are necessary to provide end-to-end QoS.

For this reason, we have identified three principal services:

- *Configuration Service*
- *Run Time Support Service*
- *Trust Management Service*

We discuss below the structuring of the first two services (the third service will not be explained in this paper), we divide them in other parts, each of them with a certain task to carry out.

The Configuration Service (CS) is, in general, responsible for the management of the resources necessary to support the execution of the distributed application based on the QoS parameters specified in the QoS contract mentioned before. The principal responsabilities of the Configuration Service include:

- *Resource discovery*
- *QoS Negotiation*
- *Resource reservation*

Resource discovery is responsible for looking for the resources inside the overall system. With the term resources we mean not only physical resources but even processes necessary to carry out the execution of the distributed applications respecting the QoS constraints.

When applications request support for their deployment and they terminate their execution and when quality of service requirements change, it is necessary a mechanism to negotiate quality levels and resource allocations with the underlying system or other applications. This negotation is supported by the QoS Negotiation part. This implies that, given a certain QoS required by the application and given a certain QoS which can be offered by the middleware platform, the negotiation will individualize an agreed QoS, that is, another sort of contract which is, in this case, established between the application and the middleware responsible for supporting the execution of the application itself.

Given this contract, the next step to do is represented by the Resource Reservation which is responsible for the reservation of the resources identified in the previous phase.

In order to provide QoS-aware services, it is important to take into consideration another feature a middleware should have, that is, to be adaptive. Infact, the surrounding environment may change its operating characteristics to respond effectively to the new combination of situational factors.

Adaptation might be caused by changes in the required output quality of service levels, in the input quality of service levels, in external demands for resources (due to new request for service or change in resource requirements), in system-wide resource reallocation [5].

Hence, it is important to have services capable of monitoring, at run time, the QoS levels inside the system and, eventually, adapting these levels in accordance with the changes occured.

For these reasons we have created a service, the Run Time Support Service (RTSS) which is invoked, at run time, by both the Configuration Service and the application.

Its principal responsabilities are the following:

- *QoS monitoring*
- *QoS adaptation*
- *Resource release (termination)*

QoS Monitoring can be carried out by a QoS Monitor, always active, which is used to check the QoS levels in the system. For example, if the output quality achieved falls below the level required by the user, then the monitor will keep going on with the QoS adaptation operation. This operation leads to alert the QoS Negotiation service which will produce another adapted QoS level that has to be renegotiated.

The last operation, Resource Release, is responsible for managing the termination of the overall QoS management process with the release of the resources involved (e.g. advanced garbage collector).

We are designing a general architecture, named *TAPAS architecture*, which will be summariezed in details in the next section.

# 3 General Architecture

State-of-the-art application services are developed using component-based technologies, such as those provided by the Java 2 Enterprise Edition (J2EE), Microsoft's .NET or the Object Management Group's CORBA Component Model.

These component oriented middleware promote the use of *containers* to host component instances.

A container is a runtime environment which provides the system-level services to the components inside it. In this way, it is responsible for using the underlying middleware services for communication persistence, transactions, database management, security and so forth.

To make the underlying middleware QoS-aware, the services previously discussed, must be added to the basic services already provided by this runtime environment (container).

In our system model we have two important tiers: The application tiers and, that we have called TAPAS tier. It is in this layer that all the services described

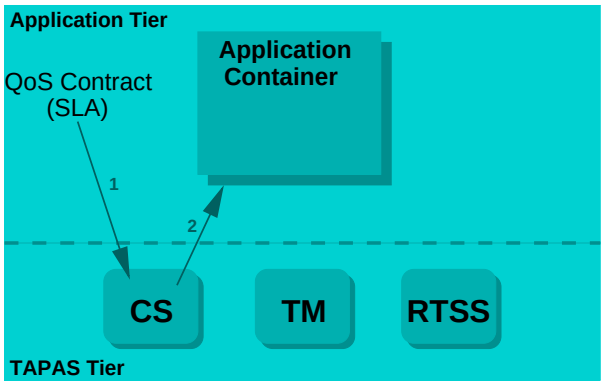


Figure 1: TAPAS Architecture Tiers

in the previous section, Configuration Service (CS), Run Time Support Service (RTSS) and Trust Management Service (TM) are located and they interact with each other to carry out the QoS-support process (Figure 1).

As shown in Figure 1, at the application level, we find the QoS contract derived from the SLA specifications. It is used as input for the Configuration Service that, after its execution, will produce a QoS level result used in turn by the application container.

This application container is the run time environment for the application components. This means that it hosts all the parts in which the application is composed and it uses the underlying services to support the execution of the distributed application.

# 4 Conclusions

The future work regards the design and the development of a component inside the system capable of producing a QoS contract from a SLA specification, that is, the input for the Configuration Service of our architecture.

Moreover, the services mentioned above, that is, every component of the Configuration Service, Run Time Support Service, have to be designed in details.

The implementation of the overall architecture will consist in extending some component-based technology, such as Java 2 Enterprise Edition (J2EE) or Jboss, integrating the basic services provided already (communication persistence, transactions, security and so forth) with our QoS-aware services.

As regards the application level, the architecture will be constructed using a particularly demanding (in terms of QoS requirements) applications such as real time distributed auction and multiplayer interactive distributed games.

# References


[1] "*Reference Model for Open Distributed Processing - Part 3: Prescriptive Model*" International Standard 10746 - 3, ITU -T Recommendation X.903, ITU -ISO, Geneva, 1995.

[2] Vogel A., Kerhervé B., Von Bochmann G. and Gecsei K. "*Distributed Multimedia QoS: A Survey*". IEEE Multimedia 2(2): 10-19, 1995.

[3] L.Bergamans, A.van Halteren, L.Ferreira Pires, M.van Sinderen and M.Aksit "*A QoS-Control Architecture for Object Middleware*", CTIT University of Twente, The Netherlands, 2000.

[4] W.Beckman and Marina Oleneva "*Application Hosting Requirements*" version 1.0 in work 27.05.2002, Adesso AG Dortmund.

[5] Mallikarjun Shankar, Miguel DeMiguel and Jane W.S. Liu *"An End-to-End QoS Management Architecture"* University of Illinois, Urbana-Champaign, USA. In *Proceedings of IEEE Woekshop on Middleware for Distributed Real-Time Systems and Services,* 1997.

[6] Giorgia Lodi "*End-to-end QoS Architectures"* Degree Thesis in Computer Science, University of Bologna 2001.

[7] *"Trusted and QoS-Aware Provision of Application Services (TAPAS): Description of Work"* 4 January 2002, project number IST-2001-34069, www.cs.ncl.ac.uk/research/projects/tapas/index.html