Delivering QoS in Open Distributed Systems

Frank Siqueira and Vinny Cahill
Distributed Systems Group, Department of Computer Science, Trinity College Dublin, Ireland
Frank.Siqueira@cs.tcd.ie, Vinny.Cahill@cs.tcd.ie

Abstract

This paper describes an architecture that provides support for quality of service (QoS) in open systems. The Quartz QoS architecture aims to avoid the dependency on specific platforms that limits the application of most existing QoS architectures in open, heterogeneous and distributed systems. Quartz adopts a flexible and extensible platform-independent design that allows its internal components to be rearranged dynamically, adapting the architecture to the surrounding environment in order to accommodate the differences among diverse computing platforms. Further significant problems found in other QoS architectures, such as the lack of transparency, flexibility and expressiveness in the specification of QoS requirements and limited support for resource adaptation, are also addressed by Quartz. This paper compares Quartz to other QoS architectures found in the literature, presents a prototype implementation of Quartz, and shows how Quartz is integrated with CORBA to form a framework for control and transfer of streaming media.

1. Introduction

In general, access to the resources provided by computational systems, such as network bandwidth, processing time and memory, follows a best-effort policy. The adoption of such policy results in unpredictable behaviour in the distribution of resources. However, there is an important category of applications that cannot tolerate uncertainty in relation to access to computational resources, demanding that the availability of the resources used by them be predictable. Such applications are said to have quality of service (QoS) requirements, and include application areas such as real-time systems and distributed multimedia.

Mechanisms for resource reservation are being added to computational systems in order to make access to computational resources behave in a predictable fashion. Nevertheless, most applications rely on distributed computing middleware, which is still being adapted to make use of resource reservation mechanisms.

Many different types of hardware, operating system and network infrastructures coexist in open systems, and multiple resource reservation protocols populate this environment. Consequently, allowing applications to reserve resources via a middleware layer implies that the differences between resource reservation protocols have to be handled by the middleware.

The term ‘QoS architecture’ is used to describe middleware that provides applications with mechanisms for QoS specification and enforcement. These architectures organise the resources provided by the system with the intent of fulfilling the QoS requirements imposed by the application. Substantial work on QoS architectures can be found in the literature (see [1] for a survey). However, the QoS architectures proposed so far consider only part of the overall problem of delivering QoS in open systems [2].

Our focus in the study of QoS architectures is on the provision of QoS-constrained services in open distributed systems. The QoS architectures proposed so far typically have a strong dependency on a particular computing platform and suit a specific application area. This approach prevents their use in open environments, where heterogeneity is always present, and limits the spectrum of applications that can make use of these architectures.

Other important problems can be identified in the QoS architectures presented in the literature. Some architectures constrain the expressiveness of the user in the specification of QoS requirements and lack transparency from the lower level, forcing the user to deal with a notion of QoS that is not familiar for him. Furthermore, in most architectures support for resource adaptation is very limited, if not completely absent.

This paper presents Quartz [3][4], a generic QoS architecture that addresses the limitations of previous proposals in this area. This is achieved by adopting a highly flexible, extensible, component-based platform-independent design, which allows user transparency from the underlying system and at the same time is suitable for open distributed systems. Since Quartz handles the QoS-related issues on behalf of the application, it allows the programmer to concentrate on the functional aspects of the application. In addition, Quartz increases the

portability of applications due to its capacity to handle the differences between multiple reservation protocols.

The remainder of this paper is organised as follows. Section 2 explains in detail the proposed QoS architecture, analyses its characteristics and compares Quartz to other QoS architectures proposed in the literature. Section 3 presents a prototype implementation of Quartz and explains how it is integrated into a CORBA-based environment for the provision of QoS-aware streaming media services. Finally, section 4 summarises our contribution and presents some conclusions.

2. The Quartz architecture

We have designed and implemented a prototype of a QoS architecture with the intent of solving the limitations of previous proposals in the area. The Quartz architecture is based on a highly flexible, extensible, and platform-independent design that allows it to be used in different application areas and in conjunction with a wide range of resource reservation protocols.

2.1. Requirements

The main goal considered in the development of Quartz was to provide support for heterogeneous environments. This implies that the architecture is required to be able to handle the different protocols and hardware that may coexist in an open, distributed and heterogeneous platform. Figure 1 illustrates the use of the Quartz QoS architecture in a heterogeneous environment.

Applications requiring QoS enforcement use the mechanisms provided by Quartz to specify their requirements. In order to provide the required QoS, Quartz employs the resource reservation protocols available in the target network and operating system.

In order to handle the heterogeneity present in open systems, the architecture is not only required to be able to be ported to different platforms, but it has also to able to handle QoS for an application when the lower-level reservation system changes without requiring a recompilation. For example, if the application is able to transfer data using both ATM and TCP/IP, the QoS architecture has to be able to perform QoS reservations for both protocols by adapting itself internally instead of requiring a new port of the architecture to be linked to the application. This level of flexibility is achieved by using an architectural design based on interchangeable components, in which components able to handle QoS for different reservation mechanisms can be plugged into the architecture dynamically. In addition, support for new reservation protocols can be added to the architecture without the necessity of porting the whole infrastructure. Instead, a new component that interacts with the new reservation protocol can be written by a programmer.

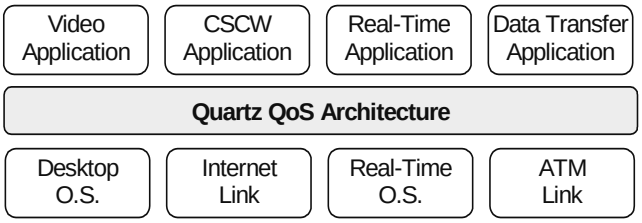

Figure 1. Quartz in a heterogeneous environment

Other problems arise from the necessity of providing a QoS architecture with such characteristics. These requirements can be subdivided in three areas: QoS specification, user transparency, and QoS adaptation.

QoS requirements must be specified according to the notion of QoS understood by the user at the higher level. Furthermore, since different notions of QoS might be present at the higher level because of the multiplicity of application areas that are targeted by Quartz, the architecture must be able to interpret a potentially infinite set of QoS parameters. The parameters specified by the user must be interpreted appropriately by the architecture in order to perform the reservation of resources at the lower level. This implies translating the parameters from their original format into parameters that are understood internally by Quartz. The architecture has also to map these parameters into resources available at the lower level and then interface with a suitable reservation protocol in order to allocate the corresponding resources for the application. This mapping process might not always be one-to-one (i.e. one QoS parameter resulting in a single resource being reserved) but it can be one-to-many, many-to-one or many-to-many. This implies that resources might be interchangeable, and that balancing requirements and resources is another task that has to be performed by the architecture.

In order to simplify the work of the application, the Quartz architecture must provide transparency of QoS and reservation mechanisms from the user's point of view. This implies that the interaction with the lower-level system, which is necessary to guarantee the level of QoS required by the application, must be performed by the architecture. Quartz must interpret the QoS requirements specified by the application and then use the available resource reservation protocols to guarantee the level of QoS requested by the application. By adopting this strategy, we hide from the application the differences between the way different underlying systems allow resources to be reserved in order to fulfil QoS requirements. This has the important effect of increasing the portability of applications across different platforms.

Quartz is also required to allow dynamic changes in the distribution of resources to be performed by the system. This must occur without causing loss of service consistency at application level. The reservation of resources might change dynamically because of dynamic

changes in the structure of the system, resource failure or the usage of priority-based policies for the distribution of resources. Any change in the reservation of resources at lower-level must be informed to the application using QoS parameters that are understood at high level. This implies that Quartz has to perform a reverse translation of parameters before informing the application that resource availability has changed. Since in some cases resources are interchangeable, dynamic changes may be overcome at the lower level by requesting additional resources from a different source. In this case, all the adaptation is performed at the architectural level, and the QoS seen by the application remains the same.

2.2. QoS specification and translation

A translation mechanism based on generic parameters is adopted by Quartz to address the problem of specification of QoS requirements in an environment composed of multiple application areas and different underlying reservation protocols.

In order avoid having a translator for each combination of application field and reservation protocol, we have adopted a three-step translation process.

Users specify their *application-specific QoS parameters*, which are first translated into a set of *generic application-level QoS parameters* defined by Quartz. These parameters are further translated into a set of *generic system-level QoS parameters* and balanced between the network and the operating system. Finally, generic system-level parameters are translated into the *system-specific QoS parameters* understood by each of the reservation protocols used by the application.

Since the application deals only with QoS parameters understood at his abstraction level and meaningful for his application field, his power of expression is not affected.

Table 1. Example of parameter translation

Parameter Set	Parameter Values
Application-specific Parameters	Audio::Quality = AUDIO_CD (i.e. 44KHz, 16 bits per sample)
Generic Application-level Parameters	App::DataUnitSize = 2 bytes; App::DataUnitRate = 44K units/s
Generic System-level Parameters	Net::Bandwidth = 88Kb/s
System-specific Parameters	RSVP::BucketSize = 88Kb/s; RSVP::TokenRate = 88Kb/s; …

Table 2. Example of parameter balancing

Parameter Set	Parameter Values	
Generic Application-level Parameters	App::EndToEndDelay = 50 ms (X)	
Generic System-level Parameters	Net::Delay = 30 ms (Y); OS::Delay = 20 ms (Z)	Note: X = Y + Z

Table 1 illustrates the transformation suffered by a parameter – in this case, the audio quality – at the different levels of the translation process until it reaches a format understood by the RSVP protocol [5]. Table 2 shows a parameter – in this example, the overall delay – which is balanced between the network and the operating system.

2.3. Architectural components

Each component defined by Quartz encapsulates a particular task in the overall problem of QoS specification and enforcement in an open, heterogeneous environment. These components can be easily replaced by new ones in order to adapt the architecture to the target environment.

The *QoS agent*, the major component of the Quartz architecture, is responsible for implementing the QoS mechanisms necessary for the provision of services with the quality requested by the user. This involves two main tasks: the translation of QoS parameters between different levels of abstraction, and the interaction with the resource reservation protocols present in the underlying system.

The QoS agent, as illustrated by Figure 2, is composed of a *translation unit* and multiple *system agents*.

The translation unit contains *QoS filters* and a *QoS interpreter*. QoS filters can be subdivided into application and system filters, and are responsible for translating between their respective set of specific QoS parameters and the generic set of parameters at the same abstraction level.

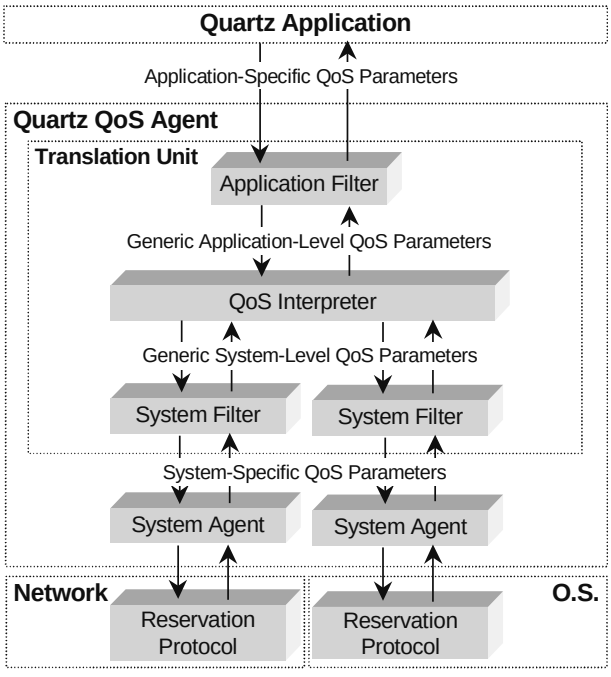

Figure 2. Detailed structure of the QoS agent

The QoS interpreter establishes the mapping between the two sets of generic parameters defined by Quartz. During this process, it also balances the usage of resources between the network and the operating system making use of a balancing algorithm.

Finally, the *system agents*, which are associated with the reservation protocols responsible for administering the use of the resources provided by the underlying system, get the values of QoS parameters provided by the translation unit and perform the necessary reservation of resources using the corresponding reservation protocol.

2.4. Achievements of the Quartz architecture

The Quartz architecture supports heterogeneity by encapsulating the QoS mechanisms necessary for interacting with a specific resource reservation protocol or application area into a replaceable component with a standardised interface. These components are plugged into the architecture whenever the associated protocol or application area is in use. As a result, the core of the architecture is highly portable, reusable and extensible because the particularities of application fields and resource reservation protocols are encapsulated by application and system filters, and system agents respectively. Changes at application level can be accommodated by replacing the application filter. Similarly, changes at system level imply the replacement of system filters and system agents. Filters and system agents may be selected from a component library provided by Quartz, or may be implemented by the application programmer.

In addition, the characteristics of the translation mechanism result in a compromise between the needs of different application fields regarding the manner in which QoS requirements are expressed and the generalisation necessary for the architecture to be deployed over heterogeneous platforms. In fact, the QoS interpreter, like the other components, can be extended to recognise new parameters and implement new balancing policies, or it can be even entirely replaced by a programmer in order to implement a whole new QoS specification mechanism.

Support for dynamic resource adaptation is also provided by Quartz. The QoS agent reports dynamic changes in QoS to the application as application-level QoS parameters by using the reverse translation path that is provided by the translation unit. During this process, a set of system-level QoS parameters passes through the translation unit and is translated into application-level QoS parameters. However, in some cases the resource adaptation can be accommodated at a low level without interfering with the QoS seen by the application. In the Quartz architecture, some QoS requirements such as cost and delay are fulfilled by the sum of resources provided by both the operating system and the network. The component responsible for dividing QoS requirements between the operating system and the network is the *balancing agent*, which is basically a resource trader that is encapsulated by the interpreter. When one of the operating system or the network reduces the amount of resources allocated for the application, the balancing agent execute a process called ‘rebalancing of resources’. This process tries to compensate for the loss of resources on one side by requesting more resources from the other. When this is possible, the resource adaptation occurs only at the lower level, and the quality seen by the application is not affected. In this case, the adaptation is completely transparent from the application’s point of view. If rebalancing at the lower level fails, Quartz tells the application to adapt its requirements in order to decrease the consumption of resources. This can be done for example by reducing the quality of a video stream or by changing the compression method used for data transfer.

2.5. Analysis of Quartz

The Quartz architecture is targeted at a wide range of network protocols and operating systems. In addition, Quartz provides support for QoS specification and enforcement for different applications areas. These characteristics distinguish Quartz from other QoS architectures.

Most of the QoS architectures found in the literature are concerned only with network protocols. In addition, he tight integration of architectures such as QoS-A [6] with the network (in this case, ATM) constrains the flexibility and limits the extensibility of the architecture. Quartz is able to cover a wider range of QoS requirements than QoS-A, and QoS can be specified at a higher level of abstraction.

Quanta [7] and ERDoS [8] adopt translation mechanisms similar to the one used by Quartz. However, Quartz allows QoS filters to be written by the application programmer, while Quanta translators, equivalent to our filters, are not accessible to the user. ERDoS incorporates the translation mechanisms to the resource and application agents, which can be written by the programmer as in Quartz. Nevertheless, due to the adoption of an approach to the provision of QoS based on monitoring and adaptation, QoS enforcement through resource reservation is not provided by ERDoS.

Proposals such as the QoS Broker in the Omega architecture [9] ‘require a huge amount of mapping and management knowledge to support large-scale distributed applications, and that service management through a single entity is too centralised and severely inflexible’ [10]. Quartz does not incur this problem, because QoS agents encapsulate only the support necessary for specification of QoS for the corresponding application field and for interaction with the reservation protocols

supported by the end-system, making the architecture lightweight and increasing scalability. Flexibility and extensibility are guaranteed by the use of replaceable components, instead of a monolithic structure such as the one adopted by the QoS Broker.

QuO [11] and Arcade [12] are examples of architectures that provide QoS languages for QoS specification. We could simulate the same functionality by implementing application filters that interpret the QoS languages used by QuO and Arcade and translate the QoS requirements expressed with these languages into the generic application-level QoS parameters used by Quartz.

The QuO architecture adds mechanisms for QoS specification to the client-server interaction model adopted by CORBA [13]. Based on this approach, QoS requirements are specified in relation to method invocations issued between CORBA objects. However, the client-server interaction model is known to be inappropriate for applications that use continuous media [14]. We believe that this approach is likely to impose limitations to the use of the QuO architecture.

Arcade is an example of a purely system-level QoS architecture based on a specific platform, in this case the Chorus kernel. Quartz could be easily made able to interact with this operating system by writing a system agent and a filter that interface with Chorus. In addition, Quartz allows the user to define network QoS constraints.

3. Validation, evaluation and application

We have developed a functional prototype of the Quartz architecture in order to analyse its behaviour while supporting applications with QoS requirements in a heterogeneous environment. In addition, we have integrated Quartz into a framework that provides QoS-aware transfer of streaming media over the network.

The prototype developed by us has system agents and filters for the RSVP protocol, for ATM networks, and for the real-time mechanisms provided by Windows NT©.

3.1. Validation and evaluation

We have implemented a few applications on top of Quartz in order to evaluate its behaviour [4]. One of these applications is a simple program that transfers data packets over the network. This application is able to use either TCP, UDP (including multicast) or ATM for transferring data. Quartz was used as a means for reserving resources for both network supports without adding complexity to the application. According to the network reservation protocol being used, a suitable pair of system agent and filter is plugged into the QoS agent. The RSVP agent and filter are used for TCP and UDP, while ATM requires a different filter and agent. At application level, we have implemented an application filter, the data packet filter, which interprets the QoS requirements used by applications performing data transfers.

The data transfer application allows the user to specify QoS requirements by providing values for packet size and packet rate as well as the service guarantee (i.e. best-effort, deterministic or unloaded) through a graphical interface. These values are passed to the QoS agent and then processed by the data packet filter, by the QoS interpreter, which translates and balances requirements, and by the corresponding system filters. Finally, the system agents reserve the necessary resources by interacting with the corresponding reservation protocols.

Independently from the reservation protocol used at the network level – i.e. RSVP or ATM – equivalent behaviour was observed from the application's point of view in regard to the provision of QoS. This shows that, by using Quartz, the reservation mechanism became transparent for the application despite the different characteristics of the lower-level reservation protocols.

3.2. The Quartz/CORBA framework

Quartz has also been integrated with a series of mechanisms provided by the CORBA architecture [13] in order to provide a complete framework for the deployment of applications with QoS constraints.

The Quartz/CORBA framework relies on the audio and video streaming mechanism recently standardised by the OMG [15] as the basis for media transfer between objects distributed over the network. The components defined by this standard are illustrated by Figure 3.

Virtual device objects abstract multimedia devices (i.e. cameras, speakers, etc) used by multimedia applications. A stream control object allows the user to control the media flow (i.e., start and stop it) and to add/remove parties to/from a multi-party connection. A stream endpoint transfers stream data through the network, getting data from and delivering data to virtual devices. Each stream endpoint has one or more media flows, which are abstracted as flow endpoint objects (flow producers and consumers) located at each end of the flow.

The data delivery is subject to QoS constraints described during the creation of the stream. The QoS parameters associated with a stream can be modified through the stream control object.

The entire process of QoS specification, translation and provision of QoS is delegated to the QoS agent. The QoS agent performs the reservation of resources available in the underlying system required to fulfil the QoS requirements specified by the application. In addition, the QoS agent keeps track of resource adaptation, starting QoS adaptation whenever needed. The use of Quartz in the framework makes applications more portable and simplifies significantly the job of the application programmer.

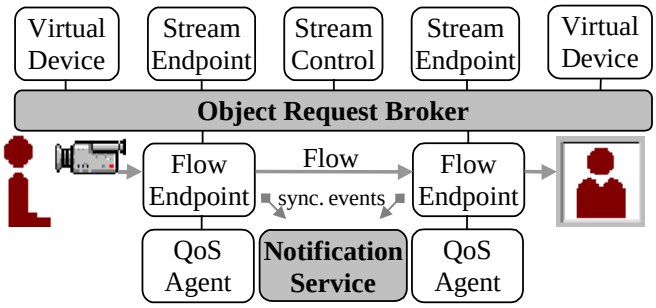

Figure 3. The Quartz/CORBA framework

For synchronisation of media we adopt the notification service [16], an extension to the CORBA event service that allows the specification of constraints on the latency of events propagated between objects.

Figure 3 shows an application scenario in which a video streaming application is built using the Quartz/CORBA framework. The application specifies the characteristics of the media stream, i.e. the number of flows of audio and video and their expected quality, obtained either from a user profile or through interaction with the user. The CORBA streaming mechanism handles the transmission of media between the source and the destination, and controls the flow of media and the membership of the connection. The QoS agent is informed of the QoS requirements associated to the media flows and handles them accordingly.

4. Conclusions

In this paper we have introduced a QoS architecture, called Quartz, that deals with QoS constraints present in distributed applications. Quartz makes the lower-level aspects of resource reservation transparent from the user's point of view, although allowing the necessary control through notification in the case of resource adaptation.

Quartz was designed to allow its use in heterogeneous platforms, enabling its integration into frameworks for the development of distributed computing applications with QoS requirements. The design of the Quartz architecture allows its easy extension to support new classes of applications, operating systems and network protocols by selecting components from a library or by adding new components written by the application programmer.

We have developed a prototype of Quartz that provides mechanisms for specification and enforcement of QoS to applications with QoS requirements. This prototype has been integrated into a complete framework for the deployment of applications with QoS constraints in CORBA-based systems.

We have also performed a complete validation and evaluation study of Quartz, which was described briefly in this paper. We have implemented a few applications with QoS requirements and used Quartz to enforce these requirements. This study showed that Quartz is able to fulfil the QoS requirements of diverse applications built in open, heterogeneous systems.

Acknowledgements

The authors would like to thank Iona Technologies (http://www.iona.com) and the Capes foundation (http://www.capes.gov.br) for supporting this project.

References

[1] C. Aurrecoechea, A. Campbell and L. Hauw "A Survey of Quality of Service Architectures", MPG Group, University of Lancaster, Tech. Report MPG-95-18, 1995.

[2] R. Steinmetz and L.C. Wolf "Quality of Service: Where are We?", IWQoS'97 Proceedings, May 1997.

[3] F. Siqueira and V. Cahill "Quartz: Supporting QoS-Constrained Services in Heterogeneous Environments", RTSS'98 Proceedings (work in progress), November 1998.

[4] F. Siqueira "Quartz: A QoS Architecture for Open Systems", Ph.D. Thesis (submitted), Trinity College, University of Dublin, October 1999.

[5] R. Braden et al., "Resource Reservation Protocol (RSVP)". IETF RFC 2205, September 1997.

[6] A. Campbell, G. Coulson and D. Hutchison, "A Quality of Service Architecture", ACM Computer Communications Review, Vol. 24(2), April 1994.

[7] S. Dharanikota and K. Maly, "Quanta: Quality of Service Architecture for Native TCP/IP over ATM Networks", HPDC'96 Proceedings, February 1996.

[8] S. Chateerjee, B. Sabata and J. Sydir "ERDoS QoS Architecture", SRI International, Tech. Report, May 1998.

[9] K. Nahrstedt, and J. M. Smith, "The QoS Broker", IEEE Multimedia, Vol. 2(1), Spring 1995.

[10] D. Waddington, C. Edwards and D. Hutchison, "Resource Management for Distributed Multimedia Applications", ECMAST'97 Proceedings, May 1997.

[11] J. Zinky, D. Bakken and R. Schantz "Architectural Support for Quality of Service for CORBA Objects", Theory and Practice of Object Systems, Vol. 3(1), January 1997.

[12] I. Demeure, J. Farhat and F. Gasperoni, "A Scheduling Framework for the Automatic Support of Temporal QoS Constraints", IWQoS'96 Proceedings, March 1996.

[13] Object Management Group, "The Common Object Request Broker: Architecture and Specification (Revision 2.0)", OMG Document PTC/96–03–04, March 1996.

[14] G. Coulson and D. Waddington "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", TreDS'96 Proceedings. Lecture Notes in Computer Science, Vol. 1161, Aachen, Germany, October 1996.

[15] Iona Technologies, Lucent Technologies and Siemens-Nixdorf, "Control and Management of Audio/Video Streams", OMG Document Telecom/97–05–07, July 1997.

[16] BEA Systems et al. "Notification Service", OMG Document Telecom/98-01-01, January 1998.