The IndiQoS Message Broker: an Instantiation Using RSVP

Filipe Araújo

Luís Rodrigues

DI–FCUL TR–02–3

March 5, 2002

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

The IndiQoS Message Broker: an Instantiation Using RSVP

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

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

March 5, 2002

Abstract

This paper describes the first proof-of-concept implementation of the IndiQoS architecture: a system that intends to support the seamless integration of QoS provision in publish-subscribe systems. A unique feature of the IndiQoS architecture, is that applications are structured following the publish-subscribe paradigm and express QoS parameters with constructs similar to those used to handle other aspects of the data flow, such as the ones used to express interest in a particular subject or content.

Although we aim to build a system that is able to manage different QoS mechanisms at the network and operating system level, the current paper focus on the use of RSVP and integrated services in the IndiQoS architecture. Two main aspects are studied: definition of QoS parameters for use by publish-subscribe applications and optimal assignment of data network resources for a given configuration of publishers and subscribers.

1 Introduction

The indirect communication model (supported by publish-subscribe systems) owns a number of attractive characteristics that cannot be found in the direct communication model (supported by mechanisms such as sockets or remote invocations). In particular, the indirect communication model offers a weak coupling of participants in distributed computation that simplifies reconfiguration of applications as well as re-use of components in new applications.

Unfortunately, most existing publish-subscribe architectures and implementations, such as for instance, the CEA (Cambridge Event Architecture) [3], SIENA (Scalable Internet Event Notification Architectures) [6], CORBA Event Service [9], CORBA Notification Service [8] or JMS (Java Message Service) [13], offer very limited support to QoS provision. For instance, the specification of bandwidth or latency constraints is typically not considered in these architectures. This is a significant drawback, since QoS parameters are an important component of modern applications.

We have recently sketched an architecture capable of guaranteeing QoS to publish-subscribe applications[2]. This architecture, called IndiQoS, has the interesting feature of supporting the definition of QoS parameters constructs similar to those used to handle other aspects of the data flow, such as the ones used to express interest in a particular subject or content. In the same paper, we have also identified the principles that should guide the development of the IndiQoS message broker the main challenges in its implementation.

In this paper, we present a particular instantiation of the IndiQoS architecture for the case where RSVP [4] with Integrated Services [5] are used as the underlying network mechanisms to support the provision of QoS parameters. In particular, we concentrate on two problems:

- *selection of meaningful QoS parameters for publishers and subscribers*: Qos parameters for applications must be chosen carefully to allow their translation into integrated services parameters;
- *mapping problem*: given a concrete set of QoS advertisements and subscriptions, how should the message broker configure the underlying network. This is an optimization problem.

We must emphasize that the goal of the IndiQoS architecture is to support different low-level mechanisms, and not only RSVP with Integrated services, and to hide from the applications the details of these underlying mechanisms. Therefore, we seek for generic solutions and assume that the message broker itself must be responsible for managing the necessary QoS connections or reservations and QoS control traffic on behalf of applications. QoS connections/reservations are seen as "resources" of the message broker. In particular, as we intend to use RSVP in our QoS publish-subscribe architecture, RSVP reservations are a particular type of resources to be managed by the IndiQoS message broker. For instance, when new subscription with QoS parameter is made, the message broker may decide to allocate new resources by setting up a new reservation or it may decide to share an existing resource to support the dissemination of notifications.

The paper is structured as follows. In Section 2 we first present an overview of our framework for a publish-subscribe architecture with support for QoS. In Section 3 we define QoS parameters for use by QoS publish-subscribe applications. In Section 4 we point out the problems that will arise when implementing the message broker. Section 5 concludes the paper.

2 A QoS Publish-Subscribe Architecture

We have recently proposed the IndiQoS architecture, a publish-subscribe architecture with support for QoS guarantees [2]. In this architecture, publishers are responsible for characterizing the QoS properties of the stream of notifications they produce. On the other hand, when subscribers register their interest in receiving certain types of notifications, they can specify filters that express QoS constraints. The interpretation of these filters depends on the QoS profiles advertised by publishers.

The IndiQoS architecture, requires producers of information to disseminate advertisements that include the QoS profile of the notification flow. These advertisements are required for two main reasons. Firstly, subscribers need to be aware of the QoS parameters of the information they wish to receive. It would not make much sense for a subscriber to ask for some QoS constraints with no correspondence to any publisher's generated traffic. Secondly, it is necessary to ensure QoS to notifications in transit. Hence, the message broker needs to reserve the necessary resources to support the notification flow.

In the IndiQoS architecture, the profile of the notification stream is advertised in the following way:

Publisher p = **new** *Publisher* **of** <*Type*> **withProfile**(*X*) **withQoSProfile**(*Y*)

Where *X* describes the contents of the information and *Y* is its QoS characterization. For instance, consider a provider of information that collects and disseminates data from a temperature sensor: we could have *X = (room="lab1", temperature=any, precision=0.01)*. *Y* describes the QoS properties that should be guaranteed to the set of notifications covered by *X*. How to express these properties is one of the core issues of this paper and its discussion is postponed to section 3.

On the subscriber side, QoS attributes are expressed with a filtering condition similar to filters used on information contents:

Subscriber s = **subscribe** <*Type*> **where** (*filter*(<*Type*>)) **withQoS** (*Y'*)

<*Type*> should be the same type present in the advertisement. A subscriber limits the set of notifications it receives of that particular type with a function (*filter*) that depends on the particular type. An additional constraint is introduced in our architecture with term *Y'* that expresses QoS. Discussion of this term *Y'* is also postponed to section 3.

To publish a notification, the publisher would then do:

e = **new** <*Type*>(*X*$_1$)
p.publish (e)

Where we would have, for instance, *X*$_1$ *= (room="lab1", temperature=22, precision=0.01)*. The QoS properties enforced at run-time are derived dynamically from the combination of the profile *Y* advertised by the publisher and from the filter *Y'* specified by the subscriber(s). Therefore, the actual QoS parameters of each notification, are not explicitly specified in the *publish* operation but computed by the IndiQoS message broker which is also responsible for making the appropriate reservation of network resources to support them.

As in any system supporting indirect communication, in the IndiQoS architecture an application does not need to be aware of the number or location of its peers. Instead, the application needs only to be concerned with the properties of the information being produced or consumed. The same reasoning applies

to QoS considerations: the application needs only to be concerned with expressing QoS profiles and constraints without dealing explicitly with the reservation of the underlying resources (that, naturally, depend on the number of publishers and subscribers).

In the IndiQoS architecture, complexity of selecting the appropriate set of reservations of network resources is delegated on a QoS-aware message broker. To support QoS, the message broker must use data networks and operating systems with provision for QoS and use resources from these underlying infrastructures. In this paper, we study a particular case that considers exclusively RSVP resources.

3 RSVP Resources

This section is concerned with defining the parameters to be used by the publishers to specify the profile of the notification stream, and by the subscribers when specifying QoS filters. The definition of these parameters must conciliate the following goals:

- it should be possible to translate these parameters into the low-level parameters used by the underlying RSVP [4] with IETF integrated services [5] mechanisms;
- they should be generic, in the sense that its use should not prevent an application to be ported to other networks, using different types of mechanisms to enforce QoS.

Therefore, to define generic QoS parameters for IndiQoS we need first to be aware of integrated services QoS parameters. For commodity we summarize them here (see [14]):

- The sender specifies the traffic it is going to generate in RSVP SENDER_TSPEC objects. A token bucket with some additional parameters describes QoS parameters: token bucket rate (r), token bucket size (b), peak data rate (p), minimum policed unit (m) and maximum packet size (M). For a comprehensive foundation on token bucket see for instance [11]. Peak rate parameter (p) is the sender's peak traffic generation rate or, if not controlled, physical interface line rate (it may be set to infinity if no other value is available); the minimum policed unit parameter (m) is the size of the smallest packet generated by the application [1]; the maximum packet size parameter (M) is the size of the largest packet generated by the application. SENDER_TSPEC objects do not depend on the class of the service. This means that they have the same QoS parameters for both controlled-load and guaranteed services.
- The receiver RSVP FLOWSEPC object differs from controlled-load to guaranteed service. For Controlled-Load service the QoS parameters are (TSpec): token bucket rate (r), token bucket size (b), peak data rate (p), minimum policed unit (m) and maximum packet size (M). These parameters look like those for SENDER_TSPEC objects except that i) M is now set for the smallest Maximum

[1] Including all protocol headers above the IP level.

Transmission Unit (MTU) existing on the path connecting sender and receiver; ii) description of (m) is much more complicated and is not included here. For guaranteed service, additional QoS parameters are required (RSpec): rate (R) and slack term (S). Rate (R, $p > R \geq r$) is a value that represents the bandwidth that a dedicated channel should provide between source and receiver to achieve end-to-end behavior conforming to the model defined in [12]. This value is useful to determine a bound to the delay. The slack term (S) is added to the so calculated delay to provide the final delay the application is requesting.

Hence, to create a publish-subscribe architecture on top of RSVP with Integrated services we must consider the following:

- QoS parameters specified by publishers in advertisements must be translated to token bucket parameters, together with a peak data rate. Limits for advertisement (packet) sizes should also be provided;
- QoS parameters specified in subscriptions must also be translated to the same token bucket and peak data rate parameters. However, subscribers can add latency requirements. If they do this, the class of service should be guaranteed, if they do not, the class of service should be controlled-load.

From [12], which describes the guaranteed service it is easy to see that the only QoS parameters that are guaranteed are bandwidth (for both services, controlled-load and guaranteed) and maximum latency (for guaranteed service only). It is therefore pointless to try to guarantee other parameters like jitter, loss ratio or availability.

Table 1 shows the QoS parameters that are supported by this instantiation of the IndiQoS architecture. Both publishers and subscribers have to express the size of the notifications and notification rate. Subscribers may also wish to add another constraint, concerning maximum latency.

Table 1: QoS parameters for publish-subscribe applications

Publisher		Subscriber	
size	rate	size & rate	latency
size of notifications **or** (if notification size varies) minimum **and** maximum notification size	average number of notifications per time unit **and** (if known) maximum number of notifications on some period of time **or** (if the produced information is strictly periodic) period of time between two consecutive notifications **or** token bucket parameters	identical to publisher	may or may not be specified

At this stage, we only admit two simple classes of service: with and without QoS. If applications do not require QoS, then, QoS parameters do not need to be present in either advertisements or subscriptions.

We call “sporadic” to this class of service. In RSVP with Integrated services this corresponds to a best-effort service. If applications require some QoS, then the resulting RSVP class of service depends on QoS parameters present on both advertisements and subscriptions. Figure 1 shows all possible combinations in Karnaugh map-like form. An ’X’ is used whenever the combination makes no sense.

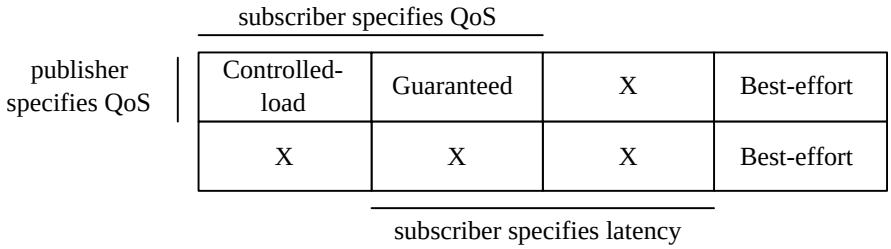

Figure 1: Correspondence to Integrated Services

Translating application level QoS parameters to integrated services parameters is straightforward, because they have an almost direct correspondence. However, there are some scenarios that deserve further attention. Whenever notification size is not available, maximum notification size should be used, because this represents the worst-case situation. Maximum notification rate should also be used to provide a value for peak data rate (p).

A slightly more difficult scenario occurs when subscribers impose restrictions on latency. In this case, it is necessary to determine the rate (R) and slack term (S). By [12] we have inequalities 1, where d is the end-to-end delay and l the requested latency. C_{tot} and D_{tot} are rate dependent and rate independent delays across entire path, respectively. [12] does not specify how C_{tot} and D_{tot} are obtained by the application. It just ensures that there should be a mechanism for it. Considering only the latency inequality we may derive a lower bound for R in inequality 2.

$$d \leq \frac{b-M}{R} \times \frac{p-R}{p-r} + \frac{M+C_{tot}}{R} + D_{tot} \leq l \tag{1}$$

$$R \geq \frac{\frac{b-M}{p-r}p + M + C_{tot}}{l - D_{tot} + \frac{b-M}{p-r}} \tag{2}$$

If we consider the slack term S, the result will be given by inequality 3, instead. This equation gives R as a function of S, after fixing the remaining variables.

$$R \geq \frac{\frac{b-M}{p-r}p + M + C_{tot}}{l - D_{tot} - S + \frac{b-M}{p-r}} \tag{3}$$

Notice that it makes no sense to have $l < D_{tot} + S$, since it is not possible to require a latency smaller that a fixed end-to-end delay plus a given slack.

Given these QoS parameters to be used by publish-subscribe applications we have the following problem to solve: how to determine the optimal correspondence between QoS requested by applications and

RSVP resource reservations. By optimal solution we mean a solution that minimizes network utilization based on some criterion while still satisfying application's requests.

For now, we will consider the following input variables to our problem: location of publishers (subscribers) and their respective advertisements [2] (subscriptions), existing QoS parameters both for applications and data network, available IP multicast addresses and, possibly, the data network topology. We keep QoS parameters as an input variable to reduce costs in changing both network infrastructures and APIs.

The desired output necessarily depends on the network infrastructure. In the case of RSVP with Integrated services the output is expressed in terms of RSVP resource reservations. These reservations may belong to unicast or multicast sessions and have specific QoSes also to be determined as solutions to the global problem.

4 Mapping QoS Subscriptions Into RSVP Reservations

In this section we address the problem of determining the mapping between a given configuration of publishers and subscribers (with the associated profiles and filters) and the required RSVP resource reservations.

As noted before, this mapping is established dynamically by the QoS-aware message broker. The IndiQoS message broker is a distributed entity. In the instantiation that is described here, all mapping decisions are established by a centralized control component that executes in a given node of the system. Therefore, in this particular case, the message broker has only two main types of components: local proxies, that execute at every node of the system, and the centralized control module. The message broker proxies are responsible for forwarding to the control module information about advertisements and subscriptions and for establishing the required reservation as instructed by the control module. These actions are transparent to publisher or subscriber applications that use the message broker. The proposed architecture is depicted in figure 2.

We are aware that to rely on a single centralized control module has a number of drawbacks. Not only it represents a single point of failure but it represents a limiting factor for the scalability of the broker. Therefore, in the future we plan to study the implementation of decentralized version of the control component. On the other hand, the use of a centralized control approach in the first prototype provides us with an idea of the quality of the solutions that can be achieved: it is likely that versions using distributed control would approximate, but not reach, the same quality of a solution that relies on global knowledge.

A naive solution for the mapping problem would consist in establishing a point-to-point reservation between any pair of matching publisher and subscriber. Although trivial to implement, such solution would not promote a satisfactory usage of available resources. In fact, when several similar subscriptions occur, it is possible to obtain important benefits in terms of reduced network traffic by supporting several subscriptions using a single IP multicast address.

[2] Keep in mind that advertisements as well as subscriptions now carry QoS information.

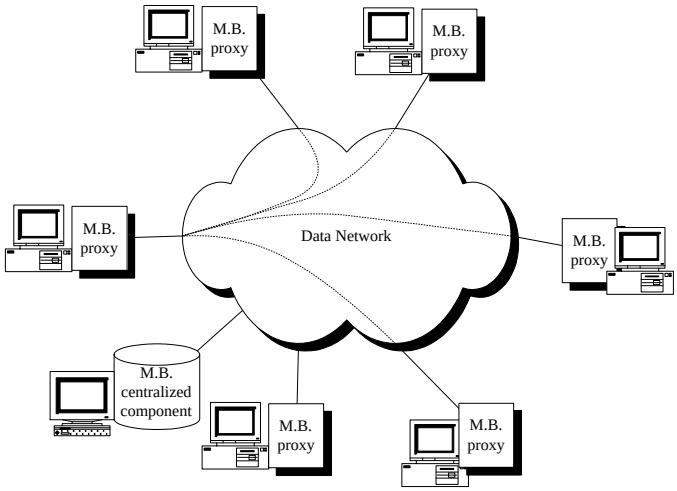

Figure 2: Outline of a centralized publish-subscribe architecture

On the other extreme, one could group all subscriptions on the same multicast address, regardless of the filters specified by each individual subscriber. From the point of view of the number of RSVP sessions, this solution would be optimal since it would promote a maximal share of resources. Unfortunately, this solution is also not satisfactory, because every notification would be sent to all subscribers independently of their subscriptions.

Neither of the two previous extreme solutions (point-to-point for each subscription and single multicast address for all subscriptions) requires a message broker since the mapping is pre-defined. However, the type of solution we are seeking consists in mapping related subscriptions (*i.e.*, subscriptions that share a significant number of common parameters) on a single IP multicast address but to use different addresses to subscriptions that are unrelated. Given that the number of available multicast addresses is typically scarce [7], it is important to adequately distribute the available addresses among all existing subscriptions. To solve this problem, called the "mapping problem", one needs a message broker with a control components, even if not necessarily centralized.

Note that not all subscriptions need to be supported by multicast addresses: it is possible that some subscriptions are better addressed using point-to-point reservations. Subscriptions that are frequently made by a large number of subscribers are better supported by multicast, since there are many opportunities for resource sharing, and subscription that are uncommon are likely to be served by a point-to-point reservation. The advantage of having a message broker is that the applications do not need to be concerned with the optimization issues relevant to the mapping decision. It is up to the message broker to find the appropriate mappings on their behalf. Since the broker tries to optimize the resource usage at the global level (*i.e.*, on behalf of many publishers and subscribers), it may happen that some multicast addresses serve nodes that are not interested in receiving *all* notifications delivered to that address. These extra notifications have to be filtered by the proxy of the message broker at the affected node.

The pairing problem, as described so far has already been addressed in the context of publish-subscribe systems without QoS features [1, 10]. The interesting aspect of addressing this problem in the context of a QoS-aware broker, is that the problem becomes more complex, and new algorithms or heuristics need to be derived. Table 2 shows an example with a number of matching advertisements and subscriptions. Two different sets of QoS parameters are provided as an example. In these two sets, only token bucket rate (r) in *bps* is considered.

Assume that the quality of a solution for the pairing problem is evaluated in terms of wasted bandwidth, measured as the number of bits delivered to nodes that have not registered interest in the corresponding notifications (therefore, these notifications need to be filtered locally by the proxy of the message broker at those nodes). To simplify the presentation, consider that only two multicast addresses are available to support this set of advertisements and subscriptions. One possible solution is depicted in Table 3: this solution wastes 2020 *bps* for the first set and 1290 *bps* for the second set (these values are obtained by summing the rates of all unsolicited notifications for all nodes). Consider now the alternative mapping of Table 4: this solution wastes 7500 *bps* for the first set and 845 *bps* for the second set. This small example clearly shows that the quality of a solution is highly dependent of QoS parameters such as requested bandwidth. Thus, existing algorithms, such as for instance [1, 10], should be adapted (or even replaced) to cope with QoS.

Table 2: Subscriptions

adv. w/ matching subs.	subscriber(s)	set 1	set 2
a	A, D, E	r = 800 bps	r = 20 bps
b	B, C	r = 80 bps	r = 30 bps
c	A, E	r = 500 bps	r = 1000 bps
d	B, C, E, F	r = 20 bps	r = 50 bps
e	D, E	r = 600 bps	r = 60 bps
f	B, C, F	r = 100 bps	r = 25 bps
g	D, E	r = 1000 bps	r = 100 bps

Table 3: Multicast mapping addresses — solution 1

Mulicast address	adv. w/ matching subs.
α = A, D, E	a, c, d, e, g
β = B, C, F	b, d, f

Table 4: Multicast mapping addresses — solution 2

Mulicast address	adv. w/ matching subs.
α = A, E	a, c, d, e, g
β = B, C, D, F	b, d, e, f, g

We are currently working on deriving algorithms to solve the mapping problem in QoS-aware settings. We illustrate the shape of the solution with a very simple mapping strategy. Consider that N multicast

groups are available. The strategy consists in distributing, in a first phase, these N addresses to the N subscriptions that require higher bandwidth. The bandwidth is calculated considering the notification token bucket rate times the number of subscribers. After this first step, the remaining subscription would be mapped on one of the previous addresses or on unicast addresses, depending on the degree of overlapping with the mappings performed in the first phase.

5 Conclusion

This paper has described a particular instantiation of the IndiQoS architecture: a publish-subscribe system where applications may expresses QoS requirements using constructs similar to those used to handle other aspects of the data flow, such as the ones used to express interest in a particular subject or content.

The instantiation addressed in this paper has considered the use of RSVP and integrated services as mechanisms to support the provision of QoS parameters in IP networks. Two main problems need to be solved when building a message broker based on these network level mechanisms. In the first place, we need to define QoS parameters that are suitable for publish-subscribe applications and translate them into integrated services parameters. Second, we need to share network resources to optimize resource consumption while satisfying the requirements of the applications.

When defining the QoS parameters, we have opted to favor simplicity. Hence, we have defined QoS parameters that allow a straightforward mapping into integrated services QoS token bucket parameters. When addressing the resource consumption issue, we have considered the aspect of distributing a limited number of IP multicast addresses by existing subscriptions — the “mapping problem”. This is a known complex problem, that it is further complicated by the presence of QoS parameters. We claim that QoS parameters should be one of the determining factors when finding adequate solutions for the “mapping problem”. As a proof-of-concept, a preliminary solution to the mapping problem is given, but more complex, better solutions will be subject of future work.

References

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

[2] Filipe Araújo and Luís Rodrigues. On QoS-aware publish-subscribe. DI/FCUL TR 02–2, Department of Computer Science, University of Lisbon, February 2002.

[3] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *IEEE Computer*, March 2000.

[4] 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.

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

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

[7] S. Hanna, B. Patel, and M. Shah. Multicast address dynamic client allocation protocol (madcap), December 1999. Request For Comments 2730.

[8] Object Management Group, OMG Headquarters, 250 First Avenue, Suite 201, Needham, MA 02494, USA. *Notification Service Specification*, June 2000.

[9] Object Management Group, OMG Headquarters, 250 First Avenue, Suite 201, Needham, MA 02494, USA. *Event Service Specification*, March 2001.

[10] Lukasz Opyrchal, Mark Astley, Joshua S. Auerbach, Guruduth Banavar, Robert E. Strom, and Daniel C. Sturman. Exploiting IP multicast in content-based publish-subscribe systems. In *Middleware*, pages 185–207, 2000.

[11] Craig Partridge. *Gigabit Networking*. Addison-Wesley, 1994.

[12] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service, September 1997. Request For Comments 2212.

[13] Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303, USA. *Java Message Service*, November 1999.

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