Augmented Reliable Multicast CORBA Event Service (ARMS): a QoS-Adaptive Middleware

João Orvalho[1] and Fernando Boavida[2]

Communications and Telematic Group
CISUC – Centre for Informatics and Systems of the University of Coimbra
Polo II, 3030 COIMBRA – PORTUGAL
Tel.: +351-39-790000, Fax: +351-39-701266
E-mail: {orvalho, boavida@dei.uc.pt}

Abstract. The CORBA Event Service specification lacks important features in terms of quality of service (QoS) characteristics required by multimedia information. The main objective of the work described in this paper is to augment the standard CORBA Event Service specification with a set of extensions, to turn it into an adaptable QoS middleware multimedia framework. To meet this, some extensions to the CORBA Event Service already developed with the aim of providing multicasting and reliability features have been enhanced in order to allow the close interaction with multicast transport protocols and with QoS monitoring mechanisms. The result was a QoS-aware middleware platform that actively adapts the quality of service required by the applications to the one that is provided by the underlying communication channel. The main quality of service features addressed by the platform – and discussed in the paper – are the support of sessions with different reliability levels, the provision of congestion control mechanisms and the capability to suppress jitter.

1. Introduction

A continuous distributed interactive medium is a medium that can change its state in response to user operations as well to the passage of time [1]. A broad variety of applications use this kind of media, such as multi-user virtual reality (VR), distributed simulations, networked games and computer-supported co-operative work (CSCW) applications. Systems dealing with multimedia events combine aspects of distributed real-time computing with the need for low latency, reduced jitter, high throughput, multi-sender/multi-receiver communications over wide area systems and different levels of reliability [2]. These event-driven systems also require efficient and scalable communications components.

[1] College of Education, Polytechnic Institute of Coimbra
[2] Informatics Engineering Department of the University of Coimbra

The work being described in this paper consists of a middleware platform for continuous distributed interactive media applications based on some extensions to the CORBA Event Service. The platform - named Augmented Reliable Multicast CORBA Event Service (ARMS) – provides an end-to-end communication framework with QoS-matching capabilities, having in mind VR application requirements. VR applications have specification features concerning interactivity, reliability, continuity, coherence and strict time constraints that render difficult their traffic characterisation in terms of quality of service (QoS) requirements.

ARMS is focused on VR environments based on VRML models, which deal with two different types of information: events and states. Events are time critical and described by small amounts of information, as opposed to states that are generally non-time-critical and require large amounts of information for their description. Therefore, there is the need for different levels of reliability when exchanging event and state information: minimal reliability (with loss detection) for events, and full reliability for states. In addition, ARMS assumes that there is a common time reference, which requires the clocks of all participants to be synchronised by means of NTP or GPS clocks.

In continuous distributed interactive media there are operations that need to be executed at a specific point in time and in a correct order for consistency to be achieved. [1] investigates this problem and proposes a solution by deliberately increasing the response time in order to decrease the number of short-term inconsistencies, leading to the concept of local lag. Instead of immediately executing an operation issued by a local user, the operation is delayed for a certain amount of time before it is executed. The determination of this value is a typical issue of application adaptability.

VR applications are associated with three important functions: scalability, interaction and consistency [2]. The QoS characteristics that influence the previous functions are reliability, losses and delay jitter. Additionally, those functions are influenced by application adaptability factors like frequency of events, synchronisation delay, number of participants, consistency and playout time (display frequency) [2].

The ARMS platform addresses the above-mentioned QoS issues and explores the adaptation between the quality of service required by applications and the one that is provided by the underlying communication system. This paper presents the main approaches taken by ARMS in order to provide QoS adaptability. Section 2 provides a general description of the ARMS architecture. As the work presented in this paper corresponds to an enhancement of a previous, non-QoS-adaptive platform, section 3 presents the characteristics of this previous platform. Section 4 provides details concerning the approach taken by ARMS in terms of reliability, congestion control and jitter. Section 5 describes additional features of the ARMS platform, namely the IIOP/IP multicasting gateway service and the federation of event channels. Section 6 identifies related work. The conclusions and guidelines for further work are presented in section 7.

2. ARMS general characteristics

Portability over heterogeneous environments is a typical requirement of distributed multimedia systems. Although this portability could be provided by middleware such as CORBA, there exists a widespread belief in the virtual reality community that the quality of service offered by CORBA is not suitable for next-generation large distributed interactive media [3]. Also, another obvious deficiency of CORBA is its lack of support for interaction styles other than request/reply [4]. However, due to their nature, the CORBA Event Service [5] and the CORBA Notification Service [6] have some potential be used for continuous distributed interactive media applications.

Previous research has been focused on limitations of the CORBA Event Service, namely multicasting, reliability and bulk data handling [7,8]. The work presented in this paper extends it to support adaptive QoS middleware functionalities.

ARMS offers a set of QoS-related mechanisms for reliability guarantee, congestion control and jitter control. The QoS management process is supported by object-based monitoring and adaptation functions (Figure 1). Monitoring is the process of observing the utilisation of resources and/or QoS characteristics in the system. ARMS has specific objects for loss and jitter monitoring.

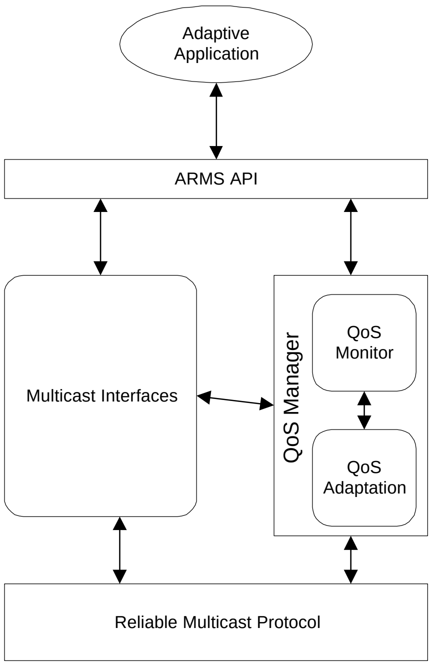

Fig. 1. ARMS architecture

Adaptation mechanisms generally rely on resource control, reconfiguration or change of service [9]. ARMS uses a resource control paradigm, providing adaptation mechanisms for network congestion, losses and jitter. Placing adaptation capabilities in the middleware gives applications the ability to concentrate on specific functionalities, to enforce different adaptation policies and to interact with other components in the system in order to ensure fairness and other global properties [10].

3. Synopsis of ARMS version 1

The first generation of ARMS offered a set of extensions to the CORBA Event Service specification [7], namely mechanisms for IP multicast communication, reliability and fragmenting/reassembling of large events (Figure 2).

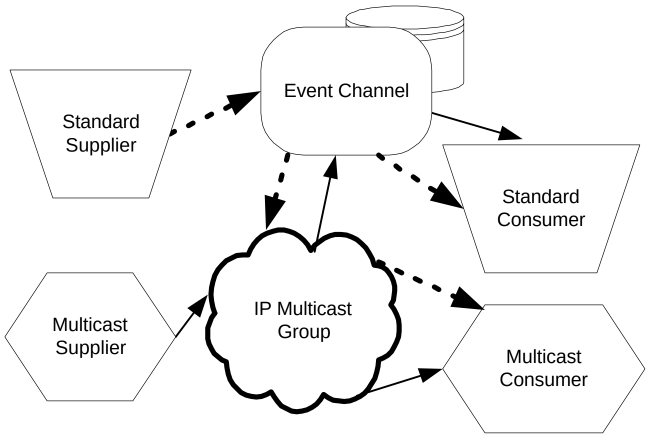

Fig. 2. ARMSv1 high-level view

This service is a pure Java implementation (JDK 1.2 compatible) [11], and so it benefits from all the strengths of Java. These include portability, security, and robustness. It is, also, ORB-agnostic. It is written to the standard IDL-to-Java mapping, so it should work with any Java ORB that supports the standard mapping. As in the original specifications made by OMG CORBA Event Service, suppliers and consumers are decoupled, that is, they don’t know each other’s identities; thus, the standard Event Service may still be provided by this extended service.

The approach on which the developed work was based is the one called *Push* model. The objective is to allow the consumer to be sent the event data as soon as it is produced. The canonical Push Model allows Suppliers of events to initiate the transfer of event data to Consumers.

The reliable multicast extension can be seen as an alternative way to get the event across to the consumer. This assumption forces the new service to keep all the standard interfaces with the same functionality (the same methods) that are defined by OMG, allowing a choice to be made by the supplier/consumer between IIOP and Reliable IP Multicast. The service implements two kinds if IP multicast interfaces: IP Multicast-Any and IP Multicast-Streams. The IP Multicast-Any deals with *Any* Values while IP Multicast-Streams deals directly with byte-stream values, avoiding the overhead caused by marshalling and de-marshalling of the proprietary *Any*. The reliable multicast solution is based on the Light-weight Reliable Multicast Protocol

(LRMP) [12], which deals with IP Multicasting and provides the necessary reliability and better scalability (Figure 3).

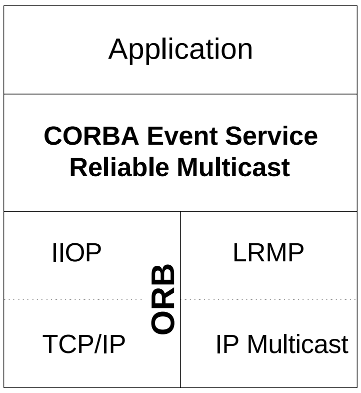

Fig. 3. ARMSv1 architecture

4. ARMS version 2

The quality of service provided to multimedia sessions is determined, in general, by packet losses and delay. The second generation of ARMS acts as a transport level with QoS monitors that provide information on key aspects of QoS provision such as losses and jitter. Adaptation mechanisms for network congestion (congestion control), losses (reliability sessions) and jitter (jitter filter) have an important influence on the performance of virtual environment applications.

4.1 Reliability Sessions

ARMSv2 supports several reliability levels: Reliable, Reliable-Limited Loss, Loss Allowed, Unreliable with loss notification and Unreliable.

The first is a typical strong reliability session. Reliable-Limited Loss is a session where limited losses are permitted for some types of packets and loss notifications will be triggered when this happens. It provides guarantees for congestion control at sender side and sequence control at the receiver side. Loss Allowed is a session where losses are allowed and accepted for all types of packets but provides loss notification, congestion control at sender side and sequence control at the receiver side. Unreliable with loss notification is an unreliable session, where data are not subject to congestion control at the sender side. However, the ARMS upper level maintains a queue for sequence number control, which allows loss notification. Lastly, Unreliable is a purely unreliable session. Table 1 summarises the characteristics of the various types of sessions offered by ARMSv2.

Table 1. Types of ARMSv2 sessions

	Permitted Losses	Loss Notification	Congestion Control	Sequence Control
Reliable	No	Yes	Yes	Yes
Reliable-Limited Loss	Yes, for some data types	Yes	Yes	Yes
Loss Allowed	Yes	Yes	Yes	Yes
Unreliable with loss notification	Yes	Yes	No	Yes
Unreliable	Yes	No	No	No

4.2 Congestion Control

Currently, different approaches are being discussed to solve QoS and congestion control problems [13]. Complex distributed applications residing in heterogeneous end-to-end computing environments must be flexible and adapt to QoS variations in their end-to-end execution [10]. That is, applications must adapt the bandwidth share they are using to the network congestion state [13]. Usually, there are two distinct levels at which adaptation may take place – the system level (e.g. operating systems and network protocols) and the application level – with different objectives. In order to balance the objectives of these approaches, the ARMS middleware closely interacts both with application needs and with multicast protocols, monitoring network parameters and operating systems resources.

In terms of network QoS control mechanisms, ARMS directly monitors the reliable multicast communication protocol, LRMP, adapting the sender transmission rate to the network congestion state. The adaptation is based on information carried by NACK packets and on local congestion information [8] as the sender window size [12]. Based on congestion information gathered from lower communication objects, ARMS and applications adjust the upper-level sending rate.

QoS mechanisms that are based on adapting the sender transmission rate to the network congestion state don’t work well in large multicast groups and heterogeneous environments, because poor performance receivers would impose a low transmission quality. To avoid this, several proposals have been made for hierarchical data distribution [14,15,16]. Nevertheless, in virtual reality environments, data layering approaches are not appropriated for most data, especially for time critical data such as VRML events. Delay is the most important QoS factor for this type of data. Layered data mechanisms solve heterogeneity problems but cause additional delay at the receivers [13]. Thus, to avoid this, ARMS adapts the minimum rate to ensure the fairness of the adaptive congestion control. Receivers should leave the session when the loss rate is very high and the data rate in not reduced by the sender. Nevertheless, layered multicast [14] can be useful and will be explored in subsequent stages of the work.

4.3 Jitter

Variance in end-to-end delay is called delay jitter or, simply, jitter. Critical information such as the case of audio, video and continuous distributed interactive media (distributed simulations, multi-user virtual reality) should be played back continuously, which means that there must be some form of jitter compensation.

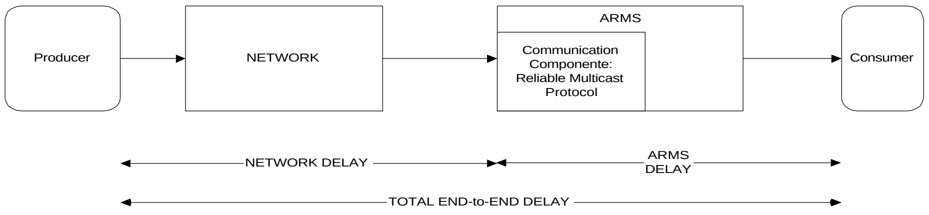

Fig. 4. Components of end-to-end delay

In addition to components for error control, ARMS provides mechanisms for monitoring and controlling jitter (Figures 4 and 5), so that the original temporal relationships can be recovered. ARMS lower level reliable multicast protocol queue sends ordered packets (reliable data) to an upper level Jitter Filter Queue. This Jitter Filter Queue is used to absorb delay variations exhibited by arriving packets. So, this compensation is done by introducing an additional and variable delay, followed by the delivering of packets to the application level, as opposed to a compensation made at the application level [17].

ARMS allows applications to contract this jitter compensation [the threshold synchronisation] for certain types of data, namely unreliable data (pure or with loss notification) and reliable data with limited loss. Packets that arrive after a given threshold are considered to be too late. These can simply be dropped or, alternatively, be marked as late packets and passed to the application. Applications can ignore these packets or can react by requesting that a special NACK be sent by ARMS.

Jitter Algorithms

ARMS implements the Filter Jitter with two different algorithms. The choice of which algorithm to use is made at configuration time. The first algorithm is described in the following paragraphs.

To recover the original timing properties, the Jitter Filter buffers (Figure 5) the packets at the sink until time T + D, where T is a source timestamp and D is the bounded maximum end-to-end delay. When networks are unable to guarantee a maximum end-to-end delay bound, the receiver continuously updates an estimate of the maximum delay in order to calculate the buffering time. One of algorithms that can be used to calculate D is based on the RTP specification [18] to estimate the statistical variance of RTP data packet inter-arrival time, measured in timestamp units and expressed as unsigned integer. At a given instant, D is the maximum of all jitter values calculated up to that instant according to the formula:

$$J_i = J_{i-1} + (|Dif(i-1, i)| - J_{i-1})/16$$

where the inter-arrival jitter *J* is defined to be the mean deviation of the difference *Dif* in packet spacing at the receiver compared to the sender for a pair of packets [18]. This is equivalent to the difference in the “relative transit time” for the two packets. So, *Dif* may be calculated as

$$\mathrm{Dif}(i,j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i)$$

S_i is the reliable multicast protocol timestamp for packet *i* and R_i is the time of arrival in reliable multicast protocol timestamp units for packet *i*, the same for packet *j*. Jitter is calculated continuously as each packet is received, for each source. Factor 1/16 was chosen to reduce measurement noise while converging reasonably quickly [18,19]. The code below implements the algorithm, where the estimated jitter can be kept as an integer.

```
protected void updateJitter(int timestamp) {
        int elapsed = NTP.ntp32(lastTimeForData - timestamp);
        elapsed = NTP.fixedPoint32ToMillis(elapsed);
        int d;
        if (transit != 0)
            d = elapsed - transit;
        else
            d = 0;
        transit = elapsed;
        if (d < 0)
            d = -d;
        jitter += d - ((jitter + 8) >> 4);
    }
}
```

The second jitter algorithm is based on the work of [20], where statistical analysis of per-packet delay is used to estimate the maximum delay:

$$D = d + r * s$$

where *d* is the average delay, *s* is the standard deviation and *r* is a filter coefficient [20]. The algorithm continuously estimates the average delay and standard deviation, and is based on the ‘low pass filtering algorithm’ used in TCP for the estimation of the acknowledgement delay time [20]. So, for each packet, the transmission delay, *t*, is calculated as the difference between the reception time and the emission timestamp. The average delay and standard deviation are the calculated as $d = d_{old} + a(t - d)$ and $s = s_{old} + b(|t - d| - s)$, respectively. The constants *a* and *b* (a,b <1) are smoothing coefficients, with the typical values 1/8 and 1/16, respectively.

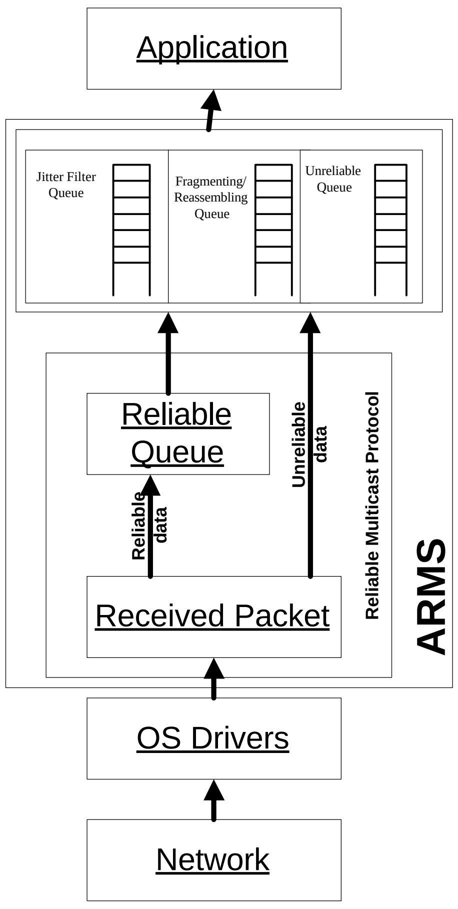

Fig. 5. Diagram of Data Flow

5. Additional features of the ARMS platform

In addition to providing QoS adaptation, the ARMS platform has some optimisation features that contribute to its transparency and scalability. With regard to transparency, the platform provides a gateway service between IIOP and IP Multicasting. Scalability is supported by the federation of Event Channels.

5.1 Event Channel IIOP/IP Multicasting Gateway Service

ARMS guarantees interoperability between event suppliers and consumers in a way that is independent of the used communication facilities – standard IIOP or IP multicast. To achieve this, the Event Channel provides a transparent IIOP gateway.

As can easily be understood, the interfaces that are provided by OMG must be maintained by the proposed service in order to deal with the *Any* type of data. In this way, interoperability between the proposed service and most of the commercially available ORBs can be achieved. In order to deal with all the values supported by the *Any* type, a new object – Any – was created as an extension of the OMG class, to override the ORBs implementation.

The values to be passed to this service, in order to be sent via multicast, shall be created with the proprietary interfaces of the service, that provide a set of new methods to deal with the resulting buffers and fill the LRMP data packet with the data contained in those buffers. So, by creating a method for extracting the sequence of bytes generated by the *Any* marshalling operation, the result is a stream of bytes that mean nothing to LRMP. Once they get across to the consumer LRMP object, it reassembles them to form an *Any*, and delivers them to the consumer as a valid *Any* that shall be extracted by the application interfaces.

To provide the interchange of Multicast events and Standard events, two scenarios are considered:

- when the supplier is a *MulticastPushSupplier* and the consumer is a normal Consumer;
- when the supplier is a normal Supplier and consumer is a *MulticastPushConsumer*.

For the first scenario, and knowing that the model to be dealt with is the Push Model, the event is sent via LRMP, i.e. is sent to a well known multicast group, and is pushed to Multicast Consumers by one LRMP object that receives events directly. As the Event Channel holds one *ProxyMulticastPushConsumer* for all Suppliers, this proxy acts as a normal consumer to which events are pushed. Each time this proxy is pushed a new event, it invokes the receive method on the Event Channel. This invokes a receive method on the *ConsumerAdmin*, that holds references to all *ProxyPushSuppliers* (standard) and shall invoke the receive method on all those proxies, that will then invoke the push method on the consumer they are attached to. The *ConsumerAdmin* does nothing on the *ProxyMulticastPushSupplier*, or else events would be sent twice to the multicast group.

For the second scenario, the supplier calls the push method on its *ProxyPushConsumer*, that will call receive on the Event Channel, to get the event to standard consumers, and will also ask for any existent *ProxyMulticastPushSuppliers*. If there are any, the proxy shall call the receive method on that proxy. In this way, the event is sent to the multicast group, and all *MulticastPushConsumers* receive it. So, the Event Channel is responsible for doing all the necessary IIOP gateway work.

In either scenario, the gateway operation implies changes to the Any Object that is being forwarded. The Event Channel also takes care of this detail, at the cost of an additional overhead caused by the de-marshalling of the proprietary Any and subsequent marshalling as an ORB *Any*, and vice-versa.

5.2 Federation of Event Channels

Distributed transparency of the Event Channel can lead to a less effective configuration. There are scenarios where consumers and suppliers reside in the same process, host or network and the Event Channel is remote. In these cases, there is a waste of network resources and unnecessary increase of latency. ARMS Event Channel object uses a configuration facility to federate Event Channels, allowing an Event Channel to be a consumer of another, remote Event Channel. Federation of Event Channels leads to the conservation of bandwidth because only a single event will be sent to all the remote users. Additionally, the average latency is reduced because part of the traffic becomes local. Figure 6 illustrates the use of the federation of event channels when communication is made via standard IIOP. In Figure 7 federation of event channels is used in conjunction with the IIOP/IP Multicast Gateway Service.

The combination of IIOP/IP Multicasting Gateway Service with Event Channel federation contributes to the enhancement of the QoS characteristics of the platform, namely in terms of transparency, latency and scalability.

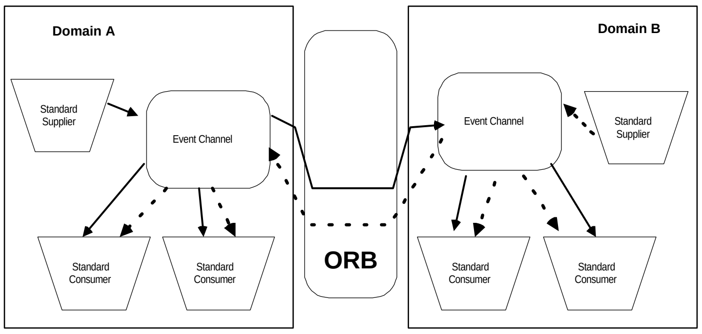

Fig. 6. Federated Event Channel configuration: standard IIOP

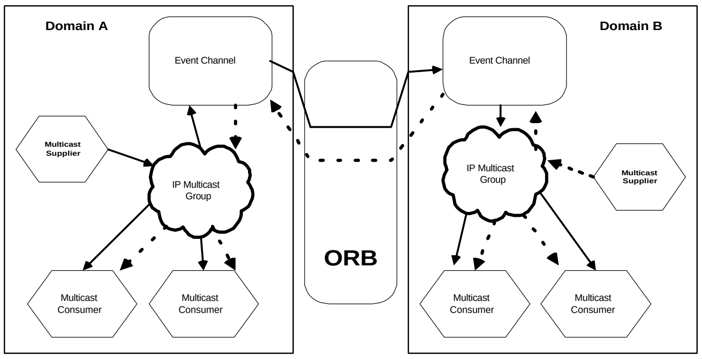

Fig. 7. Federated Event Channel configuration with IIOP/IP Multicasting Gateway Service

6. Related work

Several approaches exist that try to explore QoS issues in distributed object environments.

OrbixTalk [21,22], which is a commercial IONA implementation of the CORBA *Event Service* specification normalised by OMG [5] written in C++, uses IP multicasting and a reliability mechanism based on *negative acknowledgements*, in order to provide delivery confirmation of each information object. Because the CORBA *Event* Service specification does not address issues for real-time applications, the QoS behaviour is not acceptable for many application domains.

OMG Notification Service [6] is a superset of the CORBA *Event Service* specification. It adds some interfaces, which deal with filtering, security, federation and QoS. However, this specification does not address implementation issues.

TAO's Real Time Events Service [3] it's a powerful implementation of CORBA *Event Service and Notification Service* specifications in C++. Nevertheless, it lacks some QoS characterisitcs, such as reliability. However, OMG CORBA Messaging specification [23] defines several levels of reliability for one-way calls, which are being added to the TAO features to complement this service. This group has been developing considerable work on real-time extensions to CORBA, which enable end-to-end QoS specification and enforcement.

Other CORBA projects, such as QuO [24], implement models for distributed object application, defining, controlling, monitoring and adapting to changes in QoS parameters. The QuO project proposes various extensions to standard CORBA components and services, in order to support adaptation, delegation and renegotiation services to shield QoS variations. The development has a great focus on remote method invocation to remote objects.

ARMS is complementary to the above-mentioned approaches, since it is based on the integration of CORBA middleware and underlying services – such as multicasting – while the referred approaches are concentrated on the CORBA object model.

7. Conclusions and guidelines for further work

Quality of service requirements of continuous distributed interactive media applications encompasses several aspects that are not readily available in standard distributed object platforms. The need for different levels of reliability, congestion control mechanisms and jitter suppression strategies is apparent in applications such as virtual reality, CSCW and distributed interactive simulations.

Middleware platforms can play an important role in quality of service provision, offering flexible mechanisms that adapt the quality of service provided by the underlying communication channel to the quality of service required by applications, according to an established service contract.

This paper presented the main implementation options of the ARMS middleware platform, which builds on a set of extensions to the CORBA Event Service, providing native multicast communication, various reliability levels, congestion control and jitter suppression, with the aim to achieve QoS adaptability. The platform has been implemented at the Laboratory of Communications and Telematics of CISUC, where it is operational and currently being subject to extensive evaluation.

The first set of tests made to the platform was aimed at the verification of the platform operational status. The basic mechanisms for reliability, congestion control and jitter proved to be operational. The following tests will try to quantify the usefulness and effectiveness of these mechanisms. These will provide valuable information concerning the adequacy of placing QoS adaptation mechanisms in the middleware, as opposed to strategies that place them at application level.

Additional lines of research will explore the use of Forward Error Correction (FEC) mechanisms in order to provide some degree of reliability to time critical information, and the use of event filtering based on IP multicast groups in order to improve scalability. This latter functionality will be developed in a centralised service, which will map different event types to different IP multicast groups.

In heterogeneous environments, layered multicast can be useful. This will also be explored in subsequent stages of the ARMS platform, namely by dynamically assigning receivers to multicast groups with different QoS levels, according to the required quality of service.

Acknowledgement

This work was partially financed by the Portuguese Foundation for Science and Technology, FCT.

References

[1] "Consistency in Continuous Distributed Interactive media", Martin Mauve, Technical Report TR-9-99, Reihe Informatik, Department for Mathematics and Computer Science, University of Mannheim, November 1999.

[2] "Quality of Service Management in Distributed Interactive Virtual Environment", Dimitrios Makrakis, Abdelhakim Hafid, Farid Nait-Abdesselem, Anastasios Kasiolas, Lijia Qin, Progress Report of DIVE project.
http://www.mcrlab.uottawa.ca/research/QoS_DIVE_Report.html

[3] "Applying a Scalable CORBA Event Service to Large-scale Distributed Interactive Simulations", Carlos O'Ryan, David L. Levine, Douglas C. Schmidt, J. Russel Noseworthy, Proceedings of the "5Th Workshop on Object-oriented Real-time Dependable Systems", Montery, CA, November 1999.

[4] "Supporting Multimedia in Distributed Object Environments", D. G. Waddington and D. Hutchinson, *Internal report number MPG-99-17,* 1999.

[5] "CORBA services: Common Object Services Specification", Object Management Group-OMG, Document formal/97-07-04, ed., July 1997.

[6] "Notification Service Specification", Object Management Group-OMG, Document telecom/99-07-01, ed., July 1999.

[7] "A Platform for the Study of Reliable Multicasting Extensions to CORBA Event Service", João Orvalho, Luis Figuiredo, T. Andrade and Fernando Boavida, Proceedings of IDMS'99, Toulouse, France, October 12-15, 1999.

[8] "Evaluating Light-weight Reliable Multicast Protocol Extensions to the

3rd International Enterprise Distributed Object Computing Conference, University of Mannheim, Germany, September 27-30, 1999.
[9] "A General Model for QoS Adaptation", D. G. Waddington and D. Hutchinson, Proceedings of Sixth International Workshop on Quality of Service (IWQoS'98), pag.275-277, Napa, California, USA, May 18-20 1998.
[10] "A Control-Based Middleware Framework for Quality-of-Service Adaptations", Baochum Li and Klara Nahrstedt, IEEE Journal On Selected Areas In Communications, Vol. 17, Nº 9, September 1999.
[11] JavaSoft, www.javasoft.com
[12] Tie Liao: "Light-weight Reliable Multicast Protocol Specification", Internet-Draft: draft-liao-lrmp-00.txt, 13 October 1998.
[13] "The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme", Dorgham Sisalem and Henning Schulzrinne, In Proceedings of Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), Cambridge, UK, 1998.
[14] "TCP-like Congestion Control for Layered Multicast Data Transfer", L. Vicisano, L. Rizzo, and J. Crowcroft, In Proceedings of INFOCOM'98, San Francisco, USA, March 1998.
[15] "Receiver-driven layered multicast", S. McCanne, V. Jacobson, and M. Vertteli, In Proceedings of SIGCOMM Symposium on Communication Architecture and Implementation of High Performance Communication Systems (HPCS'97), Chalkidiki, Greece, June 1997.
[16] "Thin Streams: An Architecture for multicasting layered video", L. Wu, R. Sharma, and B. Smith, In Proceedings of Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'97), St. Louis, USA, May 1997.
[17] "Delay and Synchronization Control Middleware to Support Real-Time Multimedia Services over Wireless PCS Networks", H. Liu and M. El Zarki, IEEE Journal On Selected Areas In Communications, Vol. 17, Nº 9, September 1999.
[18] "RTP: A transport Protocol for Real-Time Applications", revision of RFC 1889, Internet-Draft (draft-ietf-avt-rtp-new-04.ps), Schulzrinne, Casner, Frederic, Jacobson, June 25 1999.
[19] "Speech Transport for Packet Telephony and Voice over IP", Maurice R. Baker, In Proceedings of SPIE Conference on Internet II: Quality of Service and Future Directions", Boston, Massachusetts, September 1999.
[20] "QoS Adaptive Transports: Delivering Scalable Media to the Desk Top", Andrew T. Campbell and Geoff Coulson, *IEEE Network*, Vol. 11 No. 2., pg. 18-27, March/April 1997.
[21] OrbixTalk da Iona, www.iona.com/
[22] "Reliable CORBA Event Channels"X. Défago, P. Felber, R. Guerraoui, EPFL, Computer Science Department, Technical Report 97/229, May 1997.
[23] "CORBA Messaging Specification", Object Management Group-OMG, Document orbos/98-05-05, ed., May 1998.
[24] "QuO's Runtime Support for Quality of Service in Distributed Objects",

Vanegas R, Zinky JA, Loyall JP, Karr DA, Schantz RE, Bakken DE.. Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, England, 15-18 September 1998.