Middleware Support for Media Streaming Establishment Driven by User-Oriented QoS Requirements

Cristian Hesselman[1], Ing Widya[2], Aart van Halteren[3], and Bart Nieuwenhuis[2]

[1] Telematica Instituut, Enschede, The Netherlands
[2] CTIT, University of Twente, Enschede, The Netherlands
[3] KPN Research, Enschede, The Netherlands
hesselman@telin.nl, widya@cs.utwente.nl,
A.T.vanHalteren@kpn.com, L.J.M.Nieuwenhuis@cs.utwente.nl

Abstract. The requirements for the QoS of distributed applications are traditionally expressed in terms of network oriented or systems oriented parameters. In general, the users of these services are not interested or capable of specifying the QoS of their services in such technical terms. In this paper, we propose modeling and engineering concepts for the mapping of end user QoS onto system and network QoS. We introduce QoS agents in structured object middleware that relate end-user QoS specifications to multimedia stream bindings. In fact, the middleware layer supports QoS classes, i.e., a set of QoS characteristics. The end user QoS requirements, generally a set of non-orthogonal specifications, must be supported using the available middleware QoS classes. We also describe the experimental environment that will be used to refine the QoS mapping mechanisms.

1. Introduction

Object middleware, such as CORBA, DCOM and Java RMI, is gaining rapid acceptance as a means to quickly and cost effectively develop a wide range of applications for various areas of industry. The main purpose of these middleware systems is to provide a software infrastructure for interacting application components.

Interactions between software components can be divided into two categories: discrete or continuous. Discrete interactions are typically RPC or message-oriented and are generally used to invoke a computational service. Continuous interactions are generally used for exchanging (multi-) media data and are usually called streams. Middleware platforms often model a stream as a binding object [Gay, Blair]. In this paper we focus on the support of middleware platforms to establish a user-oriented QoS for such binding objects.

Requirements for QoS are traditionally expressed in terms of system-oriented or network-oriented parameters. In this paper we propose a slightly different approach. We start from user-oriented requirements for QoS and identify where and how the middleware can translate them to system or network-level requirements for QoS. We define the architectural concepts that we think are necessary for understanding and decomposing the complexity of stream binding establishment. We further identify

H. Scholten and M. van Sinderen (Eds.): IDMS 2000, LNCS 1905, pp. 158-171, 2000.

how current object middleware systems can be extended to support media stream binding.

Section 2 describes a framework for QoS-aware middleware. Section 3 identifies the interfaces for QoS specification and section 4 describes how user-oriented QoS specifications can be mapped to network and system level QoS. Section 5 presents how our framework can be applied to an implementation of the CORBA A/V Streams specification. Section 6 summarizes with our conclusions.

2. A QoS Framework for Streaming

In this section, we propose a framework for QoS-aware middleware-based distributed systems that combines objects, layers and planes. We have used the framework to structure the QoS-related functions of a distributed system. This framework is currently being validated in our testbed.

In Section 2.1, we present the object model we use. In Section 2.2 we discuss how the object model fits into our framework from a high level point of view. In Section 2.3 we consider the framework in more detail, in particular in terms of its layers and planes.

2.1 Object Model

Fig. 1 shows the kind of objects that we use to structure our system [Blair]. Each object encapsulates state and behavior and can expose operational and streaming interfaces to other objects.

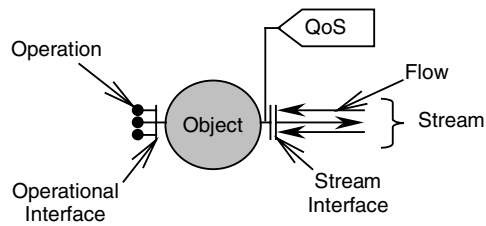

Fig. 1. An object with operational and streaming interfaces.

Operational interfaces allow client objects to invoke computational services onto the object that exposes the interface (the server object). Streaming interfaces allow objects to exchange one ore more flows of continuous information (e.g. audio or video information). Flows are unidirectional and may terminate at one or more sink interfaces. A stream may consist of flows that travel in opposite directions.

Objects may further consist of several other objects at a lower level of abstraction. The higher level object is called a compound object in such cases.

2.2 QoS-Aware Middleware Platform

We propose a middleware platform that is capable to associate a QoS with an object's streaming interface and to control this QoS through the same or another object's computational interface. We propose to extend existing middleware concepts by a QoS framework. We elaborate this framework in particular for middleware that supports multimedia applications using video and audio streams.

The application objects, e.g., cameras, microphones, speakers, files, are endpoints of audio and video stream. The application layer furthermore contains agent objects that act as stand-ins for the end-user (cf. [Jmf]). Agents invoke the services of our middleware platform to bind endpoint objects and to control the QoS of the stream that the binding carries. Agent objects also control the QoS of local endpoint objects.

A binding object allows endpoint objects in the application layer to exchange a stream of multimedia information. A binding object represents a composition of the resources that are involved in tying these application level objects together. This includes media processing resources like codecs, multiplexers, transcoders and packetizers, as well as transport-level resources such as sockets, routers and bridges.

Example
Consider a medical application that allows surgeons to view video clips stored in a database. We assume that a request to connect to the video database originates from the surgeon's machine. Fig. 2 shows the object constellation once the surgeon is viewing a video clip.

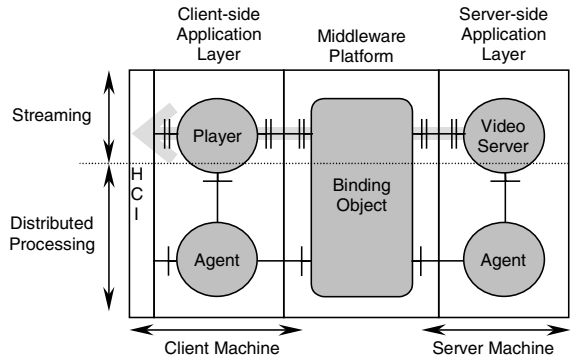

Fig. 2. Binding object interconnecting two application objects.

The video server object in Fig. 2 represents the database. It produces an audio-video stream that flows to the player object via the binding object. The player represents the presentation resources on the surgeon's machine (a display and a speaker in this case) and consumes the stream that the binding produces.

The two agent objects use the operational interfaces of the player, the video server and the binding object to control the QoS of the streams that these objects produce. The client-side agent object may for instance use the binding's operational interface to request the establishment of a certain QoS for the binding or to change the QoS of

an existing binding. Similarly, the agent can call the player's operational interface to locally reduce the volume of the video clip's audio part.

The operational interfaces of the binding object have two special properties. First of all, their operations are *application oriented*. This means that the binding object allows the agents to deal with QoS in application-level terms. For example, the binding object allows the client-side agent to specify that it wants the player object to be bound to the video server object using a binding object with an High Definition TV QoS level. As a result, the agent objects do not have to be able to translate this abstract notion of QoS into low-level QoS aspects such as bandwidth, delay, and so on.

The second property is that the binding object's operational interfaces are geared toward the context in which the application is used. We call this the application's *usage context*. This notion is based on our belief that different applications require a different QoS depending on the domain in which they are used, for what purpose the application is used and by whom. For example, if a surgeon uses the system of Fig. 2 to discuss a video clip with one of his fellow surgeons, the client-side agent object will generally request a higher QoS from the binding object than when the surgeon discusses the same clip with one of his patients. We will have more to say on the application-orientation of the binding and our notion of a usage context in Section 3.

2.3 The Platform's Internals

The binding object of Fig. 2 may encapsulate a large number and a variety of resources It may furthermore need to perform complex QoS control activities. We tackle this problem by decomposing the platform into two horizontal layers and three vertical planes. The vertical planes are a data transfer plane, a QoS control plane and a QoS management plane (cf. [Aurre]). Across these planes we distinguish between a middleware (software) layer and the Distributed Resource Platform (DRP) layer (i.e. computing and network resources).

Data Transfer

The data transfer plane contains the objects that are concerned with forwarding the data units of a multimedia stream.

An object in the data transfer plane of the middleware layer encapsulates resources that perform transport-independent as well as transport dependent stream processing. Examples of the former are encoders, transcoders and multiplexers; an example of the latter is an RTP packetizer that can adapt an MPEG-1 encoded stream for transmission over UDP.

The objects in the data transfer plane of the DRP encapsulate the distributed resources (e.g. IP routers, bridges, etc.) that provide end-to-end connectivity.

QoS Control and Management

The objects in the QoS control and management planes of the middleware layer and the DRP govern the QoS of the stream that flows through the data transfer plane.

Each binding object encapsulates a set of objects in the QoS control plane of the middleware layer and the DRP. These objects govern the QoS of the binding's streaming interfaces during its lifetime. We propose that objects in the QoS control

plane are responsible for *establishing* a QoS for a binding. The establishment of QoS typically involves the negotiation of an acceptable QoS followed by the reservation and initialization of objects in the data transfer plane to effectuate the negotiated QoS. Other activities that can be found in the QoS control plane involve [ISOQoS, D3.1.2] predicting a binding's current and near-future QoS, keeping a binding's current QoS in line with the negotiated QoS, and releasing a binding and its resources.

The objects in the QoS management plane of the middleware layer and the DRP are not part of a particular binding. Rather, they can be considered part of every binding because their activities transcend the lifetime of individual bindings. Objects in the QoS management plane for instance take care of fault management and statistics collection.

Example

As an example, assume that the binding object of Fig. 2 uses an MPEG-1 encoder to compress the audio-video stream. Also assume that the binding relies on an RSVP reserved UDP transport channel to convey the encoded stream from the server to the client. Fig. 3 shows how the example of Fig. 2 maps onto the layers and the planes of our QoS framework. Observe that it zooms in on the middleware portion of Fig. 2. Fig. 3 does not show the application layer and the objects that it hosts.

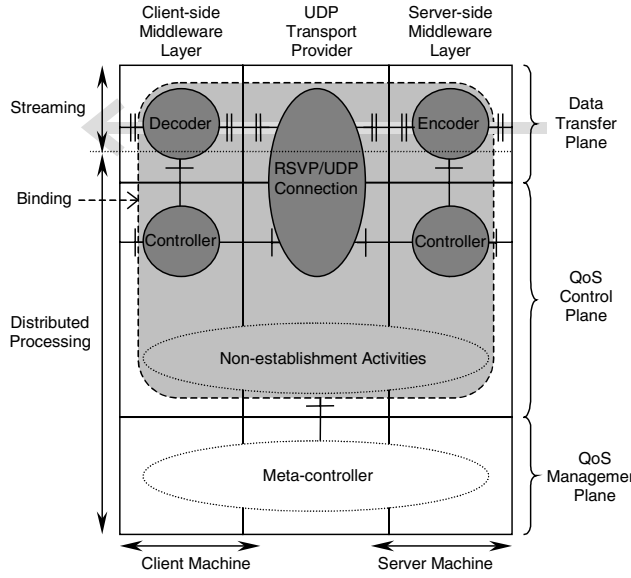

Fig. 3. Two-dimensional version of the QoS framework.

The encoder and decoder are middleware level objects of the data transfer plane. The encoder for instance encapsulates the MPEG-1 encoder and an RTP packetizer. The RSVP/UDP connectivity object is part of the data transfer plane of the DRP and encapsulates the resources that connect the encoder to the decoder.

The controller objects are part of the middleware layer's QoS control plane. They use the operational interfaces that the encoder, decoder and connection objects expose to control the QoS of the streams that these objects produce. The controllers may for instance first negotiate an encoding and a packetizer to use. The controller on the server-side can then invoke the operational interface of the selected encoder, for example to set its output bitrate parameter. After that, the controller can use the operational interface of the connection object to request a QoS that supports the encoder's output. Observe that the notion of QoS at the connection object's operational interface is of a lower level than that at the binding object's operational interface. The former typically expresses its notion of QoS in terms of bandwidth, delay, jitter, and so on, whereas the latter uses application-oriented notions such as "HDTV QoS".

Observe that the objects in the distributed processing part of Fig. 2 and Fig. 3 build on an ORB infrastructure therefore they can invoke each other's operational interfaces even when they reside on different machines. For the sake of clarity, we have not drawn these interactions.

Also observe that Fig. 3 represents non-establishment activities of the QoS control plane (e.g. checking the current QoS against the negotiated QoS) as a single object. The activities of this object are outside the scope of this paper and we have therefore not shown its interactions with the other objects in Fig. 3. For similar reasons, we have also not shown how the object that represents the management activities (the meta-controller object in Fig. 3) relates to the objects in the binding.

3. QoS at Interfaces

In the previous section, we presented our QoS framework. In this section, we elaborate on the specification of QoS at interfaces and the mapping of these specifications to QoS support of the middleware. We define a QoS for a streaming interface by specifying the desired QoS at an operational interface

We use the QoS specification language QML [Frølund] to illustrate our approach.

3.1 Control-Plane QoS Interface Model

In our framework (see Section 2), QoS represents the quality aspects of the interactions between the middleware platform and one or more application objects that act as the endpoints of a stream. QoS realizes the quality aspects of the interactions between these application objects and thereby improves the usability or the utility of the applications that use the platform.

The concepts that we use to specify QoS are based on the user-provider model [ISOQoS, D3.1.2] (see Fig. 6) in which a provider (our middleware platform) mediates the interactions between its users (our application objects).

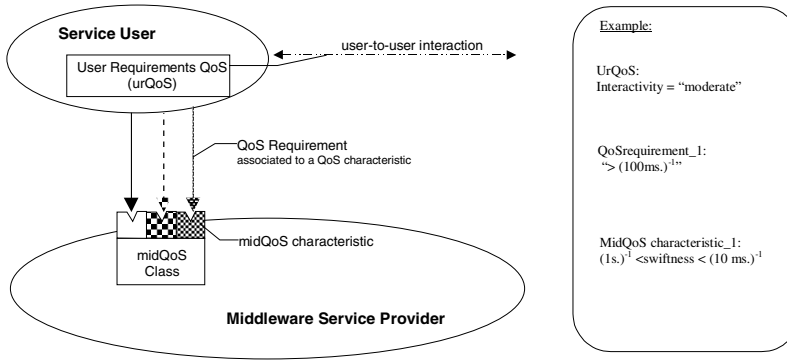

Fig. 4. Control-plane QoS Interface Model

From a top-down perspective, the quality needs of the interactions of application objects drive the QoS that is required at streaming interfaces. These quality needs are called *user requirements* [ISOQoS] with respect to QoS (urQoS). In principle, urQoS is independent of the service provider that mediates the interaction.

The provider may advertise its QoS capabilities in terms of the values that it supports for a certain set of *QoS characteristics* (midQoS characteristics in Fig. 4). QoS characteristics are the quantifiable aspects of QoS such as accuracy, freshness or reliability. The provider's capabilities are in principle also defined independently from the users. Aggregations of QoS characteristics and their associated values are called *QoS classes*. A provider may support several QoS classes. Each class may be optimized for a certain category of services. Examples are QoS classes for real-time services, messaging services, or multimedia services. For reasons of flexibility, a provider may also offer QoS classes of different abstraction levels. QoS classes may for instance be network-oriented or application oriented.

To specify a QoS at a provider's interface, we need to translate urQoS (including the associated values needed) to an instance of a QoS class that the provider offers. To facilitate this, we use the concept of a *QoS requirement* [ISOQoS]. A QoS requirement consists of required values of a QoS characteristic and the qualifiers for these values (e.g. maximum, mean, or minimum).

3.2 User Requirement

The urQoS of Fig. 5 typically depends on what we call the *application context of use*. We have analyzed a number of reports (see for instance [Ewos]) that describe the properties of multimedia data interchange in the medical domain. The reports cover user scenarios as well as the media characteristics and alternative stacks of protocols (including the compatible parameter options) that are suitable for these scenarios. The reports are suitable for our work because communication experts as well as medical specialists have contributed to them.

Our analysis leads to the following elements as determining an application's usage context:

- the application domain: examples are the medical domain of surgery and the game-hall domain;
- the role of the user: examples are imaging specialist, surgeon, game-hall subscriber or guest user;
- the purpose for which the application is used: for example, a surgeon may use a moving image retrieval application to check his diagnostic hypothesis or to view images during surgery. Latency requirements of image retrieval may for instance be different for both cases.

We have furthermore identified the following urQoS dimensions: availability, fidelity, integrity, interactivity, and regulatory. Since these dimensions are not orthogonal the translation of urQoS dimensions to provider-oriented QoS characteristics (midQoS) is not straightforward.

Table 1. User-oriented QoS dimensions

QoS dimension	*Quality aspect*
Availability	Present or ready for immediate use
Fidelity	Good (i.e. correct) enough in respect of the application purpose
Integrity	Delivering the whole truth in respect of the source
Interactivity	Being responsive
Regulatory	Conformant with rules, the law or established usage

Example of urQoS specification

This example illustrates urQoS dimensions and values that are relevant for a surgeon who retrieves moving images from a medical imaging repository. The surgeon uses a Video-on-Demand (VoD) application to retrieve the images for one of the following purposes:

- to validate a diagnostic hypothesis;
- to have a closer look at the images to prepare a surgical treatment, for example in case the hypothesis has been confirmed;
- as reference material during surgery.

We may specify a template QoS category, i.e. a group of related urQoS dimensions, suitable for the surgeons' VoD application by the following QML contract type:

```
type Surgeon_VoD_QoScategorytype = contract
{//VoD appl. contains list, browse, and retrieve methods
  availability: set {"list","browse",
                "diagnostic phase","surgery phase"};
  integrity: increasing set {"lossy", //for listing
             "functional lossless",   //compression <25:1
             "perceptual lossless"};  //compression <12:1
             //increasing= higher values are better
  //remark: fidelity is modelled via integrity
  interactivity: increasing set{"low","normal","high"};
  regulatory: set {"not relevant","CR standard",
                   "Xray standard"};
};
```

The QML contract that specifies the urQoS dimensions and their possible values for a diagnostic hypothesis validation may then look as follows:

```
Diagnostic_validation_QoScategory =
                  Surgeon_VoD_QoScategorytype contract {
  availability == {"list","browse","diagnostic_phase"};
  integrity == {"lossy","functional lossless"};
  interactivity == {"low","normal"};
  regulatory == {"CR standard","Xray standard"};
};
```

For the preparation of a surgery, we may use a more stringent value for integrity, `"perceptual lossless"` for instance. Similarly, during the surgical treatment we may instantiate the VoD application with the more stringent value `"surgery phase"` for availability and the value `"high"` for interactivity.

3.3 QoS Classes and Characteristics

In the user-provider model of Fig. 5, the provider offers services as well as a set of associated QoS capabilities. We use QoS classes to facilitate these capability offers. Similar approaches can be found in ATM networks [Alles] and QoS-aware systems (e.g. [Lazar]).

A QoS class defines a set of QoS characteristics that the provider supports. In this paper, the definition of a QoS class also encompasses the (ranges of) values of its QoS characteristics.

Example:
The following QML contract defines the dimensions of a QoS class for an RSVP-based transport service:

```
type RSVPbased_QoSclasstype = contract {
      delay: decreasing numeric msec;
      rate:  increasing numeric Mb/s
};
```

An instance of the above QoS class type that supports maximum delay guarantees is:

```
RSVPguaranteed_QoSclass =
     RSVPbased_QoSclasstype contract {
     delay < 100;  //bounded delay in a guaranteed service
     rate < 0.064; //ISDN is e.g. a bottleneck link
};
```

The QoS characteristics of a QoS class are generally not completely independent of each other. A provider can furthermore define alternative sets of QoS characteristics and offer them to the users in the form of similar QoS classes. For example, a provider may offer a QoS class that uses the inverse of the delay QoS characteristic, i.e. the swiftness characteristic. A provider may also combine characteristics in a new characteristic (or perform other types of transformations). Instead of using image-height and image-width as characteristics, the provider may for example use image-size characteristic with permissible values like `"CIF"`, `"QCIF"`, etc.

3.4 QoS Interface Specification

In this section, we illustrate an approach to link the user requirements in respect of QoS to the provider-oriented QoS classes in accordance with the QoS model (Fig. 4).

QoS specification scenario at control-plane
In the example QoS class of Section 3.3, the transport provider guarantees that delays will not exceed 100 ms. This means that the provider has to be able to estimate the communication delay between the client and the VoD server before it offers the QoS class.

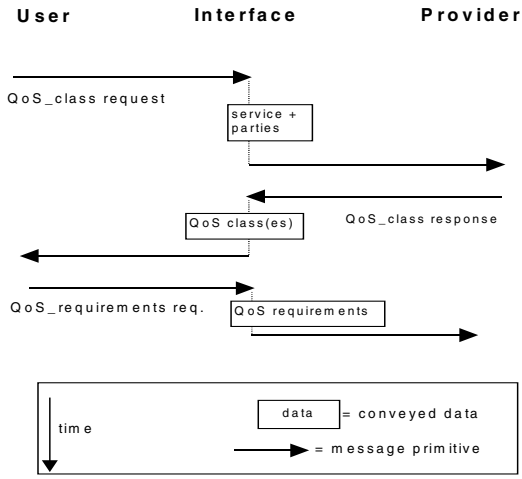

Fig. 5. QoS interface interaction scenario

The scenario of Fig. 5 illustrates how the user needs for QoS can be linked to a provider's QoS capabilities using the QoS model of Fig. 4. The user first queries the provider for the QoS classes that it supports. The user invokes a control-plane operational interface for this purpose. The provider then determines the set of most appropriate QoS classes for this user. What constitutes most appropriate may for instance depend on the delays between the participants of the communications session. The provider can for example consult the information base in the QoS

management plane to make an estimate of this delay. After having received the QoS classes, the user can select a suitable one and determine the values of the QoS characteristics that best match its requirements (i.e., urQoS). The user then responds with QoS requirements that represent the required qualifiers and values of the QoS characteristics (see also Fig. 5).

QoS requirement derivation

In the user-provider based QoS model, the user (typically in the form of application objects) must translate the urQoS dimensions and values to the midQoS characteristics.

An important factor in this translation is application domain knowledge. Of particular importance is the application domain information that has been elaborated and documented by communication experts. Examples are international standards like the analyzed CEN/TC251 and EWOS/EG MED medical reports, e.g. [Ewos]. Other ICT standards (e.g. CCIR.601) and results of ergonomic studies are also useful for QoS translations and mappings.

The urQoS dimension availability typically maps to the accessibility, reliability, swiftness and urgency midQoS characteristics. Integrity maps to accuracy, freshness, linkage unity and also swiftness. Table 2 illustrates the translation of availability values relevant in the medical VoD examples to accessibility. Table 3 illustrates the translation of integrity values relevant in the medical VoD examples to accuracy and linkage unity.

Table 2. Example availability urQoS to accessibility midQoS translation

AVAILABILITY	ACCESSIBILITY
"diagnostic phase"	"offline normal" or "offline high"
"surgery phase"	"online normal" or "online high"

Table 3. Example integrity urQoS translations

INTEGRITY	ACCURACY	LINKAGE UNITY
"functional lossless"	"CR degraded" or "Xray degarded"	"sync normal"
"perceptual lossless"	"CR" or "Xray"	"sync high"

4. QoS Characteristic Mapping

This section briefly describes the mapping of middleware QoS characteristics (e.g. accessibility, accuracy or swiftness) to the QoS characteristics of the underlying DRP. Fig. 6 shows the QoS mapping relations when we recursively apply the model of Fig. 4 to the interface between the middleware and the DRP.

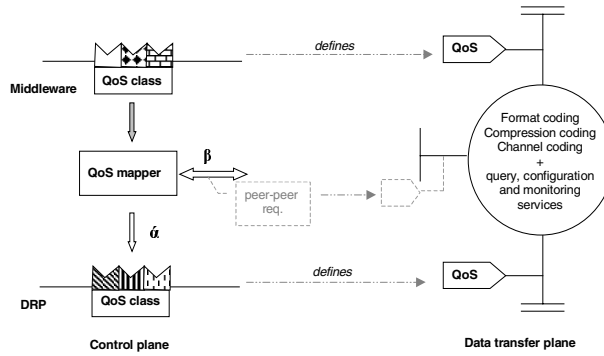

Fig. 6. Control-plane QoS characteristic mapping and its impact on the data transfer plane

The QoS mapper of Fig. 6 has to take the influence of the objects in the middleware's data transfer plane into account when it determines the QoS requirements for the DRP (ά). For example, if the accuracy midQoS characteristic value permits compression of video images, the video coding object in the data transfer plane typically introduces additional end-to-end delay.

A QoS mapping may also involve peer-to-peer negotiation (β). For instance, the two controller objects of Fig. 2 may need to agree on the encoding format and the encoding parameters to use. The DRP takes part in this negotiation in the role of the provider. This is because the DRP for instance has to provide a connection that has sufficient capacity to carry the output of the encoder. QoS characteristic mapping thus generally involves a *multiparty negotiation*.

In some cases, we can (also) use QoS mapping tables to convert the midQoS characteristics and associated QoS requirements to QoS characteristics that are supported by the DRP. For example, linkage unitity `"sync normal"` for pointing device-to-video synchronization may map to a maximum delay difference in the range of -580 to +820 ms at the DRP-level [Steinmetz].

5. Implementation

We are currently building a testbed to validate our framework. We have developed a video on demand demo based on the light profile of the CORBA Management and Control of Audio/Video Streams RFP [AVStreams]. The demo uses the Java Media Framework [Jmf] for streaming communications. The server and the client are interconnected by an RSVP-enabled router. The client allows the end-user to select a movie and conveys the user's selection to the server. The server responds by streaming an MPEG-1 video to the client using RTP/UDP.

Fig. 7 shows an overview of the CORBA A/V Streams RFP in terms of the objects that we use (see Section 2.1). The objects shown in gray together represent a stream. The str_ctrl object acts as a control point for application objects, which can be used to

request the establishment of a multimedia stream with a certain QoS. The str_ctrl object would therefore be present in a controller object of Fig. 3.

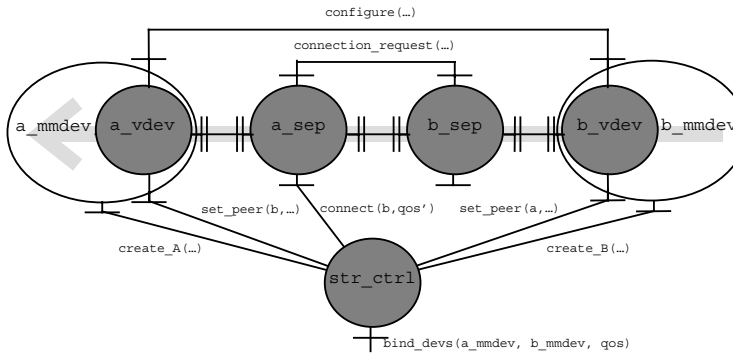

Fig. 7. Overview of CORBA A/V Streams objects.

The a_mmdev and b_mmdev objects encapsulate multimedia devices on the client and server machines of Fig. 3, respectively. They act as a factory for the *_vdev and *_sep objects. The a_vdev and b_vdev objects encapsulate the resources at the endpoint of the stream (e.g. a camera or a speaker) as well as the transport-independent media processing resources (e.g. an MPEG-1 encoder). They for instance engage in a peer-to-peer QoS negotiation process to determine the encoder and its settings to use. The *_mmdev and *_vdev objects would for this reason be divided over the application and middleware level objects of Fig. 3.

The a_sep and b_sep objects encapsulate the transport specific media processing resources such as an RTP packetizer and an RSVP entity. These objects would therefore be distributed over the middleware-level objects of Fig. 3. Observe that the streaming interface that interconnects a_sep and b_sep is the transport object of Fig. 3.

6. Conclusions

The architectural model for middleware support for QoS provisioning presented in this paper is a combination of an object model, a layered model, parts of the ISO QoS model and a model consisting of planes. The model is suitable for structuring the middleware internals for establishing media streams. A key feature of our model is the identification of QoS control objects (postioned in a control plane) that are responsible for establishing a (multi) media stream based on a QoS agreement.

Our approach is application oriented rather than system or network oriented. This is facilitated by the notion of a usage context and by the fact that the applications and our middleware-based system talk to each other in application oriented QoS terms.

We have validated parts of our architecture. To support this claim, we have shown that the CORBA A/V Streams specification fits into our architecture and we have

validated this through an initial implementation. This proves that the composition of our architecture in terms of layers, planes, objects and interfaces is a viable one.

We furthermore show that our notions of QoS specification can be expressed in QML and that the issue of QoS mapping can be realized in a heuristic manner by defining static mappings (e.g. by using mapping tables) between QoS specifications at the different layers.

Acknowledgements

This work was partially sponsored by the Telematica Instituut within the project AMIDST (http://amidst.ctit.utwente.nl). We would like to thank the members of the QoS workpackage for their contribution to this work. We particularly want to thank Dave de Vries for his input.

7. References

[Amidst] M. van Sinderen, "Amidst Project Description", http://amidst.ctit.utwente.nl

[Alles] A. Alles, "ATM Internetworking", Engineering InterOp, Las Vegas, 1995

[Aurre] C. Aurrecoechea, A.T. Campbell and L. Haw, "A Survey of QoS Architectures", Multimedia Systems Journal, Special Issue on QoS Architecture, May 1998;

[AVStreams] "Control & Management of Audio/Video Streams", document number: formal/98-06-05, June 1998

[Blair] G. Blair and J-B. Stefani, *"Open Distributed Processing and Multimedia",* Addison-Wesley, 1998;

[Cen] CEN/TC251 N98-034, "Quality of Service Requirements for Healthcare Information Interchange", draft CEN report, Feb. 1998;

[D3.1.2] A. van Halteren et al., "QoS Architecture", Amidst project deliverable D3.1.2,1999, http://amidst.ctit.utwente.nl/workpackages/wp3/index.html;

[Ewos] EWOS/ETG068, "Multimedia Medical Data Interchange", European Workshop for Open Systems (EWOS), Brussels, 1996, and earlier versions;

[Frølund] S. Frølund and J. Koistinen, "Quality-of-service Specification in Distributed Object Systems, Distributed Systems Engineering, 5, 1998, pp. 179 – 202;

[Gay] V. Gay, P. Leydekkers, R. Huis in't Veld, *Specification of Audio and Video interaction based on the reference model of ODP*, Computer Networks & ISDN systems, special issue on RM-ODP, January 1995.

[ISOQoS] ISO/IEC, "Information Technology – Quality of Service – Framework", ISO/IEC JTC1/SC21 N13236, Geneva, 1997;

[Jmf] Java Media Framework API Guide, Nov 1999, http://www.sun.com

[Lazar] A. Lazar, L. H. Ngoh and A. Sahai, "Multimedia Networking Abstractions with Quality of Service Guarantees", Proceedings of the SPIE Conference on Multimedia, Computing and Networking, San Jose, CA, Feb 1995

[Steinmetz] R. Steinmetz, "Human Perception of Media Synchronization", IBM European Networking Center, IBM – Technical report no 43 9310, 1993;

[Verma] D. Verma, *"Supporting Service Level Agreements on IP Network*s", Macmillan Technical Publishing, Indianapolis, 1999;