

# Specifying QoS for Multimedia Communications within Distributed Programming Environments

Daniel G. Waddington, Geoff Coulson and David Hutchison

Computing Department,  
Lancaster University,  
Lancaster LA1 4YR  
e-mail: [dan,geoff,dh]@comp.lancs.ac.uk

## Abstract

*Because of the increasing emphasis on distributed object programming for the provision of multimedia telecommunications services, it has become apparent that a unification between the distributed programming environment and techniques for QoS specification in multimedia communications must be made. Various QoS frameworks and QoS reference models have already been established and/or standardised. Each has approached the problem of QoS specification in its own way but it is possible to draw similarities across the board. This paper examines four different approaches to QoS specification and attempts to integrate these ideas into the distributed programming environment (DPE).*

## 1. Introduction

QoS can be defined as how “good” a communications service is between two or more points. QoS is hierarchical in that an end-to-end definition of QoS can be represented as a series of point-to-point QoS specifications. A QoS contract is interpreted as a two-way service contract whereby the QoS provider undertakes to support a given level of QoS if and only if the traffic generator undertakes to supply its data at the agreed QoS [Hutchison, 94]. This paper is concerned with the specification of QoS for multimedia communication at the application layer. We propose an approach to QoS specification derived from a study of several previous approaches and then discuss the application of this approach to multimedia telecommunication service provision through Distributed Programming Environments (DPEs). DPEs, which are layered on top of the operating system and communications system, provide an object based computational model for the benefit of programmers of distributed multimedia applications and facilitate flexible inter-object communications in a networked environment.

The rest of this paper is structured as follows. Section 2 introduces concepts relevant to distributed programming whilst section 3 defines various QoS concepts and definitions. Section 4 discusses some accepted QoS frameworks and section 5 outlines techniques of service provision through DPEs. Section 6. examines how the ideas of specifying QoS can be applied in DPEs and, in particular how these ideas could be used in the provision of multimedia network services. Finally section 7 draws our conclusions.

## 2. Distributed Programming Environments

The role of a distributed programming environment (DPE) is to provide the application programmer with a set of tools and abstractions to facilitate both location transparent and platform independent inter-object communications. DPEs communicate via message passing in the form of method requests/replies and within a distributed multimedia computing application objects also require the ability to exchange QoS dependent continuous data streams. Many DPEs only offer abstractions for remote method invocations [OMG, 91], however an increasing number of research bodies are currently looking at the provision of continuous media stream interactions [Coulson, 96]

[Halteren, 95]. A DPE offers its tools and services to the application programmer through both compile-time libraries and programming tools, together with run-time domain resident server processes.

## **2.1 Services of a DPE**

### **2.1.1 Simple Object Oriented Programming (OOP) model**

Distributed programming models are based usually on OOP techniques, as OOP provides a simply model for data and method encapsulation, together with clear units of functionality (i.e. the object and its interface) which are ideally suited to distribution.

### **2.1.2 Location transparency and object location services**

Within a DPE, a client application is able to 'transparently' make an invocation on any object method within its domain<sup>1</sup> as if it were in its own address space. In contrast to the standard programming paradigm, the programmers domain is extended out of the application address space into the network environment. To clarify the distinction between simple Remote Procedure Call (RPC) environments and DPEs, the latter offers additional services for remote object invocation, including object location and client authentication. Object location allows a client to 'enquire' on the availability of objects and their methods within a domain. Client authentication services provide mechanisms for ensuring security between end-users and service providers.

In addition to encapsulating the communications for inter-object method calls, bindings may also offer abstractions for continuous media communications between objects. Such services are more complex to abstract upon, since such interactions may require strict temporal dependencies and other QoS requirements.

### **2.1.3 Platform Independence**

DPEs also offer to the application programmer platform independence. This means that a programmer can make a remote method invocation, or process a media stream without concern to the remote platform. This is achieved through the use of a universally accepted language to define an objects interface signature (an interface is a window onto the methods or communication interactions offered by a particular object instance).

## **2.2 Common Object Request Broker Architecture (CORBA)**

CORBA is the distributed object model, produced as a result of research of a consortium of vendors called the Object Management Group (OMG). The model which is now in its second version (CORBA 2.0) is one of the accepted standard distributed computing models.

At the core of the CORBA architecture is the ORB itself. The ORB is the object interconnection bus, where clients are insulated from the mechanisms used to communicate with server objects. Within CORBA, the OMG have also specified how to define an interface between the component and the object bus, using the Interface Definition Language (IDL). Components specify in IDL the types of services that they provide, including methods they export, their parameters, attributes, error handlers, and inheritance relationships with other components. IDL becomes the contract that binds client to server components. It is the use of IDL that shields the client application programmer from the details of implementation and supporting platform of remote server objects.

Other than the provision of inter-object communications, the ORB offers various additional services. These service include object location, run-time interrogation of remote object interfaces, client authentication, object naming, event notification, concurrency control and more. Finally, the ORB is realised as a set of libraries and tools available to the application programmer.

---

<sup>1</sup> A domain may be physical or logical. A physical domain is comparable to physical communication paths between nodes, whilst a logical domain is analogous to the concept of a virtual network connection. Logical domains may even be a hierarchy of other domains.

### 3. QoS Concepts and Definitions

QoS specification is concerned with the required levels of quality interpretable in a particular layer of a system. In order to define how QoS is specified we need first to establish some basic terms which are introduced in the following sub-section.

#### 3.1 QoS Views

QoS is apparent at all layers in a communications architecture but is 'viewed' differently by each layer. A QoS view consists, in general, of notations for quality characterisation, service commitment and cost, together with a protocol for peer to peer QoS negotiation. Figure 1 illustrates the various QoS views of a typical system. The end user is concerned with perceptual QoS which essentially defines how "good" a service appears to the user, together with non-performance related QoS such as cost. User QoS is also the least *service-specific*<sup>2</sup> representation and is produced as a direct effect of application's performance and the supporting environment. There is a fine line between application QoS and user QoS, but it is probably best differentiated by the fact that application QoS is usually in terms of computational concepts where user QoS is not.

System QoS defines the QoS expected by the application of the underlying system. Incidentally, the system incorporates everything between the application and the hardware devices, including the DPE, the Operating System and any reusable system components<sup>3</sup>. System QoS is more service specific as its QoS characteristics (see next section) are directly relevant to the type of service.

Finally, at the lowest level, QoS is specified with respect to device capabilities, either peripheral or network. These QoS specifications, which we term 'device QoS' and 'physical network QoS', are the least service specific. Such specifications are usually very detailed and often concerned with the performance a particular device can offer. The term physical network QoS is simply the device QoS associated with networking, since from an engineering viewpoint there is no real difference between a peripheral device and a physical network device. Physical network device QoS can be characterised in terms of throughput, delay and error rate. However, system services over the physical network devices, such as those provided by the transport layer, are described with more complex system QoS characteristics.

Examples of network and device QoS include ATM cells per second of a virtual connection or video frames per second of a video capture device.

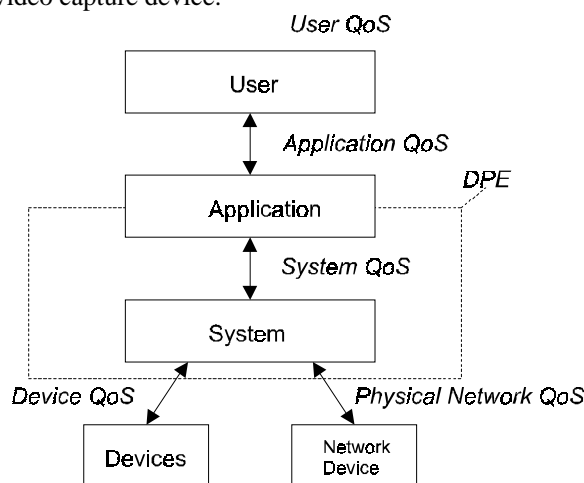


Figure 1 : QoS views

<sup>2</sup> Service specificity relates to how dependent upon the type of service the QoS specification is.

<sup>3</sup> System components include available resources such as codecs, filters and device abstractions.

### 3.2 QoS Characteristics

A QoS characteristic is a quantifiable aspect of QoS which is defined independently of the means by which it is represented or controlled [ISO, 95]. QoS characteristics are used to formulate a representation of the behaviour of an entity. Because of the diversity of multimedia services, there is no finite set of QoS characteristics. The idea of defining a set of ‘generic’ QoS characteristics is unsound and can only be realistically applied to particular devices. Because the concept of resource varies between layers, QoS characteristics are often layer dependent. The process of making hierarchical translations between characteristics of different layers is known as *QoS mapping*. Some QoS characteristics can also be derived, as opposed to mapped, from other characteristics. An example of such derivation is forming the variance of transit delay from transit delay, both are still defined as QoS characteristics. This reinforces the assumption that there is no single finite set of QoS characteristics suited to all requirements.

In addition to deriving characteristics, we can also ‘specialise’ characteristics. A specialisation may or must be applied in order to make a characteristic concrete and useable in practice [ISO, 95]. Several levels of specialisation can also exist, for example:

*time delay* → *transit delay* → *transit delay between Network Service Access Points*

### 3.3 QoS Categories

Logically related QoS characteristics can be grouped into categories corresponding to a classification of topics relating to QoS at its highest level. Table 1 presents an extensive selection of QoS categories which we have derived from various well defined QoS frameworks (in later sections we look at QoS categories and characteristics in more specific detail). The timeliness and volume categories are of principal concern to the provision of multimedia services, nevertheless other categories including accountability are still important.

QoS Categories	QoS Characteristics
Timeliness	Latency
	Delay jitter
	Recovery time
	Guarantee
	Synchronisation Intervals
	Availability
	Setup time
Volume	Throughput
	Peak Rate
Accuracy	Addressing accuracy (Cell Insertion Ratio)
	Error rate (Cell Loss Ratio, Bit Error Rate ...)
	Integrity
Robustness	Reliability (Mean Time Between Failures)
	Maintainability (Mean Time To Repair)
	Resilience
	Survivability
Accountability	Cost
	Auditability
Manageability	Monitorability
	Controllability
Privacy	Authenticatability
	Confidentiality
	Traffic Flow Security

**Table 1 : QoS Categories and Characteristics**

### 3.4 QoS Contracts and QoS Guarantees

A QoS contract is an agreed QoS specification between one or more entities. Contracts are of particular importance since a component 'C' can only be expected to 'provide' a defined level of QoS if and only if the requested 'required' QoS is also met by other components on which 'C' depends. If the bilaterally agreed QoS specification is not maintained, then the contract is broken. During the negotiation phase of service set up, various QoS contracts must be agreed between the consumer, retailer, network provider and service provider.

Another aspect concerning QoS contracting is the concept of *level of service* or *guarantee*. The guarantee defines to what extent a party will respect QoS parameters within a contract. Three classes of service guarantee are defined [ISO, 95]: *Compulsory*, *Statistical reliable (or Guaranteed service)* and *Best effort*. A guarantee level can be assigned to each parameter in a QoS contract or to the contract as a whole.

### 3.5 QoS Notation

QoS characteristics must be represented in a suitable notation to be of any use. The following describes some of the more common methods of QoS notation.

- *Single Static QoS Parameters* - representation of QoS characteristics through a single static value.

```
jitter = 1ms
throughput = 100kbps
peak duration = 10ms
```

- *Multiple QoS Parameters* - usually in the form of pairs. Multiple QoS parameters enable bounds upon QoS characteristics to be declared and more complex representations such as synchronisation intervals. For instance it may be a QoS requirement that a particular media stream be capable of supporting synchronisation at 10, 20 or 30 millisecond intervals.

```
delay = <10,100>          {min, max}
synchronisation intervals = <10,20,30>
```

- *QoS Structures* - more complex data structures including unions.

```
struct VideoQoS {
    union Throughput switch (Guarantee) {
        case Statistical: float Mean;
        case Deterministic: float Peak;
        case BestEffort:
            struct Interval {
                float min;
                float max;
            };
    };
    union Jitter switch (Guarantee) {
        case Statistical: float Mean;
        case Deterministic: float Peak;
    };
};
```

- *QoS Specification Languages* - such languages provide a more flexible specification mechanism. Examples include QL [Stefani, 93] and SCTL [Lakas, 96]. These notations express QoS as a set of temporal logic formulae that are linearly interpreted on timed state sequences. The formulae are built upon a set of atomic propositions using logical and temporal operators.

example of QL:

where 'e' is the event identifier and 'n' the occurrence of 'e';

delay\_QoS:  $T(e,n+1) - T(e,n) = 20 \text{ ms}$   
jitter\_QoS:  $10 < T(e,n) - T(e,n-1) < 100 \text{ ms}$

example of SCTL:

always *capture\_frame* with\_rate 15  
if x then *play\_frame* with\_rate 25 else *play\_frame* with\_rate 15

## 4. QoS Frameworks

In this section we present QoS frameworks that provide outlines for the concept of quality of service within a particular environment. For instance the ISO QoS framework, as described in section 3.1, is aimed at the provision of QoS in an ISO network environment. The ATM Forum's guidelines on aspects of QoS are aimed at QoS provision in an ATM networking environment and the Lancaster QoS-Architecture attempts to give guidance for aspects of QoS within an ATM and multimedia computing environment.

### 4.1 ISO QoS Framework

The basic ISO QoS framework is intended to assist those defining communications services and protocols, and those designing and specifying systems, by providing guidance on QoS applicable to systems, services and resources of various kinds. It describes how QoS can be characterised, how QoS requirements can be specified, and how QoS can be managed.

The key ISO QoS framework concepts include:

- *QoS requirements*, which are realised through QoS management and maintenance entities;
- *QoS characteristics*, as previously defined in section 3.2;
- *QoS categories*, which represent a policy governing a group of QoS requirements specific to a particular environment such as time-critical communications (see also section 3.3).

Table 2 shows the QoS categories and characteristics as defined by the framework. Each of these groups is considered to be relatively independent and it is intended that systems implement a specific set of characteristics, according to some QoS policy.

QoS Category	QoS Characteristics
Time-related	delay, validity, remaining lifetime
Coherence	temporal coherence, spatial consistency
Capacity-related	capacity, throughput, processing capacity, operation loading
Integrity-related	accuracy errors
Safety-related	safety
Cost-related	cost, user cost, data cost, resource cost, event cost
Security-related	protection, access control, data protection, confidentiality, authenticity
Reliability-related	availability, reliability, fault containment, fault tolerance, maintainability
Other	precedence

Table 2 : ISO QoS Framework Categories

Table 3 expands on the definitions for some of the ISO QoS framework defined characteristics, further details can be gained from [ISO, 95].

Temporal Coherence	Indicates whether an action has been performed on each value in a list within a given time window.
Spatial Consistency	Indicates whether the value of each variable in a list has been consumed in a given time window.
Capacity	The amount of service that can be provided in a specified period of time.
Processing Capacity	An amount of processing performed in a period of time
Operation Loading	Ratio of capacity used to capacity available
Accuracy	Accuracy is a QoS characteristic of concern to the user, for whom this characteristic refers to the integrity of the user information only. (The integrity of headers and similar protocol control information may be the subject of other characteristics)
Protection	The security afforded to a resource or to information
Access control	Protection against unauthorised access to a resource
Availability	The proportion of time that satisfactory service is available
Maintainability	Duration of any continuous period for which satisfactory, or tolerable, service is not available, related to some observation period (quantified as the probability Mean Time To Repair)
Fault Containment	Ability to operate in the presence of one or more errors/faults
Precedence	The relative importance of an object or the urgency assigned to an event

**Table 3 : ISO QoS Characteristics in detail**

In addition to QoS characteristics and categories, the framework outlines various aspects of QoS management. The framework is made up of two types of management entity that attempt to meet the QoS requirements by monitoring, maintaining and controlling end-to-end QoS [Campbell, 96]:

- i) *layer specific entities*: The task of the policy control function is to determine the policy which applies at a specific layer of the open system. The policy control function models any priority actions that must be performed to control the operation of the layer. The definition of a particular policy is layer-specific and therefore cannot be generalised. Policy may, however, include aspects of security, time-critical communications and resource control. The role of the QoS control function is to determine, select and configure the appropriate protocol entities to meet layer-specific goals.
- ii) *system wide entities*: The system management agent is used in conjunction with OSI system management protocols to enable system resources to be remotely managed. The local resource manager represents end-system control of resources. The system QoS control function combines two system-wide capabilities: to tune performance of protocol entities and to modify the capability of remote systems via OSI systems management. The OSI systems management interface is supported by the systems management manager which provides a standard interface to monitor, control and manage end-systems. The system policy control function interacts with each layer-specific policy control function to provide an overall selection of QoS functions and facilities.

The OSI QoS management framework also outlines general QoS mechanisms and their operational phases, the assignment of QoS function to entities and definitions for conformance, consistency and compliance.

## **4.2 ATM Forum UNI 3.1 Recommendations**

The ATM Forum's UNI 3.1 ATM bearer service QoS guidelines are based on the ITU-T I.350 recommendations [ITU, 93]. The aspects of QoS that are covered are restricted to the identification of

parameters that can be directly observed and measured at the point at which the service is accessed by the user. The forum assumes that QoS has a direct relationship to the Network Performance (NP) and that the principal difference is that QoS pertains to user oriented performance concerns of an end-to-end service, while NP is concerned with parameters that are of concern to network planning, provisioning and operations activities. To facilitate QoS provisioning in an ATM network, the forum defines ATM performance parameters and various QoS classes.

#### 4.2.1 ATM Performance Parameters

Table 4 summarises the set of ATM cell transfer performance parameters which correspond to the generic criteria of the assessment (shown in the right hand column) of the QoS.

Cell Error Ratio	$\frac{\text{Errored Cells}}{\text{Successfully Transferred Cells} + \text{Errored Cells}}$	Accuracy
Severely-Errored Cell Block Ratio	$\frac{\text{Severely Errored Cell Blocks}}{\text{Total Transmitted Cell Blocks}}$	Accuracy
Cell Loss Ratio	$\frac{\text{Lost Cells}}{\text{Total Transmitted Cells}}$	Dependability
Cell Misinsertion Ratio	$\frac{\text{Misinserted Cells}}{\text{Time Interval}}$	Accuracy
Cell Transfer Delay	Delay in transfer of an ATM cell between nodes.	Speed
Mean Cell Transfer Delay	Mean CTD is defined as the arithmetic average of a specified number of cell transfer delays for one or more connections	Speed
Cell Delay Variation	Describes 1-point Cell Delay Variation (1-point CDV) and 2-point Cell Delay Variation (2-point CDV). 1-point variation is measured with reference to a single measurement point, where as 2-point variation is measured from the output of a connection portion with reference to the pattern of the corresponding events observed at the input to the connection portion.	Speed

**Table 4 : I.350 ATM Performance Parameters**

#### 4.2.2 QoS Classes

UNI 3.1 also establishes the concept of QoS classes. A QoS class can have a specified performance parameter (Specified QoS) or no specified performance parameters (Unspecified QoS). QoS classes are inherently associated with a connection. The network itself may support several specified QoS classes and at most one unspecified QoS class. The performance provided by the network should meet (or exceed) performance parameter objectives of the QoS class requested by the ATM end-point. Both ATM VPCs and VCCs should indicate the requested QoS by a particular class specification.

##### **Specified QoS Parameters**

A Specified QoS class provides a quality of service to an ATM virtual connection (VCC or VPC) in terms of a subset of the ATM performance parameters previously described. For each Specified QoS class, there is one specified objective value for each performance parameter. The following list shows

those QoS classes which have been defined in UNI3.1 (the italicised print represents the Service Class).

Specified QoS Class 1: *Circuit Emulation, Constant Bit Rate Video*

- should yield performance comparable to current digital private line performance.

Specified QoS Class 2: *Variable bit Rate Audio and Video*

- is intended for packetized video and audio in teleconferencing and multi-media applications.

Specified QoS Class 3: *Connection-Oriented Data Transfer*

- is intended for interoperation of connection oriented protocols, such as Frame Relay.

Specified QoS Class 4: *Connectionless Data Transfer*

- is intended for interoperation of connection-less protocols, such as IP, or SMDS.

### Unspecified QoS Class

No objective is specified for the performance parameters. However, the network provider may determine a set of internal objectives for the performance parameters. In fact, these internal performance parameter objectives need not be constant during the duration of the call.

### 4.3 Lancaster's QoS-A

Lancaster's QoS architecture [Campbell, 94] offers a framework to specify and implement the required performance properties of multimedia applications over high-performance ATM based networks. This architecture does take into account the higher communication layers and incorporates notions of media flows, service contract and flow management. It also acknowledges the different representations of QoS between layers, including the representation of QoS within a distributed platform. Table 5 represents the QoS characteristics relevant to the QoS-A Multimedia Enhanced Transport Service [Campbell, 94], which has a strong emphasis on full end-to-end guarantees.

QoS Category	QoS Characteristics
Flow specification	media type, frame size, frame rate, burst, peak rate, delay, loss, interval, jitter
Adaptation	loss adaptation, jitter adaptation, throughput adaptation, delay adaptation, disconnect
Maintenance	maintenance
Connection-related	service, start time, end time
Cost	cost

**Table 5 : QoS-A QoS Categories**

Media Type	Common flows for video, voice and data.
Frame Rate/Frame Size	Average throughput requirement.
Peak Rate	Maximal throughput.
Burst	Size of potential bursts of traffic.
Delay	End-to-end delay.
Maintenance	Options for maintenance are monitor, maintain and no_maintain. Monitor instructs the QoS-A to periodically deliver measured performance assessments relating to the specified flow. Maintain, on the other hand, attempts to transparently exert fine grained corrective action to maintain QoS levels according to the service contract, but does not deliver any assessments.
Service	Support for fast, negotiated or forward service types.

	Negotiated service supports full end-to-end protocol negotiation. Fast connect service reservation and data transfer phases coincide. Forward reservation allows network and end-system resources to be booked ahead of time.
Start time/End time	Service window bounds.

**Table 6 : QoS-A QoS Characteristics in detail**

The precise interpretation of the QoS characteristics as defined in Table 5, is determined by the commitment specification. The QoS-A `commitment_t` parameter is the same as the service guarantee as defined in Section 2.4.

Although the QoS-A architecture only specifies a small set of QoS characteristics relevant to media transport the model does realise the necessity for the management of QoS at all levels and the mapping of QoS between the layers in the architecture.

#### 4.4 TINA-C Recommendations

The Telecommunications Information Network Architecture (TINA) Consortium is an international collaboration aiming at defining and validating an open architecture for telecommunications systems for the broadband, multimedia, and information era [TINA, 94]. The intention of the consortium is to make use of recent advances in distributed computing, particularly the advent of the CORBA distributed object model, for the provision of telecommunication services in an open architecture. TINA-C have identified the requirements for QoS specification at both the network and service level. We now briefly discuss their approach to specifying QoS at the service level, since this has a direct relevance to QoS specification in a DPE.

TINA acknowledges the distinction between QoS associated with a contract between objects and their environment (object service attributes) and streams and their environment (stream service attributes). This is further divided into *functional* and *non-functional* QoS requirements; functional requirements which are relevant to the operation/stream signatures and non-functional requirements provided by the DPE, e.g. availability, security. Tables 7 and 8 show some of the TINA object and stream service QoS attributes.

QoS Category	QoS Characteristics
Availability	Peak MTBF, Mean MTBF
Performance	Peak Response Time, Mean Response Time
Others	Security, Consistency, Reliability

**Table 7 : TINA Object Service Attributes**

QoS Category	QoS Characteristics
Throughput	Peak Throughput, Mean Throughput, Bound Throughput
Jitter	Peak Jitter, Mean Jitter
Delay	Peak and Mean Delay
Error Rate	Peak and Mean Error Rates

**Table 8 : TINA Stream Service Attributes**

Because the CORBA object model is essentially limited to simple request/reply method interactions the TINA Consortium extended the model to support multiple interfaces on an object and stream interfaces on an object. Their interface definition language, known as ODL-95, supports these requirements and the necessary placeholders for the specification of QoS. Section 4.2 briefly introduces the use of ODL-95 and further information can be gained from [OMG, 95].

## 5. Distributed Multimedia Models

This section is intended to outline the general architecture of distributed multimedia platforms and their requirements for QoS support in the provision of broadband networking services. Distributed multimedia applications are built up through the logical “binding” of multimedia service components [ODP, 92]. This binding is usually controlled centrally through a composite *binding object*. The binding object is the client’s point of contact for the establishment and control of a multimedia service, see figure 2.

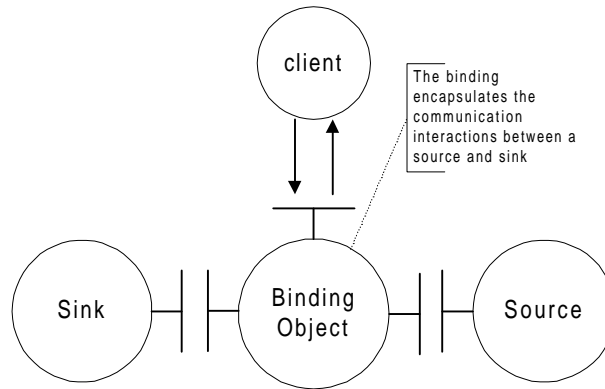


Figure 2 : Binding model

During the service establishment phase, which involves the selection and local binding of various service components, the binding object is responsible for type checking the compatibility of the to-be-bound stream interfaces. This type checking includes the verification of required and provided QoS through the examination of the interface’s associated QoS specification.

### 5.1 Operational Interfaces

Operational interfaces comprise a set of object method signatures (a method is defined as a unit of functionality offered by an object). Methods are invoked in a request/reply interaction style and invocations made upon them in a distributed programming environment are both location and implementation transparent<sup>4</sup>. Within a distributed multimedia system operational interfaces are primarily used for control interfaces on system and service objects.

Below is an example standard CORBA operational interface description. The standard CORBA model does not support QoS specification on object methods.

```
interface VideoControl {
    void getBrightness(in short level);
    short setBrightness( void );
};
```

### 5.2 Event Interfaces

To specify the continuous flow of media in multimedia applications, it is necessary to create an abstraction which captures the concept of information flowing over time [Blair, 95]. We could model such continuous interaction as an asynchronous method calls on operational interfaces, however with this approach there is no concept of a connection governed by overall QoS, as each invocation is a separate isolated event. Therefore to facilitate this requirement we add the concept of *event interfaces*. Event interfaces provide a signature describing unidirectional points of interaction, *events*. An event is described by an identifier, directionality and associated data. Events are also typed.

<sup>4</sup> Different object models, such as CORBA, Distributed COM and Distributed SOM each have their own method of achieving this transparency goal.

Event interfaces specify component level interactions, these interactions are defined as device QoS and physical network QoS. An event interface as described in Lancaster's extended CORBA computational model [Coulson, 96] describes purely the interaction points without any reference to values of required QoS. The inheritance of QoS\_events provides the signature for standard events used for QoS management. These standard events provide mechanisms for feedback from QoS monitoring and notification of QoS degradation or failure.

```
events VideoDecompressor : QoS_events {
    in FrameCompressed(INDEO_T frame);
    out FrameDecompressed(RAW_T frame);
    out DecompressionFailed(MESSAGE_T reason, TIME_T time);
};
```

Below is described an event interface (known in TINA terminology as a stream interface) using the TINA Object Definition Language [Kitson, 95]. The QoS structures are simple and static.

```
interface VideoCompressor {
    behaviour "This is a Indeo Video Compressor";

    sink    VideoFlowType  display    with Video_QoS requiredQoS;
    sink    AudioFlowType  speaker    with Audio_QoS requiredQoS;
    source  VideoFlowType  camera     with Video_QoS offeredQoS;
    source  AudioFlowType  microphone with Audio_QoS offeredQoS;
};
```

The most prominent differences between the above described approaches to event interface specification is that the TINA-ODL description uses the concept of flow type to describe the associated flow of data. Lancaster Event Definition Language (EDL) allows multiple data types to be associated with a particular event. This approach gives a wider flexibility in the specification of continuous data interactions.

Event interfaces are "locally"<sup>5</sup> bound through the sharing of resources. This binding provides the communication path from source to sink and establishes an overall quality of service associated with the connection. Before the event interfaces are bound, they are typed checked to ensure that each interface provides the QoS as required by the adjacent interface. Such type checking is an important part of QoS management, however the issue will not be discussed further in this paper.

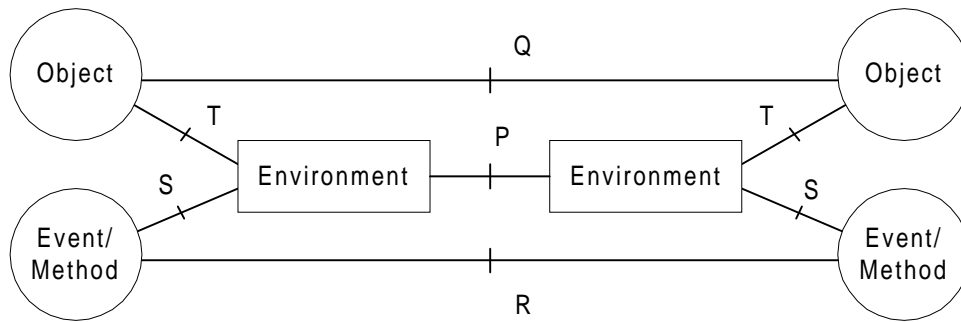
## 6. QoS Specification on Interactions within a DPE

In this section we extend the ideas of QoS specification described in section 4 and propose an approach to the application of QoS within a DPE. We define the concept of a QoS interaction point, which is any interaction between entities which may require the specification of QoS constraints. Each of the frameworks and recommendations as previously described, are concerned with specifying QoS at one or more of these interaction points. Both the ISO QoS framework and the ATM Forum's recommendations emphasise QoS parameters which are relevant to interactions on the underlying network, whilst in addition to this, the Lancaster QoS-Architecture extends these interactions into a multimedia computing environment. The TINA-C recommendations, described in section 4.4, differentiate between the QoS interactions of an object and its environment, and also between different object methods.

We extend the idea of interaction from the TINA-C recommendations and define the complete set of QoS interactions apparent within a DPE. Figure 3 shows this set, which includes all mathematically possible interactions except for the interactions between an object and a method of event, as we do not consider this to be relevant. Each of the points is arbitrarily assigned a letter so that they can be later referenced.

---

<sup>5</sup> Local binding means that there is no stub or skeleton proxy code involvement. The stub provides client side location transparency and the skeleton provides server side location transparency support.

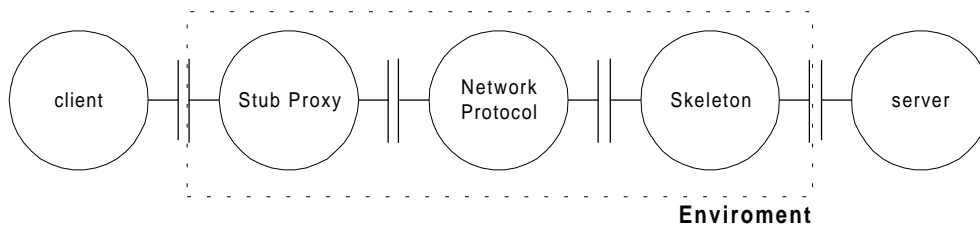


**Figure 3 : QoS Interaction Reference Points**

Each reference point is considered to potentially require QoS specification. Creating a multimedia service in a DPE involves establishing and negotiating QoS contracts at each of these points. The rest of this section details further our defined QoS interaction points and proposes techniques for their specification.

### 6.1 Reference Point 'P' : *Environmental QoS*

The primary purpose of a DPE is the provision of transparent communication between remote objects. The DPE also provides other services such as object location and user authentication. Requirements of QoS relating to a DPE, concern the quality of these services it provides, that is the services as supplied by the underlying middleware and in particular remote procedure calling services of the client stubs, skeleton proxies and transport protocols. QoS concerned with these DPE services is termed environmental QoS, and can be mapped onto the QoS requirements of various system components. For example, if a method requests a particular request/reply delay time, this may represent various QoS constraints on methods within a transport object. Figure 4 illustrates this further.

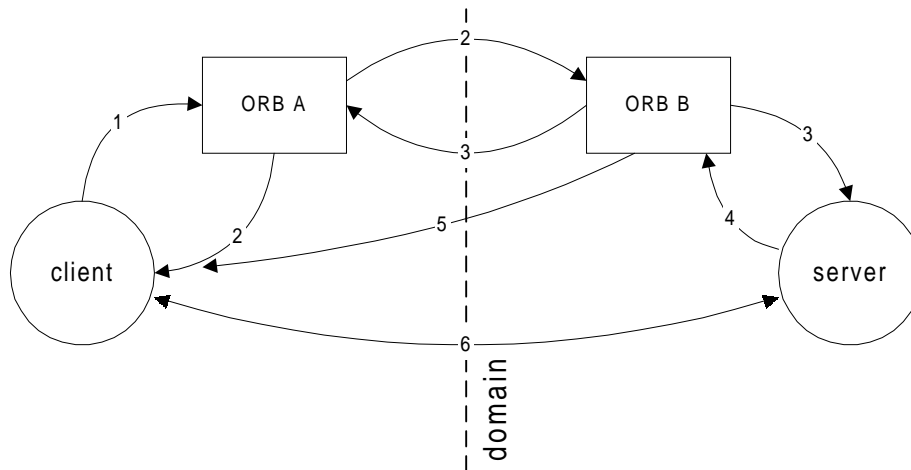


**Figure 4 : DPE Objects**

Interactions between distributed objects in different domains requires the negotiation of QoS contracts between environments. An example of this is the interaction between ORBs in a CORBA distributed programming environment, where by QoS negotiation of ORB services, such as authentication, delivery guarantee and message security must be made. This allows a client object residing in one domain to be assured the quality of the ORB services in another domain.

QoS characteristics applicable to the environment include, cost, message security, invocation timeliness, invocation accuracy, resilience (the ability to recover from errors), data integrity, authenticability (the ability to supported client authentication), reliability and monitorability (the ability to support QoS feedback from environment).

The service characteristics can be defined through static data structures, similar to those as defined by TINA-C. Actual mechanisms for the exchange of environmental QoS is really platform implementation dependent.



**Figure 5 : CORBA Location Interactions**

Figure 6 shows the series of messages passed between objects and ORBs within a CORBA environment. Message interactions 2 and 3 are suitable placeholders for the exchange of environmental QoS requirements. One possible approach to facilitating QoS requirements exchange between CORBA environments would be through the extension of the *LocateRequest* and *LocateReply* messages [OMG, 95] as follows;

```

module GIOP {
    struct environment_QoS {
        short    mean_request_time;
        short    mean_reply_time;
        boolean  authentication;
        short    invocation_accuracy;
        short    security_level;
    };

    enum LocateStatusType {
        UNKNOWN_OBJECT,
        OBJECT_HERE_QOS_SUCCESS,      /* indicating QoS agreeable
        OBJECT_HERE_QOS_FAILED,      /* QoS not-agreeable
        OBJECT_FORWARD
    };

    struct LocateRequestHeader {
        unsigned long    request_id;
        sequence <octet>    object_key;
        environment_QoS    requiredQoS;
    };

    struct LocateReplyHeader {
        unsigned long    request_id;
        LocateStatusType    locate_status;
        environment_QoS    requiredQoS;
    };
};

```

## 6.2 Reference Point 'Q' : Interface-interface QoS

Interface interactions are concerned with the specification of QoS as provided by the whole set of methods or events. Such specification is important when establishing bindings between objects, where each party must ensure that the other to-be-bound object can maintain certain object QoS

contracts. QoS information about an object must be obtainable from both an object interface and an instance of the object since a particular instance may have certain run-time resource limitations imposed on it. The exchange of run-time QoS specification can be done through methods on a object. Specification on an object interface must is done through the interface definition language itself or some other association.

```

struct camera_QoS {
    boolean    monitorability;           /* support for monitoring */
    short      cost;                     /* cost of use */
};

interface Camera : QoS_methods with camera_QoS    {
    void      start_stream();             /* camera control methods */
    void      stop_stream();
};

interface QoS_methods {                  /* methods to determine QoS
    short     get_remaining_lifetime();    requirements */
    short     get_current_loading();      /* ascertain load */
    float     get_error_ratio();
};

```

The methods provided through the QoS\_methods interface are used to determine the state of QoS characteristics for a particular object instance during run-time. Such methods may be used by a binding object to determine the suitability of an object before binding to.

### 6.3 Reference Point 'R' : *Method-Method / Event-Event QoS*

Reference point 'R' defines QoS requirements between methods or events. A particular event may require a specific QoS from another method in order that it maintain its own QoS contract. An example of this is a multimedia component requiring timing events from a system timer. Operational interface methods may also make demands on other methods.

```

struct method_id {
    char * interface;
    char * method;
};

struct QoS_control {
    method_id method;
    float     availability;
};

interface Compressor {
    void Stop() with QoS_control;
};

interface Control {
    void PostQuitSignal();
};

```

The method Compressor.Stop can only successfully offer its QoS contract provided that the method Control.PostQuitSignal maintains a certain availability.

```

events Compressor {
    in FrameReady(RAW_T frame);
    out FrameCompressed(INDEO_T frame);
    out CompressionFailed(MESSAGE_T reason, TIME_T time);
};

```

```

events Camera {
    out FrameCaptured(RAW_T frame);
    out CaptureFailed(MESSAGE_T reason, TIME_T time);
    in SignalCapture(TIME_T time);
};

events Timer {
    out SignalEvent(TIME_T time);
};

```

Event QoS Definitions:

```

/* signal every 10 ms */
T(Timer.SignalEvent,n+1) - T(Timer.SignalEvent,n) = 10ms

/* less than 1ms delay between capture and signal) */
T(Camera.FrameCaptured,n) - T(Camera.SignalCapture) < 1ms

/* less than 5ms delay between signal and compression */
T(Compressor.FrameCompressed) - T(Timer.SignalEvent,n) < 5ms

/* mean time between failures */
average( T(CompressionFailed,n+1) - T(CompressionFailed,n) ) > 2000

/* frame dropping ratio */
(  $\sum$ CompressionFailed /  $\sum$ FrameCompressed ) < 0.001

```

The TORBoyau [Dang, 95] distributed multimedia platform, derived from CORBA, also supports the specification of event interfaces in the form of *signal* interactions. A QoS clause is optionally associated with each signal as shown in the following example.

```

interface<stream> Telephone {
    flowIn mike(audio);
    requires QoS(
        "audio_encoding=LINEAR",
        "audio_frequency=22050",
        "audio_precision=16"
        "audio_channels=2");

    flowOut speaker(audio);
    providesQoS(
        "audio_encoding=ULAW",
        "audio_frequency=8000",
        "audio_precision=8",
        "audio_channels=2");
};

```

The TORBoyau system's extended IDL is also capable of supporting temporal logic by replacing the QoS parameters with language clauses.

## 6.4 Reference Point 'S' : *Method-Environment/Event-Environment*

Reference point S defines the interaction between the methods/events with the distributed programming environment. Requirements, as described in section 5.1, include RPC security, request and reply delay, etc.

### 6.4.1 Method-Environment QoS

The application programmer may want to associate a particular QoS specification on a single method. An example of such requirement is a method which needs individual costing requirements and whose

request/reply time must also be bounded. Performance characteristics which are independent of subsequent calls, such as jitter and synchronisation, have no relevance to one-off operational method calls. Method signatures can attach a QoS definition structure with a suitable place-holder in the definition language. One possible technique is appending the 'with' clause at the end of the method signature. Below illustrates this approach.

```
interface video_control {
    short get_frame_rate( void ) with monitor_QoS;
    void stop_stream(in short level) with shutdown_QoS;
};

struct T_monitor_QoS {
    TIME_T reply_delay;
};

struct T_shutdown_QoS {
    TIME_T request_delay;
    TIME_T reply_delay;
    COST_T cost;
};
```

Another possible technique, suited to a CORBA environment, is the application of contexts. A context is itself a pseudo-object<sup>6</sup> which represent a set of name, value pairs and simply operations to add, remove and interrogate this information. Contexts are associated with methods in the interface description and provide an ideal mechanism for the exchange of QoS information.

```
interface video_control {
    short get_frame_rate( void )
        context("reply");
    void stop_stream(in short level)
        context("request", "reply", "cost");
};
```

#### 6.4.2 Event-Environment Interface QoS

In the same way a method can expect a certain QoS from its environment, an event may also make certain QoS requirements (since the only difference between methods and events is their style of interaction). QoS characteristics such as jitter become more important since events usually occur recurrently.

```
struct fail_QoS {
    float    max_request_time;
};

events Camera {
    out FrameCaptured(RAW_T frame);
    out CaptureFailed(MESSAGE_T reason, TIME_T time) with fail_QoS;
    in SignalCapture(TIME_T time);
};
```

### 6.5 Reference Point 'T' : *Interface-Environment QoS*

Finally point T defines interactions between interfaces and their environment. This is similar to point S QoS except that it is concerned with QoS requirements which are specific to the complete set of methods or events, rather than one in particular.

---

<sup>6</sup> A pseudo object is a component which appears to over methods but in fact is not an object and does not have reference identifiers etc.

```

struct camera_QoS {
    boolean    authentication;    /* support for authentication */
    short      security;          /* level of RPC security */
};

interface Camera with camera_QoS {
    void       start_stream();    /* camera control methods */
    void       stop_stream();
};

```

## 6.6 QoS Specification on Bindings

The term 'binding' refers to the end-to-end connection between one or more service components. It is the binding that provides an agreed end-to-end QoS for the application. Binding QoS is different from the ODP defined 'QoS provided by the binding' which relates purely to the service offered by the transport system and excludes the QoS required and provided by the periphery components. QoS of a binding is achieved through the configuration and negotiation of QoS at the interaction points as previously described.

Binding QoS is specified during a `_bind` call. Binding objects, which are a result of a call to a factory object, are service specific and provide a pre-defined service type, e.g. MPEG-1 audio/video stream, CD-audio and Indeo Video. The required QoS for the binding is specific to the type of service and can be requested through the bind call. An example of QoS specification on a binding using a CORBA context object is given below.

```

CORBA::Context QoS_ctx = CORBA::Context::Context();

QoS_ctx -> set_one_value("Quality Level",100);
QoS_ctx -> set_one_value("Format",ID_IV32);

binding bo = bind_factory.create("Indeo_Video_Stream");
binding ctrl indeo_control = bo.bind("cam1:camSrv",
                                     "decin1:decSrv",
                                     "decout1:decSrv",
                                     QoS_ctx,&env);

```

Current CORBA implementations of the `bind` operation do not support QoS specification hooks (mechanisms for associating QoS parameters to a binding), however this is viewed as a simple extension to the environment.

Through the realisation of these defined QoS interactions into a DPE implementation, we create a set of tools and abstractions ideally suited to the requirements for the provision of multimedia services. The application and DPE become QoS 'aware' which is essential to the provision of QoS constrained telecommunication services.

## 7. Conclusion

We have discussed the relevant issues concerning QoS specification for multimedia telecommunications services by initially identifying points of interaction which are considered QoS oriented and then applying characteristics from various QoS frameworks to specify QoS at these points. Defining how QoS should actually be specified is a difficult problem to solve since the approach taken depends upon the nature of the service, and therefore it is suggested that only guidelines for QoS specifications within a DPE can be made, as the advent of new services will affect the requirements for specification.

Currently we are using the ideas presented in this paper in the development of a distributed programming environment supporting QoS management. The prototype system will provide the application programmer with the tools necessary to build continuous media stream bindings from a set of user requirements together with the system components for dynamic QoS management. These

elements will contribute to a “QoS Manager”, which is being developed in a project funded by the BT Labs under their University Research Initiative.

## ACKNOWLEDGEMENTS

We would like to acknowledge the kind support of BT Labs in funding this research under the Management of Multi-service Networks project within BT’s University Research Initiative (BT-URI). We would also specifically like to acknowledge the collaboration of our colleagues at Imperial College who are also funded under this BT-URI project.

## References

[Blair, 95] G.Blair and G.Coulson, “Supporting the Real-time Requirements of Continuous Media in Open Distributed Processing”, *Computer Networks and ISDN Systems* 27 p1231-1246, 1995.

[Bochmann, 96] G.Bochmann and A.Hafid, “Some Principles for Quality of Service Management”, *Universite de Montreal*, May 1996.

[Campbell, 94] A.Campbell, G.Coulson and D. Hutchison, “A Quality of Service Architecture ”, *ACM Computer Communications Review*, Volume 24, Number 2, pp6-27, 1994.

[Coulson, 95] G.Coulson, G.S.Blair, “Architectural Principles and Techniques for Distributed Multimedia Application Support in Operating Systems”, *ACM Operating Systems Review*, Vol 29, No 4, pp17-24, October 1995.

[Coulson, 96] G.Coulson and D.G.Waddington, “A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks”, *Lancaster University MPG*, June 1996 (to be published in Springer LNCS series).

[Dang, 95] F. Dang Tran, V. Perebaskine, "TORoyau: Architecture et Implementation", *Note Technique NT/PAA/TSA/TLR/4587*, CNET, Centre Paris A, 38-40 rue du General Leclerc, Issy-les-Moulineaux, France, December 1995.

[Fedaoui, 95] L.Fedaoui, A.Seneviratne, E.Horlait, “Implementation of End-to-End Quality of Service Management Scheme”, *Universite Pierre et Marie Currie, QoS Workshop*, Paris,1995.

[Halteren, 95] A.Halteren, P.Leydekkers, H.Korte, “Specification and Realisation of Stream Interfaces for the TINA-DPE”, *Proceedings of TINA’95 Conference*, Melbourne Australia, p299-312, February 1995.

[Hensall,88] J.Hensall, S.Shaw, “OSI Explained: End-to-end Computer Communication Standards”, ISBN 07458-0253-2 *Ellis Horwood*, New York, 1988.

[Hutchison, 94] D.Hutchison, G.Coulson, A.Campbell and G.Blair, “Quality of Service Management in Distributed Systems”, in *Distributed Systems Management* (M.Sloman, ed.), Chapter 11, *Addison-Wesley*, 1994.

[ISO, 95] ISO/IEC, “QoS - Basic Framework - CD Text”, *Joint ISO/IEC&ITU-T Interim Meeting*, Toronto, January 1995.

[ITU, 93] ITU-T Recommendation I.350, "General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDN", *COM XVIII-R 114E*, 1994.

- [**Kawalek, 95**] J.Kawalek, "A User Perspective for QoS Management", 3<sup>rd</sup> International Conference on Intelligence in Broadband Service and Networks (IS&N 95), September 1995.
- [**Kerherve, 96**] B.Kerherve, A.Pons, G.Bochmann, A.Hafid, "Metadata Modelling for Quality of Service Management in Distributed Multimedia Systems", IEEE Multimedia 1996.
- [**Kitson, 95**] B.Kitson, P.Leydekkers, N.Mercouroff, F.Ruano, "TINA Object Definition Language (TINA-ODL) Manual, TR\_NM.002\_1.3.95, TINA Consortium, 1995.
- [**Lakas, 96**] A.Lakas, G.Blair, A.Chetwynd, "A Formal Approach to the Design of QoS Parameters in Multimedia Systems", Lancaster University (awaiting publishing), 1996.
- [**Leydekkers, 95**] P.Leydekkers, V.Gay, L.Franken, "A Computational and Engineering View on Open Distributed Real-time Multimedia Exchange", Proceedings of NOSSDAV'95 (Springer-Verlag), Boston, USA, April 1995.
- [**MMCF, 95**] Multimedia Communications Form, Inc. "Multimedia Communications Quality of Service", Final MMCF Document , MCFI, 1995.
- [**Nahrstedt, 95**] K.Nahrstedt, A.Hossain, S. Kang, "Probe-based Algorithm for QoS Specification and Adaptation", University of Illinois, QoS Workshop, Paris, 1995.
- [**ODP, 92**] ISO/IEC JTC1/SC21 Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3:Prescriptive Model, ANSI, 1430 Broadway, New York, NY 10018, USA, 24<sup>th</sup> June 1992.
- [**OMG, 91**] "The Common Object Request Broker : Architecture and Specification", Object Management Group, OMG Document Number 91.12.1, <http://www.omg.org>, December 1995.
- [**OMG, 95**] CORBA 2.0 "Interoperability - Universal Networked Objects", Object Management Group, OMG TC 95-3-xx, 1995.
- [**Sluman, 92**] C.Sluman, "Interoperability Is Not Enough", Communications Week International, Page 20, April 1992.
- [**Stefani, 93**] J.B.Stefani, "Computational Aspects of QoS in an Object Based Distributed Architecture", 3<sup>rd</sup> International Workshop on Responsive Computer Systems, Lincoln, NH, USA, September 1993.
- [**TINA, 94**] TINA-C, "Overall Concepts and Principles of TINA", TB\_MDC.018\_1.0\_94, February 1995.
- [**Vogel, 95**] A.Vogel, B.Kerherve and G.Bochmann, "Distributed Multimedia and QoS: A Survey", IEEE Journal of Multimedia, Volume 2, Number 2, 1995.