Remote Object Oriented Programming with Quality of Service or Java's RMI over ATM*

Yehuda Afek† Michael Merritt‡ Gideon Stupp†

November 22, 1996

An online version of this paper can be found at url
"http://theta.math.tau.ac.il/QoSrmi/mascot97/abstract/abstract.html"

Abstract

In this work we present a new simple powerful network programming interface that supports quality of service (QoS) requests and moderation. Our interface enhances well-known object oriented network interfaces (such as Java's RMI, and CORBA), thus keeping with the accepted standards.

We claim that in most cases there is a total order over the levels of quality for a specific service. Thus, a single number may be used to specify each level of QoS. Furthermore, we employ pricing concepts to manage resource consumption.

We implemented a prototype of the interface using *Java* and the *Remote Method Invocation* network interface. We use ATM as the underlying network supporting QoS.

*This work was done in collaboration between the ATM lab of AT&T labs and the ATM lab in Tel-Aviv University.

†Computer Science Dept., Tel-Aviv U., Israel 69978, and AT&T Labs.

‡AT&T Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

1 Introduction

Multimedia applications over networks are becoming commonplace. Video on demand, video conferencing, Internet telephony, and live network games are just a few examples of such applications. Interest has intensified with the advent of broadband networks such as ATM. However, while there are many tools available for network programming (such as sockets, RPC, CORBA, etc.), none of them support the specific needs of multimedia applications, such as guarantees for network quality of service. Indeed, live multimedia may need connection bandwidth, cpu time, and disk access guarantees, much like hard real-time applications. The need for networks that support quality of service is part of an ongoing transition in the computer world from *best effort*, non-reliable services to *guaranteed quality* services. This is similar to the difference in real life between voluntary services and business services.

Our goal in this project is to define a simple and efficient interface for quality of service negotiation and moderation. Such an interface (API) should be simple to comprehend, yet powerful enough to describe the needs of real-time distributed applications. We suggest an extension to object oriented remote method call interfaces (CORBA is one example of such an interface) that can be used by software developers to negotiate for quality of service of any type. In particular, we implement a distributed network object that supports QoS requests made by applications to the network.

In our implementation we have extended the Java RMI (Remote Method Invocation) interface to support QoS, assuming that the underlying network is ATM. We keep our interface as simple as possible so that it is easy to implement and use. Our implementation uses two Sun sparc machines running Solaris 2.4/5. The machines are connected to a FORE ASX-200 ATM switch by FORE SBA200 cards.

Section 2 describes our basic principles and guidelines. The specific API designed in this work is described in Section 3. In Section 4, the framework in which our prototype was implemented is given. Summary, conclusions and future work are provided in Section 5.

2 The QoS management scheme

We consider a system of distributed objects as in object oriented languages (e.g., Java, C++). We further assume the client server model in which servers supply services to clients. Moreover, the responsibility for ensuring and supplying the desired quality of service between the server and the client is on the server, thus keeping the client relatively simple.

To facilitate the acquisition of QoS from servers and the sharing of services by different clients, we develop a model that borrows from the economic/business world. In this model, servers partition their services into quality classes and assign a price to each quality of service class. Clients that request a service at a certain class of quality, receive it if it is available. When granting a service at a certain class, the server notifies the client of the price for that service class.

In our implementation, upon receiving a request for a service, servers subcontract and moderate different services by interaction with a special set of objects defined for this purpose. We have implemented *the network object*, a distributed set of these special objects, that together are able to provide network services with different qualities to the servers. Other services, such as buffer space, CPU time may be acquired by the server from other system objects that are not necessarily distributed.

Thus, in the abstract, we assume the following general scenario: 1. Client wakes-up. 2. Client asks a server for a service from a certain quality class. 3. Server acquires and arranges the service in that (or close to that) quality class. 4. Server notifies the client of the service price. 5. Client either start using the service or decides to abort perhaps because the service is too expensive.

In our model, just like in the real world, every service provider must be able to *price* the service it supplies. The service provider (hence the server) may require specific quality of service from the network, the cpu, the memory or even from other servers. However, for the client, it must summarize all resources consumption into two numbers: the *base price*, and the *price rate* per second. Also, the server must not assume the client is aware of the specifics of the service. Therefore, it must be able to accept a generic request for quality of service, such as "give me a good service" or "give me a cheap service", and map it to specific quality of service classes. On the other hand, the client must be able to define which service it needs and, if it is able, any specific quality of service parameters.

Since pricing is involved, unique identification of both client and server is crucial. For simplicity, we chose the credit account metaphor. That is, clients identify themselves to the server and the server may manage an account per client. One could conceive of a system, similar to the cash system, where anonymous clients can receive services by paying "cash". However, such systems are harder to implement and need to address many other issues that are orthogonal to the issues addressed here.

Currently, we assume a "friendly" environment where each object chooses its own unique name. Obviously, there is much place for improvement at this point.

To clarify the model we consider some real world example and its analogy in multimedia applications.

Assume that a person needs to fix her car. Generally, the person (the client) would contact a garage (the server) and ask to fix her car. However, for the garage to give a service better than *best effort* the client must supply the class of service it expects; maybe the client want to use old parts, thus keeping expenses low; maybe she wants to get the best parts, no matter what the price is.

Once the garage accepts the work, it has a multitude of expenses. Spare parts, workers time, external jobs etc. However, all of this should be transparent to the client. Usually the client is only interested in the bottom line, i.e., the total price.

This example highlights the fact that the monetary system provides a way to summarize all the services and their associated quality in one number, the total cost.

Several extensions of the above scenario are possible; First, the client may be able to define specific modes that affect the service, such as "I need the car by tomorrow" or "don't spend more than $200". Second, a client may want to know the server's expenses so it can track the server's performance. Third, the client may get a price estimation before actually committing to the buy the service - thus making it possible to "shop around".

The analogy of that example is a software client requesting a movie from a video on demand server. First, the client may want to specify service modes such as: which movie it wants, what is the maximal resolution of the screen and so on. Then, it must specify what class of service it expects (this can also be seen as how much the client is willing to pay). The server translates this generic quality class into specific quality parameters, such as actual resolution and number of frames per second. Now, the server can request specific quality of service from the network, cpu, disks etc. And, the server can price the service and send the client the bill.

3 QoS requests Application Programming Interface (API)

Here we suggest a simple QoS requests API that we believe is general and powerful enough to support a wide range of applications. We claim that in practice all service classes of a particular service can be ranked in a total order. Hence, each quality class can be represented by a single number (between 0 and 255 in our prototype).

The general premise is that a client that wants a service at quality q invokes the `giveMeServiceWithQ(`q`,`Cid`)` (`giveMeQ(`q`,`Cid`)` in short) method of the server, where Cid is the client identity. That is, in each QoS providing server object such a method must be implemented.

To achieve a higher degree of service specification, we extended the `giveMeQ` method. The method may either take two parameters, as above, or three parameters where the third parameter is a vector of QoS objects, one for each tunable parameter of the service, as particular to the server. This optional parameter may be used by expert clients that are familiar with the internals of the service in question.

When method `giveMeQ(`q`,`Cid`)` is invoked, the server acquires the necessary resources at the desired QoS class, as determined by q, and calculates the total price (both base price and price rate) of the service. The calculated price (pair of numbers) is then returned to the invoking client.

Upon completing the `giveMeQ(`q`,`Cid`)` method and receiving the price, the client may either start using the service by invoking the appropriate method at the server, or the client may decide not to take the service (i.e., it may shop around for better prices etc.).

Due to space limitations a more extensive code example is not included but can be found with other details in
http://theta.math.tau.ac.il/QoSrmi/report/html/report.html.

4 Framework

Our goal in this project was to be as general as possible. The model is relatively general, we only assume an object oriented language following an "RPC" like network protocol. To test the basic concepts we have implemented the first prototype on a restricted system with the following tools:

Java We programmed our implementation in Java because it is easy to use as a developing tool, and because it is becoming the language of choice for network applications.

RMI We preferred to extend the Remote Method Invocation interface of Java rather than the CORBA interface because it is simpler to implement and because it is better integrated with the Java language.

ATM Nowadays, ATM networks seem to be the most common quality-of-service supporting network available. In our prototype all the machines are connected to an ATM network via FORE[1] cards.

These choices and restrictions can be changed at later time by implementing the interface in C++ and/or Smalltalk; Importing the interface to CORBA; Implementing the software over other vendors' equipment; and Implementing the software over other networks such as RSVP[2] and IPv6 [3].

We used two Sun[3] sparc ultra machines running Solaris 2.4/5. The machines were connected to a FORE-Runner ASX-200 ATM switch and by a FORE SBA200 card running version 4.0 of the drivers. We used Java Development Kit (JDK) 1.0.2 and RMI 0.8.

5 Summary

In this abstract we described a simple extension to the standard object oriented programming model which lets objects specify quality of service needs from other

[1] http://www.fore.com
[2] http://www.ito.darpa.mil/Summaries95/8420–USC_ISI_ReSerVation.html
[3] http://www.sun.com

objects. In particular, we used this model to implement a network object which supplies quality of service guarantees for connection between objects residing on different machines. Our prototype implementation is written in Java and extends the RMI interface. Also, we assume that the underlying network is based on FORE ATM equipment.

The two observations that we can make from our initial experiments that managed only network bandwidth allocation are; first that CPU time in many cases can be the bottleneck rather than the channel BW. The CPU speed may become the bottleneck because of different parameters, such as buffer size (small buffers result in many context switches). Second, hardware components miss-match may create artificial bottlenecks. For example, the FORE SBA 200 ATM driver uses one queue for all the outgoing connections. Thus a low quality connection in the head of the queue degrades the service quality received by connections stuck behind it.

In the future we hope to improve the implementation to overcome the above problems and to extend it to address issues such as security and synchronization of multimedia streams.

References

[1] *The Java Language Specification (1.0 Beta).* Online information is available at ftp://ftp.javasoft.com/docs/javaspec.ps.tar.Z.

[2] *Java Remote Method Invocation Specification.* Online information is available at ftp://chatsubo.javasoft.com/pub/releasesbl31/rmi.ps.Z.

[3] *Internet Protocol, Version 6 (IPv6) Specification* . Online information is available at ftp://ds.internic.net/rfc/rfc1883.txt.

[4] Mario Miguel Fonseca and Joaquim Soares. ATM Sinalling Implementation on a UNIX STREAMS Enviroment. *The fourth conference on Broadband Islands*