An Open QoS Architecture for CORBA Applications

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

Distributed application programmers rely on middleware such as CORBA in order to handle the complexity that arises from the distributed and heterogeneous nature of the underlying computing platform. CORBA, in particular, provides a media streaming mechanism that can be used for media streaming and for associating QoS requirements with media streams. Despite defining the interfaces of the media streaming mechanism, the corresponding specification does not prescribe how QoS is enforced at low-level by the middleware. This paper describes the design and implementation of a QoS architecture, called Quartz, which has been integrated with CORBA in order to provide a framework that allows applications to transfer real-time media in open systems. This framework is employed to model and simulate a pattern recognition mechanism for use in an automated manufacturing cell, which is also described and analysed in this paper.

1. Introduction

In the last decade we have observed the acceptance of distributed object computing as a technique that allows distributed applications to be developed for open, heterogeneous platforms and to interoperate with other applications through well-defined object-based interfaces. In order to be deployed in this context, a distributed application has to follow an object model defining how objects are structured and how they interact with each other.

Distributed object models are supported by middleware that provides the programmer with powerful tools for the development of distributed applications. This middleware hides from the programmer the complexity involved in network communication, handling of network addresses, conversion between data formats, and other issues arising from the distributed and heterogeneous nature of the computing environment.

Many proposals for middleware for distributed object computing can be found in the literature although few have a large market acceptance. The CORBA architecture [1], which was proposed in 1992 by the OMG, has established itself as the *de facto* standard in this area. The wide acceptance of CORBA is a result of its openness as a standard, of the suitability of its object model for heterogeneous systems, of the diversity of CORBA-compatible products that are provided by several vendors, and of the support given to this technology by the approximately 800 members of the Object Management Group (OMG).

Despite its adequacy for transmission of best-effort data, CORBA did not initially provide support for delivering real-time media (i.e., media that is associated with timing constraints) due to the quality of service (QoS) requirements that have to be fulfilled while delivering this category of data traffic. The immediate solution for the implementation of CORBA-compatible multimedia applications subject to QoS requirements would be the use of CORBA only for the transmission of best-effort data such as control information, while the media would have to be transferred using a lower-level network protocol with provision for QoS specification and enforcement. However, it is not reasonable to use high-level middleware to provide protocol and distribution transparency for best-effort data, and on the other hand expose the programmer to the lower-level protocols used for transmission of real-time media.

In order to make CORBA suitable for the transmission of real-time media, OMG has defined a media streaming mechanism that is integrated with the CORBA architecture [2]. The A/V streams mechanism, when used together with software that enables the administration of computational resources with the intent of fulfilling the QoS constraints imposed on the delivery of real-time media data, allows CORBA applications to transfer media data by using high-level CORBA-compatible abstractions.

However, the mechanisms necessary for QoS enforcement are not defined by the CORBA A/V streams specification. In order to address this deficiency, we have designed and implemented a QoS architecture, known as Quartz, that, when integrated with CORBA, can provide a complete framework for the development of distributed multimedia applications. Besides providing mechanisms for QoS enforcement, the integration of Quartz with

CORBA makes the middleware more portable, and easier to implement and maintain due to the separation between mechanisms for data transmission and QoS enforcement.

This paper is organised as follows. Section 2 introduces the area of distributed object computing, including CORBA and its media streaming mechanism. Section 3 introduces the Quartz architecture, which has been designed to provide mechanisms for specification and enforcement of QoS constraints in open distributed systems. Section 4 shows how Quartz has been integrated with the CORBA architecture in order to fulfil the requirements present in applications that handle continuous media. Section 5 describes an example of the use of CORBA and Quartz for pattern recognition in an automated manufacturing cell. Finally, Section 6 presents some conclusions.

2. Distributed object computing

In the last decade the development of software, which was typically based on the use of structured design and procedural languages, has rapidly migrated towards the adoption of object-oriented design and programming techniques.

With the widespread adoption of object-oriented design and programming techniques for application development, it was natural that this tendency extended to the area of distributed computing. The adoption of object-based approaches in distributed programming was reinforced by the availability of standards proposed by the OMG, a consortium created by software vendors, developers and end-users to promote object technology for the development of distributed computing systems.

The OMG has proposed a set of standards for distributed object computing, which are described by the Object Management Architecture (OMA) [3]. The OMA is a framework of standards and concepts for open systems, centred around the concept of the Object Request Broker (ORB). In this architecture, methods of remote objects can be invoked transparently in a distributed and heterogeneous environment through the ORB.

The OMG also specifies a series of standard services that complement the ORB, such as a naming service, a life cycle service, an event service, a transaction service and a security service, which are often necessary in a distributed environment.

2.1. CORBA

The *Common Object Request Broker Architecture* is the standard ORB defined by the OMG [1]. The CORBA specification establishes the roles of the components of the ORB and defines their interfaces. By introducing a common architecture, the OMG makes transparent for applications the differences between distinct CORBA implementations and lower-level systems.

In the CORBA environment, each object implementation has its interface specified in IDL (Interface Definition Language). Clients issue remote method invocations as if they were local object calls. These calls are transferred to the server object through the ORB, which acts like an "object bus", establishing a "network of objects".

A communication protocol called IIOP (Internet Inter-ORB Protocol) is used to allow interoperability between different CORBA implementations. In addition, the OMG has approved mappings of the IDL interfaces for the main programming languages in order to allow objects written in different languages to interoperate.

CORBA has become a *de facto* standard for distributed object computing. Several legacy applications in sensitive areas such as banking and telecommunications have been ported to CORBA platforms, and new applications have been built based on CORBA in order to obtain easy interoperability with other applications and to improve portability and reuse.

2.2. The CORBA streaming mechanism

The CORBA streaming mechanism [2] was proposed by the OMG in order to provide support for delivery of continuous media data in CORBA-based systems. This mechanism defines a group of abstractions to deal with stream data in multimedia systems. These abstractions are illustrated in Figure 1.

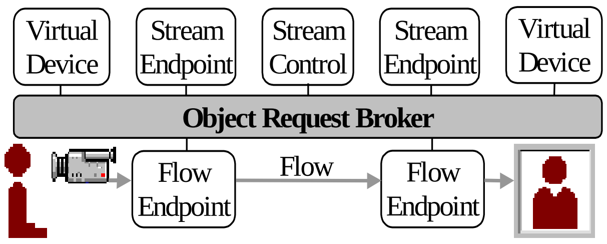


Figure 1. The CORBA streaming mechanism

Virtual device objects abstract multimedia devices (i.e. cameras, speakers, etc) used by a multimedia application. A stream control object allows the user to control media flows (i.e., start and stop them) as well as add/remove parties to/from multi-party connections.

Stream endpoints transfer stream data through the network, getting data from and delivering data to virtual devices. Each endpoint has one or more associated media flows, which are abstracted as flow endpoint objects located at each end of the flow. Flow endpoints are classified as producers and consumers according to the role performed by them in the transfer of data.

The data delivery is subject to QoS constraints described during the creation of the stream, i.e., during the binding of two or more virtual devices. A binding operation establishes a stream between two virtual multimedia devices, defining the QoS parameters associated to the stream. Two binding semantics are allowed. The simple local binding uses the virtual device and returns a stream control object. In the third party binding mechanism, a stream control object can bind two (or more) remote virtual devices and keep control over the data flow.

The QoS parameters associated with a stream (such as video resolution and the number of audio channels) can be specified through the stream control object, through the stream endpoint or through individual flow endpoints.

With the introduction of the stream mechanism, the CORBA architecture becomes suitable for describing applications that handle real-time media, and can be adopted as the common middleware for control and transfer of media. Nevertheless, despite allowing CORBA applications to handle real-time media, the CORBA streaming mechanism does not specify how QoS constraints imposed on the delivery of media data are enforced. In order to allow QoS constraints to be interpreted and enforced, we have proposed a framework that integrates a QoS architecture, called Quartz, into the CORBA architecture.

3. The Quartz architecture

Many distributed applications can function properly on currently available networking platforms and operating systems. However, there exists a category of applications that is not satisfied by the best-effort resource management policies provided by the majority of the computing platforms that are currently available. For these applications, the availability of resources provided by the underlying system is required to be predictable. These applications are said to have quality of service (QoS) requirements, and include applications varying from real-time control systems to distributed multimedia systems.

Applications that require a certain level of QoS must specify their requirements in a clear and accurate manner by using QoS parameters. The values of these parameters reflect the requirements imposed by the application, and can be stored in pre-defined user profiles containing the QoS constraints imposed on the behaviour of the application or can be obtained by the application through direct interaction with the user.

The achievement of the specified level of QoS is typically made possible through the reservation of the resources managed by the underlying system that are necessary to provide the network and operating system services used by the application with the requested level of quality. These resources include network bandwidth, processing time, physical memory, and access to multimedia hardware.

Several operating systems and network protocols incorporate mechanisms that allow applications to retain resources for their exclusive use. These mechanisms, called resource reservation protocols, are the key elements that support the provision of QoS guarantees. Unfortunately, most applications do not benefit from these mechanisms because the distributed computing middleware on which they rely is still being adapted to make use of such mechanisms. Furthermore, multiple resource reservation protocols coexist in open systems. Consequently, allowing applications to reserve resources via a middleware layer implies that the differences between reservation protocols have to be handled by the middleware.

Middleware components, usually referred to as QoS architectures, are responsible for providing mechanisms for specification and enforcement of QoS that make use of the resource reservation protocols provided by the underlying system. QoS architectures deal with issues such as the translation of QoS parameters comprehensible at the application-level into the parameters understood by the underlying reservation protocols that control access to the resources provided by the system. Without the services provided by a QoS architecture, these issues would have to be dealt with by the application.

Research on QoS architectures has resulted in several proposals that can be found in the literature (see [4] for a survey and [5] for an analysis of the open issues in this area of research). We have identified four main limitations that prevent the use of the existing QoS architectures in open systems. These limitations are:

- most existing QoS architectures require QoS to be specified using a low-level format that is not appropriate for applications with a more high-level notion of QoS, or use a format appropriate only for one specific area of application;
- QoS enforcement often occurs at either network or operating system level, instead of both, and in some cases the underlying system is not made completely transparent for the application, which still has to deal with low-level issues;
- the use of most existing architectures in open, heterogeneous systems is prevented due to their close integration with the underlying system; and
- most QoS architectures ignore the possibility of dynamic resource adaptation, which can occur due to factors such as resource failure or system reconfiguration.

A QoS architecture that addresses these limitations would, besides handling the complexity originating from the necessity of obtaining QoS-constrained services from the underlying platform on behalf of applications, make applications easier to implement and increase their portability between different underlying platforms. In addition, such a QoS architecture would allow application programmers to use the same high-level interface for specifying QoS requirements for applications inserted in different application contexts and making use of different networking infrastructures and operating systems.

Quartz is a generic architecture for the specification and enforcement of QoS that provides mechanisms necessary for building applications with QoS requirements in open systems.

3.1. Overview of Quartz

The Quartz QoS architecture [6] is composed of a QoS Agent running on top of the several resource reservation protocols available in the target system. QoS applications use the services provided by the QoS Agent to obtain the desired level of QoS.

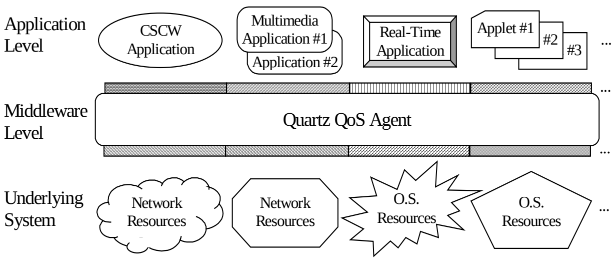


Figure 2. Overview of the Quartz architecture

Figure 2 situates the Quartz architecture in a computational system. At the application level, applications with QoS constraints (such as multimedia applications, real-time applications, etc.) interact with Quartz through interfaces that are specialised for the corresponding area of application.

The *QoS Agent*, the major component of the Quartz architecture, receives the QoS constraints imposed by applications, interprets them, and interacts with the underlying system in order to enforce these QoS constraints.

The underlying system contains resources that are used by the application. It is typically composed of several sub-systems, which can be grouped into two main areas: the network infrastructure (e.g., ATM or Internet protocols) and the operating system (e.g., a real-time or desktop operating system with reservation capabilities). The QoS Agent sees networks and operating systems as resource providers. Each resource provider allows its resources to be allocated through a resource reservation protocol. The QoS Agent must know how to interact with each of the resource reservation protocols present in the computing environment, using their interfaces in order to allocate resources for applications.

The QoS Agent is a placeholder into which other components can be plugged in order to interact with the surrounding environment. It is responsible for two main tasks:

- Interpreting the QoS requirements specified by the application in the form of QoS parameters, translating them from a format understood by the application into a format suitable for performing resource reservations; and
- Interacting with the underlying resource reservation mechanisms in order to allocate the necessary resources for performing the service subject to QoS requirements.

Being just a placeholder for other components, the QoS Agent does not perform these tasks directly. Instead, it relies on other components that are specialised for translation of QoS parameters and interaction with resource reservation protocols. Since the translation process depends directly on the application and on the resource being used, and the interaction with reservation protocols depends on the interface provided by this particular protocol, specialised components will be used in each case. These components will be plugged into the QoS Agent whenever necessary.

3.2. QoS specification and translation

In an open environment, different forms of expressing QoS are present in different abstraction levels and are used by distinct applications and platforms. A QoS architecture has to be able to interpret (i.e. translate and understand) these different QoS parameter formats to be used in multiple application fields and to enforce QoS using different resource reservation protocols.

In order to avoid having a translator for each combination of application field and reservation protocol, Quartz adopts a three-step translation mechanism. The QoS requirements of the application are specified in the form of *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.

Table 1 illustrates the transformation undergone by a parameter at the different levels of the translation process (in this case, video quality is translated into a set of ATM parameters). Table 2 illustrates the case of a parameter (in this example, the overall delay) that must be balanced between the network and the operating system.

Table 1. Example of parameter translation

Parameter Set	Parameter Values
Application-specific Parameters	`Video::Quality` = VHS (352x240 pixels; 24 bits/pixel; 30 frames/sec)
Generic Application-level Parameters	`App::DataUnitSize` = 247.5 Kb; `App::DataUnitRate` = 30 units/sec.
Generic System-level Parameters	`Net::Bandwidth` = 7.425 Mb/sec.
System-specific Parameters	`ATM::PeakCellRate` = 155 cells/sec.

Table 2. Example of parameter balancing

Parameter Set	Parameter Values
Generic Application-level Parameters.	`App::EndToEndDelay` = 500 msec.
Generic System-level Parameters	`Net::Delay` = 300 msec.; `OS::Delay` = 200 msec.

3.3. Internal structure of Quartz

The internal structure of the QoS Agent is illustrated in Figure 3.

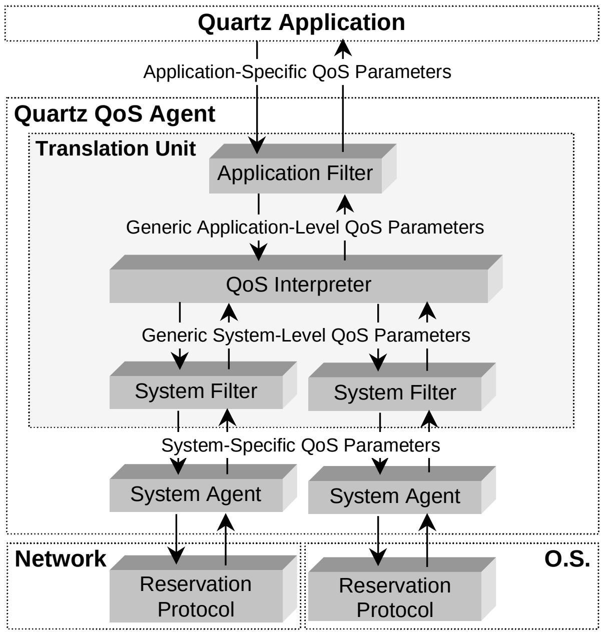


Figure 3. Detailed structure of the QoS agent

The translation unit performs the translation of QoS parameters between the different formats used at different levels in the architecture. In addition, the translation unit balances the usage of interchangeable resources provided by the different sub-systems (i.e., the network and the operating system) that are present in the underlying system. The translation unit contains *QoS Filters* and a *QoS Interpreter*. QoS Filters can be subdivided into *application* and *system filters*, one for each application field and for each sub-system present at the lower level, respectively. Filters are responsible for translating between QoS parameters represented internally by Quartz and any external format. The QoS Interpreter handles parameters formats used internally by Quartz at different abstraction levels and balances the use of resources provided by the underlying system.

The QoS Agent also encapsulates multiple *system agents*, which are responsible for interacting with the reservation protocols administering the use of the resources provided by the underlying system. The system-specific agents get the values of QoS parameters determined by the translation unit and perform the reservation of the resources provided by the corresponding sub-system using the associated reservation protocol.

3.4. Analysis of Quartz

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 architectural core is highly portable, reusable and extensible because the specifics of the application area and of the reservation protocols present in the underlying system are encapsulated by application filters and by system filters and system agents respectively. Replacing the application filter can accommodate changes at application level. 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 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 constraints are expressed and the generalisation necessary for the architecture to be deployed over heterogeneous platforms. In fact, the QoS interpreter, much like the other components, can be extended to recognise new QoS parameters and implement new balancing policies, or it can even be entirely replaced by the user in order to implement a whole new QoS specification mechanism.

Support for dynamic resource adaptation is also provided by Quartz. In the event of adaptation, Quartz tries to perform transparent resource adaptation at system level by rebalancing interchangeable resources. When adaptation at system level is not feasible, the application is notified and asked to adapt its requirements.

We have implemented a functional prototype of the Quartz architecture in order to analyse its behaviour while supporting applications with QoS requirements in a heterogeneous environment. This prototype has component agents and filters for the RSVP protocol (using PC-RSVP provided by Intel), for ATM Networks (using ForeSystem's ATM ForeRunner LE 155 Mbps PC cards and a Fore Systems ASX 100 switch), and for the real-time mechanisms provided by Windows NT©.

4. Integration with CORBA

Quartz has been integrated with the CORBA architecture in order to provide a complete framework for the deployment of applications with QoS constraints.

The Quartz/CORBA framework relies on the CORBA streaming mechanism, which was described in section Figure 4, for transferring continuous media data between CORBA objects distributed over the network. The data delivery is subject to QoS constraints described during the creation of the stream, i.e. during the binding of two or more virtual devices.

The process of QoS specification, translation and enforcement are delegated to the instances of the Quartz QoS Agent that are associated with each flow endpoint. The stream endpoint linking the partners has knowledge about the quality of the communication service that has to be provided in order to achieve the QoS specified by the application. All of this information is transferred to the QoS Agent, which interprets this information, balances requirements between components, and then passes these requirements to the corresponding system agents.

We have an in-house implementation of the A/V streams mechanism [7] that supports the light profile of the standard defined by the OMG. The support was written in C++ using Iona's Orbix 2.3. The implementation was carried out on top of Windows NT 4.0 using Microsoft's Visual Development Studio 5.0. The implementations of the TCP/IP and UDP/IP protocols (including IP multicast) provided by the operating system were used for the transmission of continuous media data. This implementation of the CORBA A/V streams mechanism was adapted in order to use Quartz for provision of QoS. This is done through the flow endpoint, which interacts with Quartz in order to specify the QoS requirements imposed by the application. By separating the support for QoS from the communication support (i.e., from the implementation of the A/V streams mechanism) we have made the middleware more portable and easier to implement, test and maintain.

The interaction between flow endpoints and Quartz comprises the following tasks:

- initialisation of Quartz, at which point the communication port and the process that will avail of the services on which QoS requirements will be imposed are identified;
- request for enforcement of QoS requirements during the establishment of a connection between flow endpoints; and
- receipt of upcalls from the QoS agent, which are issued in order to report QoS adaptation, errors or warnings.

By performing these tasks, flow endpoints make the use of Quartz completely transparent to applications, which continue to specify their QoS requirements through the QoS specification mechanisms that are provided by CORBA A/V streams.

The A/V streams filter imports parameters and values from the `AVStreams::QoS` class defined by the A/V streams specification and translates them into generic application QoS parameters. Reverse translation is also executed by the filter in case of QoS adaptation. The A/V streams QoS parameters are listed in Table 3.

Table 3. A/V streams QoS parameters

Parameter Name	Description
`A/V::AudioSampleSize`	Size of audio sample
`A/V::AudioSampleRate`	Number of audio samples
`A/V::AudioNumChannels`	Number of audio channels
`A/V::AudioQuantisation`	Quantisation (linear, u-law, ...)
`A/V::VideoFrameRate`	Number of video frames
`A/V::VideoColourDepth`	Number of bits per pixel
`A/V::VideoColourModel`	Color encoding (RGB, CMY, ...)
`A/V::VideoResolutionHorz`	Horizontal resolution
`A/V::VideoResolutionVert`	Vertical resolution

The A/V streams QoS parameters, after being translated by the A/V streams filter, are further translated by the Quartz translation unit and handed to the system agents corresponding to the reservation mechanisms present in the system. At this point, the corresponding system agents use the available resource reservation protocols in order to enforce QoS.

The use of Quartz for enforcing QoS on behalf of CORBA makes the middleware more portable and simplifies testing and maintenance due to the separation between QoS enforcement and actual data transfer.

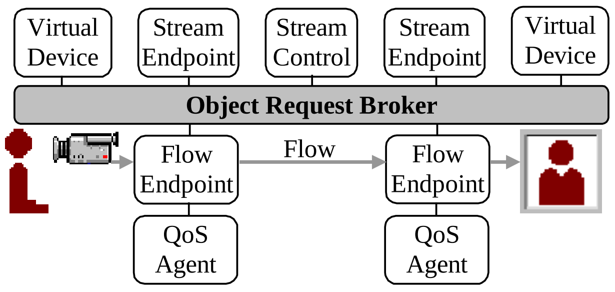


Figure 4. The Quartz/CORBA framework

By way of example, Figure 4 shows an application scenario in which a video conferencing application is implemented 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. The CORBA streaming mechanism handles the transmission of media between partners, and controls the flow of media and the membership of the conference. Quartz is informed of the QoS requirements associated to the media flows and handles them accordingly.

5. Application

We have used the Quartz/CORBA framework to simulate an automated manufacturing cell in which manufactured parts placed on a conveyor belt are classified by a robot. The classification process is done by analysing the image generated from a video camera using pattern recognition software that employs techniques based on neural networks.

In this scenario, which is illustrated by Figure 5, the image generated by the camera is transferred to a computer in the control room, which then instructs the robot to place parts in the correct storage bin or to discard parts in case they are found to be damaged. Due to the dynamics of the manufacturing process, the image has to be transferred through the network and analysed in a limited time so that the robot does not miss parts or discard them unnecessarily.

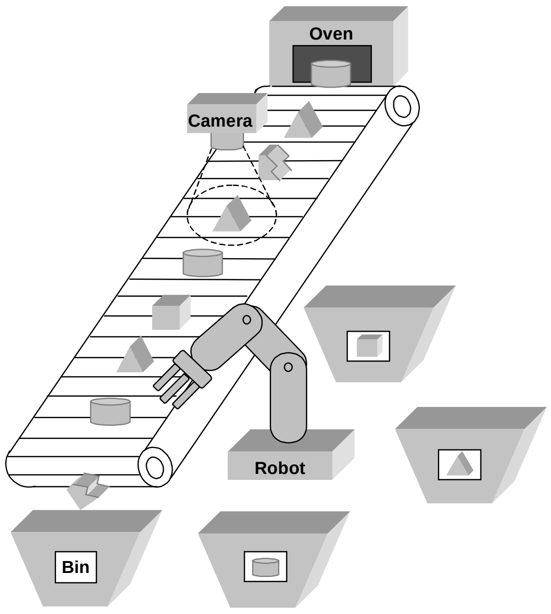


Figure 5. Automated manufacture cell

We can simulate this scenario in the context of the Quartz/CORBA Framework by imposing QoS constraints on a virtual camera device, which ensures that the video will be transferred by the camera in time for the pattern recognition process to occur.

Constraints are also imposed on the computer in the control room in order to ensure that the input device will be ready to receive the image, and that the image will be analysed in time by imposing constraints on the duration of the pattern recognition process. These QoS constraints are interpreted and enforced by the Quartz QoS Agent corresponding to the flow endpoints that are attached to each of the virtual devices. The software model that was just described is illustrated in Figure 6.

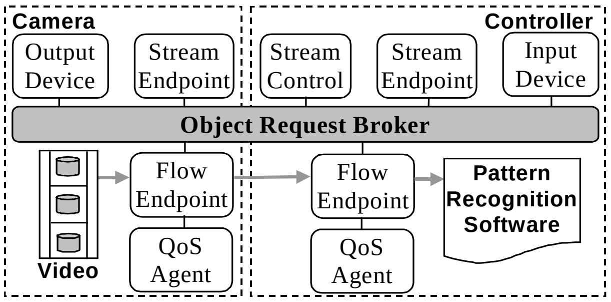
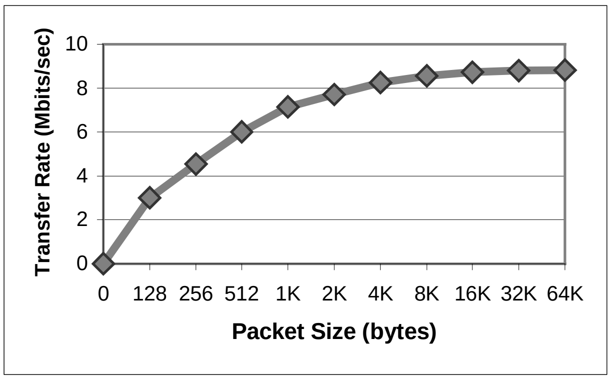


Figure 6. Software model of the pattern recognition mechanism



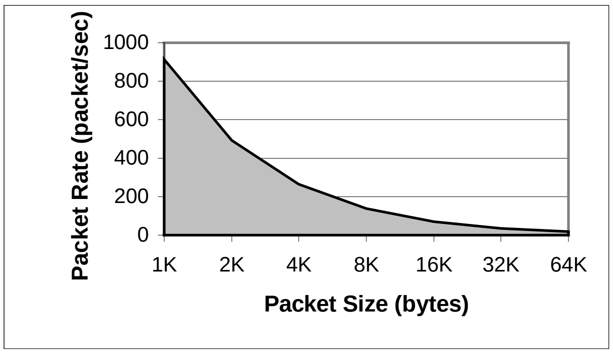


Figure 7.Transmission capacity of the A/V streams mechanism

Figure 7 shows the performance that can be obtained with our implementation of the CORBA streaming mechanism in terms of bandwidth per packet size. These performance measurements were obtained in a 10Mbps Ethernet local area network working with a light load. Similar performance can be guaranteed even under heavy load conditions by using Quartz and RSVP for performing resource reservations, provided that the frame size and frame rate of the video produced by the camera are within the area of operation displayed in the graph.

For the pattern recognition process, which is a very time-consuming process based on neural networks, we assume that it should not exceed a well-known deadline for each part being analysed with the processor working under unloaded conditions. Despite the fact that Windows NT does not provide deadline-based real-time scheduling algorithms, Quartz can guarantee unloaded behaviour by using the real-time scheduling class provided by the Windows NT scheduler.

This example shows how Quartz can be used together with CORBA in order to provide mechanisms for QoS enforcement for video transfer and real-time computing. Quartz can also provide support for QoS enforcement in different application areas and computing platforms by rearranging its internal structure.

6. Conclusions

In this paper we have presented a QoS architecture that, when integrated with CORBA, provides a complete framework for the transmission of real-time media data in open systems. In this environment, QoS requirements imposed on the flow of media data are enforced by Quartz, while CORBA is responsible for the transmission of control and media data.

The Quartz/CORBA framework has been implemented and employed to simulate a pattern recognition mechanism used in an automated manufacture cell. This application shows the usefulness and the efficiency of the framework as a tool for building CORBA-compatible applications that handle real-time media.

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] Object Management Group “CORBA 2.0 Specification”, OMG Technical Document 96-03-04, USA, March 1996.

[2] Iona Technologies, Lucent Technologies and Siemens-Nixdorf, “Control and Management of Audio/Video Streams”, OMG Doc. Telecom/97–05–07, July 1997.

[3] Object Management Group “The Object Management Architecture Guide”, OMG Technical Document 92-11-1, September 1992.

[4] 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.

[5] R. Steinmetz and L.C. Wolf “Quality of Service: Where are We?”, IWQoS’97 Proceedings, May 1997.

[6] F. Siqueira and V. Cahill “Quartz: A QoS Architecture for Open Systems”, Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (to appear), 2000.

[7] David O’Flanagan “An Implementation of the CORBA Audio/Video Streaming Service using RSVP”, M.Sc. Dissertation, University of Dublin – Trinity College, Department of Computer Science, Technical Report TCD-CS-1999-32, October 1998.