Quality of Service in Indirect Communication Systems

Filipe ARAÚJO
Universidade de Lisboa
filipius@di.fc.ul.pt

Luís RODRIGUES
Universidade de Lisboa
ler@di.fc.ul.pt

Abstract

This paper outlines an architecture that intends to ensure Quality of Service (QoS) to applications using the publisher-subscriber paradigm. The key idea is to subscribe information not only by subject and content but also according to QoS parameters. This must be done with cooperation of all intervenients: publishers, which should tag information they are producing with application-level QoS parameters; subscribers, that do content-based QoS addressing; and message brokers, which use a QoS based network infrastructure.

1 Introduction

Indirect communication, in particular the publisher-subscriber communication model, is gaining increasing acceptance as a useful alternative to direct communication models, such as the ones based on remote invocations. The main advantage of this paradigm consists of offering a communication model based on a weak coupling among participants, which do not need to be aware of the location or number of its peers. This simplifies the reconfiguration of the applications and eases the re-use of components in new applications.

A limitation of most existing architectures that support the publisher-subscriber communication, such as for instance, the SIENA (Scalable Internet Event Notification Architectures) [4] system, is that they do not offer support for the negotiation or enforcement of Quality of Service (QoS) parameters (such as required bandwidth or latency, for instance). This is a significant drawback since QoS features are an important component of modern applications, and its use and support has been widely studied in the context of direct communication [3, 2, 9, 1].

Unfortunately, most of the work on QoS is based on the establishment of channels or connections that reserve the resources required to provide the desired QoS parameters. This paradigm fits perfectly in the direct communication model but does not fit with the decoupling that makes the publish-subscribe model appealing. In the indirect communication model, the applications should not be forced to explicitly set-up channels. Instead, they should remain oblivious to the number and characteristics of the participants involved in the communication and should be concerned exclusively on the properties of the information they are able to publish or subscribe.

Therefore, a new system model has to be designed to allow the seamless integration of QoS features in indirect communication systems. This model should:

- Allow the application to indirectly negotiate QoS parameters, by allowing to express QoS properties has a characterization of the information being produced or subscribed.
- Delegate on the message broker the task of establishing the required low-level connections, on behalf of publishers and subscribers, based on dynamic information on the number, location and characteristics of producers and consumers and also on the QoS characteristics of the information exchanged in the system.

An interesting feature of the message broker that we are envisioning in our architecture, is that it should be able to autonomously set-up and reconfigure network-level channels. Dynamic adaptation is fundamental to optimize the system resources required to convey information from publishers to subscribers with the desired QoS parameters. This is in the lines of the architecture proposed in [6] where the number of IP multicast addresses needed to support a specific subscription pattern is automatically selected by the system.

It should be noted that such functionality is more complex than that offered by trading components of direct communication systems [7]. In such systems, a trader is a component that can provide references to server objects that match a set of desired characteristics specified by the clients. However, a trader is not responsible for setting up channels between clients and servers nor to reconfigure these channels based on subsequent requests.

The current paper reports preliminary work on defining an architecture that integrates the use and provision of QoS in indirect communication systems. As in many indirect communication architectures [4], publishers and subscribers only talk to intermediate servers that coordinate themselves to forward the information adequately. The paper points out some necessary changes to the publisher-subscriber model such that QoS is to be viewed in an integrated way by applications, message broker and network. Namely, we describe how applications can do content-based addressing on QoS parameters as they do with ordinary subjects. QoS ensuring in indirect communication systems raises some problems worthwhile mentioning, namely the following:

- How to expand the publisher-subscriber model such that publishers and subscribers can make hints on QoS parameters they intend to, respectively, offer and obtain. These QoS parameters should preferably be application-level parameters latter mapped onto network or operating system-level parameters.
- Understand how can message brokers fulfill QoS demands issued by applications, using traditional QoS mechanisms offered by the underlying network.

To ensure QoS, a stacked architecture with three layers will be introduced in Section 2. Section 3 discusses the operating system and network QoS properties needed to support this architecture. Section 4 presents the middle layer and points out some problems that must be faced by a message broker supporting QoS features. It also proposes a simple architecture for message broker servers. Section 5 presents the top layer and discusses how QoS reservations should be made. Finally, Section 6 provides an example and Section 7 concludes the paper.

2 Indirect Communication QoS Architecture

The proposed architecture is comprised of three stacked layers. The lowest layer, which is responsible for supporting QoS guarantees, includes the network and operating system infrastructure. The layer immediately above, the middle layer, is the message broker that must ensure the QoS requested by the applications. Finally, the application resides on the top layer. Figure 1 illustrates the architecture.

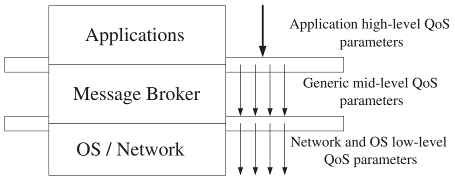

Figure 1: The proposed architecture

Note that these layers are deliberately coarse-grain and merely attempt to divide the functionality according to different concerns. In particular, the lower layer encapsulates all network-level and operating-system level mechanisms that are required to support QoS operation. There is a significant body of work already avaliable at this level and we intent to reuse these mechanisms as black-boxes in our architecture.

The QoS message broker itself constitutes the middle layer. With regard to the brokers of existing publish-subscribe systems, the QoS aware broker must know and understand the QoS attributes carried in the messages and manage the use of system resources accordingly. This middle layer offers a set of generic mid-level QoS parameters to applications (described below). These QoS parameters must easily be mapped to network and operating system-level parameters. The mapping function can even be the identity function. Note that by having a layered architecture it is possible to study and propose solutions for each level individually, with some modularity that allows, for instance, to replace individual layers by others with perhaps different QoS characteristics.

The top layer is where applications reside. In the indirect communication paradigm application can play two distinct roles: publisher and subscriber. Publishers advertise the attributes of the information they publish; in our architecture they include QoS-related attributes in the advertisements. On the other hand, subscribers describe the attributes of the information they are interested to receive, including also QoS attributes. This somehow resembles the Resource ReSerVation Protocol (RSVP) advertisements/reservations mechanism [2]. For ease of programming, applications should be able to specify their own high-level QoS parameters. These should unambiguously map to the generic mid-level QoS parameters, because the message broker must understand applications QoS requirements, in order to satisfy their demands [8].

3 OS and Networks

In order to ensure QoS to networked applications, including message broker servers and clients, networks and operating systems with provision for QoS are needed. Standard mechanisms supported by these subsystems, such as RSVP [2], differentiated services [1] or others, would be exploited by the broker to enforce the QoS required by the applications. Depending on the QoS demanded by the applications, minimal services would be required from the OS and network infrastructure: if mapping mid-level QoS parameters to network QoS parameters is not possible by lack of expressiveness at the bottom layer, the application-level requirements will not be met.

4 Distributed Message Broker with QoS support

The message broker may have a fairly complex structure, being composed of several servers connected to each other that coordinate to forward advertisements and subscriptions among them [5]. Clients connect to the message broker by means of one of the servers. A network with several message broker servers and clients is illustrated in figure 2.

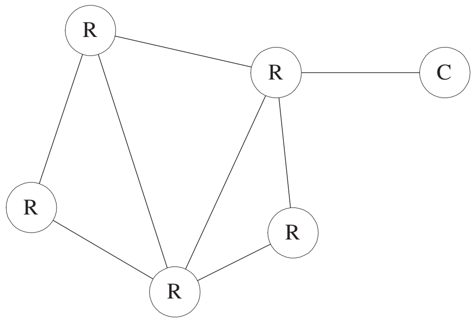

Figure 2: Message broker servers and clients

Therefore, if QoS guarantees must be given, the message broker must have the knowledge of the QoS requirements of all clients, including available resources on links connecting to clients [1]. In addition, some mid-level generic QoS parameters should be defined. These QoS parameters should not be much different from a set including {`Latency, Jitter, Bandwidth, Loss ratio, Availability`}. Others can be included in the set if they become important.

In the model we propose, at setup time the brokers establish a number of QoS channels among themselves. These reservations are shared by clients' traffic. Therefore, in response to subscriptions, the broker may allocate a share of its resources to its clients. This calls for an Admission Control (AC) module at the message broker level. This module should be responsible for accepting or rejecting QoS subscriptions depending on applications requests and depending also on message broker resource utilization. An accounting module is also necessary, for keeping track of resource consumption [2].

The response to a rejected QoS subscription can be twofold: a reject message can be generated inside the broker and headed to the subscriber; or there can be no answer at all, leaving to the subscriber the burden of understanding that it made an unreasonable request that was rejected and forcing it to undo the QoS subscription. This is clearly worse from the application point of view, but may result in simpler, thus more scalable and feasible solutions.

When new QoS subscriptions arise, the server should consider, not only the QoS requirements, but also the information requested by subscribers. The satisfaction of QoS requirements extends the content-based routing problem [5] in ways that may not be trivially solvable.

Another class of problems that will be extended relates to grouping subscribers under a limited number of IP multicast addresses. This problem is significant, because IP multicast addresses should be seen as a scarce resource. Clients' location is also an important aspect when finding answers to some of these questions.

Finally, a simple proposal for the structure of content-based server with QoS support is illustrated in Figure 3. The problems related to applications traffic shaping/conditioning are ignored for now. Note the absence of a signaling module that we believe can be omitted.

5 Content-Based QoS

5.1 Advertisements and Subscriptions

One of the ideas behind the outlined architecture is to address QoS parameters in a way that closely resembles addressing of ordinary information contents. For instance, this means that if a publisher

[1] Preferably, the broker should always deal with its generic QoS parameters for the already stated reasons, which include the possibility to use different network technologies. For instance, a link to a client may use any available technology.

[2] At this point we are not yet addressing billing issues.

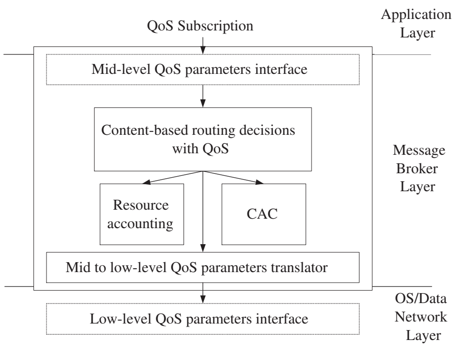

Figure 3: Message broker server internals

is broadcasting video, a subscriber might decide whether it wants or not to see the video, based on advertised bandwidth. Furthermore, if a second publisher is also broadcasting the same images, the decision about which server to use can be automatically obtained based on QoS parameters [3].

Therefore, this kind of QoS addressing fits very naturally in the publisher-subscriber model. This brings some advantages: firstly, fewer additional mechanisms will be needed for applications to select QoS. Secondly, matching publishers' capabilities with subscribers' needs is made easier, because it looks a lot like ordinary information matching.

As mentioned before, this content-based QoS addressing is only one piece of the solution. Besides that, the message broker must know both the capacity of the infrastructure it is using and the QoS applications are requesting. In fact, this means that QoS contents in a message cannot be opaque to the message broker. This is necessary for two main reasons: i) to know whether or not clients (i.e., publisher or subscribers) QoS requests are acceptable; and ii) to make optimization decisions based on clients QoS requests.

5.2 Application-level QoS parameters

Applications should be able to create high-level QoS parameters. To achive this goal, it is required that: *i)* the mid-level generic QoS parameters recognized by the broker are well defined, `ii)` rules are defined to combine the mid-level parameters to create new higher-level QoS parameters that can simplify the addressing.

Mid-level generic QoS parameters were presented in section 4. Each one of those QoS parameters has a data type. So, let the names of those parameters identify data types. For instance, `latency` may be an alternative name for `real` data type (as the `real` data type in the *C* language). Using mid-level generic QoS parameters, new types for QoS parameters may be created composing new sets with elements from the initial basic set. For instance, the type `QoSTypeX` may be composed of {`Latency, Jitter`}. We shall call these *sets of types* (note that a set of types is itself a new type).

[3]In this second case, the subscription should necessarily exclude one of the publishers otherwise it would receive video from both.

Dealing with sets of types is slightly harder than dealing with basic types like integers or strings, because composition must be considered. In particular, it would be interesting to admit matching of different sets, depending on matches of the underlying QoS types. This is something that is not allowed with the basic types that must have exact matches.

Before we proceed we shortly define the following concepts: *notification*, *subscription*, *advertisement* and *covering relation*. *Notification* is a response message to an event. It is comprised of `(Type, Name, Value)` attributes. A notification message is flooded all over the system to subscribers requesting it. A *subscription* specifies a subset of notifications that a subscriber wants to receive. An *advertisement* specifies a subset of notifications that a publisher is going to produce. The *covering relation* may exist between advertisements and notifications or subscriptions and notifications. An advertisement covers a notification when each one of the notification's attributes satisfies the advertisement specification. A subscription covers a notification when each one of the attributes specified in the subscription is satisfied by the notification. For formal definitions and examples see [5].

It is now important to understand how do covering relations happen between advertisements, notifications and subscriptions with sets of types. To do this consider the existence of two previously defined sets: `QoSTypeA` and `QoSTypeB`, such that `QoSTypeA` contains `QoSTypeB` (`QoSTypeA` $\supset$ `QoSTypeB`). Then, the following relations apply:

- A subscription s with attribute of type `QoSTypeA` does not cover a notification n with attribute of type `QoSTypeB`. There is a type mismatch.
- A subscription s with attribute of type `QoSTypeB` may or may not cover a notification n with attribute of type `QoSTypeA`. It depends on the thresholds for the common types.
- An advertisement a with attribute of type `QoSTypeA` may or may not cover a notification n with attribute of type `QoSTypeB`. It depends on the thresholds for the common types.
- An advertisement a with attribute of type

QoSTypeB does not cover a notification n with attribute of type QoSTypeA. There is a type mismatch.

Tables 1 and 2 summarize these covering relations. The covering relation *subscription covers notification* is expressed by $\sqsubset_S^N$ while the relation *advertisement covers notification* is expressed by $\sqsubset_A^N$. Remind that QoSTypeA $\supset$ QoSTypeB.

This composition mechanism is useful whenever a composed QoS parameter becomes widely used. In order to be useful, the ability of creating new types must be complemented with the ability of defining standard operating points such as "telephone quality sound transmission" or "errorless data transmission", for instance. Naturally, new problems arise with the construction of new QoS type parameters and QoS points of operation, namely:

• How should applications tell the message broker which contents are QoS-related and how are QoS parameters composed.

• Greater than relations are harder to define. If a set of QoS types includes {Latency, Bandwidth}, given variables A and B of this type, what does $A > B$ means? It should not mean that A has greater latency and greater bandwidth, because, then, A would not represent a better point of operation than B. Solutions may be found on operator overloading, where different definitions for $>$, $<$ and $=$ must be given.

6 An Example

In this section is provided an example. First, some sets of types and some QoS operating points are defined (see below). The syntax intends to resemble programming languages like *C* or *Pascal* for the sake of simplicity. Table 3 shows advertisements made by two different publishers. Only the relevant attributes are shown. Tables 4 and 5 show QoS subscriptions made by five different subscribers. Table 6 shows covering relations between subscribers and notifications sent by both publishers. "Maybe" instead of "Yes" is used, because some of the notifications sent by publishers and covered by their advertisements *may not* be covered by subscriptions.

```
QoSType Image = {Jit-
ter, Bandwidth};

const Image Color = {0.1, 5 Mbps};
const Image HDTV = {0.1, 15 Mbps};
const Bandwidth Nice = 10 Mbps;
const Jitter Sharp = 0.01;
const LossRatio NoLoss = 0;
```

Table 3: Advertisements

Publisher 1 (P1)	Publisher 2 (P2)
string Source = Camera1 string Room = Lab Image Quality = Color	string Source = Camera2 string Room = Lab Image Quality = HDTV

Table 6: Advertisements and subscriptions matching

	S1	S2	S3	S4	S5
P1	Maybe	Maybe	No	No	No
P2	Maybe	Maybe	Maybe	No	No

Returning to Figure 2, consider that client 1 is publisher P1, client 2 is publisher P2 and client 3 is subscriber S3. By table 6, P1 and S3 do not match. But if P2 produces notifications with the attributes corresponding to camera 2, from the lab and with HDTV image quality, then S3 will cover these notifications, i.e., P2 and S3 will match. This means that subscriber S3 will receive images from Camera 2. Now, consider that the path between publisher P1 and subscriber S3 goes through servers 3, 4 and 5 (this is a routing problem not studied here). Some bandwidth must be reserved on links of that path. This bandwidth reservation will only take place if AC modules along the path accept it. Note that all these steps, routing, admission control and bandwidth (or other resource) reservation occur at the message broker level.

7 Conclusion

This paper aimed at combining two usually unrelated issues: the publisher-subscriber communication model and the provision of QoS. To do this, a three layer architecture was presented. Only the two highest layers, application and message broker, have been discussed, since the bottom layer will reuse existing network and operating system mechanisms. At the top layer, applications should do content-based QoS addressing. Applications should also be able to define new complex QoS parameters built on top of basic ones, like latency or bandwidth. At the message broker level, the appropriate reservation and routing policies will be implemented to ensure that the desired QoS can be offered.

The paper identifies several challenges that must be addressed by an architecture that successfully integrated the use of QoS in indirect communication

Table 1: Covering relations between advertisements and notifications

	a includes `QoSTypeA`	a includes `QoSTypeB`
n includes `QoSTypeA`	$a \sqsubset_A^N n$ or $a \not\sqsubset_A^N n$	$a \not\sqsubset_A^N n$
n includes `QoSTypeB`	$a \sqsubset_A^N n$ or $a \not\sqsubset_A^N n$	$a \sqsubset_A^N n$ or $a \not\sqsubset_A^N n$

Table 2: Covering relations between subscriptions and notifications

	s includes `QoSTypeA`	s includes `QoSTypeB`
n includes `QoSTypeA`	$s \sqsubset_S^N n$ or $s \not\sqsubset_S^N n$	$s \sqsubset_S^N n$ or $s \not\sqsubset_S^N n$
n includes `QoSTypeB`	$s \not\sqsubset_S^N n$	$s \sqsubset_S^N n$ or $s \not\sqsubset_S^N n$

Table 4: Subscriptions 1, 2 and 3

Subscriber 1 (S1)	Subscriber 2 (S2)	Subscriber 3 (S3)
string Source = Any string Room = Lab Image Quality = Any	string Source = Any string Room = Lab Image Quality $\geq$ Color	string Source = Any string Room = Lab Bandwidth Quality $>$ Nice

Table 5: Subscriptions 4 and 5

Subscriber 4 (S4)	Subscriber 5 (S5)
string Source = Any string Room = Lab Jitter Quality = Sharp	string Source = Any string Room = Lab LossRatio Quality = NoLoss

systems and provides a sketch of some possible solutions for these problems.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differenciated services, December 1998. Request For Comments 2475.

[2] Ed.R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) — version 1 functional specification, September 1997. Request For Comments 2205.

[3] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview, June 1994. Request For Comments 1633.

[4] Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, December 1998.

[5] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Content-based addressing and routing: A general model and its application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, January 2000.

[6] L. Rodrigues M. Guimarães. Arquitectura híbrida para publicação e subscrição de informação na internet. In *3a Conferência Sobre Redes de Computadores (CRC 2000)*, November 2000.

[7] Object Management Group, Inc., OMG Headquarters, 250 First Avenue, Suite 201, Needham, MA 02494, USA. *Trading Object Service Specification*, May 2000.

[8] Frank Siqueira and Vinny Cahill. An open QoS architecture for corba applications. In *the 3rd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)*, pages 328–335, 2000.

[9] J. Wroclawski. The use of RSVP with IETF integrated services, September 1997. Request For Comments 2210.