A Software Framework for Application Level QoS Management

Varuni Witana
CRC for Distributed Systems Technology, University of Technology, Sydney
Michael Fry
School of Computing Sciences, University of Technology, Sydney
Mark Antoniades
CRC for Distributed Systems Technology, University of Technology, Sydney

Abstract

We present a software framework that facilitates the development of adaptive applications. This framework allows the installation of application dependent policies which govern the adaptive behaviour of the application. These policies take into account user preferences and resource availability. The framework provides a generic interface to a variety of re-usable resource classes. We describe an application built using this framework.

Keywords: QoS, QoS, QoS Framework, Adaptive applications, QoS Mapping/Translation, Resource management

1 Introduction

The computer networks of the future are likely to remain heterogeneous. With the growing popularity of hand held devices, networked computers will range from high powered workstations to relatively resource poor mobile devices. Changes in network technology have also led to an increase of heterogeneity. In recent years we have seen the concept of "differentiated services" with the deployment of ATM networks with support for multiple classes, as well as initiatives such as intserv [13] and diffserv [12] from IETF. In addition there are various wireless network technologies now being deployed each having different characteristics, services and costs. Wireless networks are characterised by variable QoS as the user moves around. In order to support distributed multimedia applications it is necessary for these networks to provide some form of QoS support. However the nature of this support may vary. Not all networks would provide guaranteed class of service. In fact the popularity of the Multicast Backbone (MBONE) [14] has proved what is possible even with the current best effort networks. Thus the user is faced with a choice of multiple networks some of them supporting multiple classes of services.

QoS aware applications have been developed which are capable of adapting to their environment. Multimedia applications widely used in the Internet today, such as *vic* [16], *vat* [6], *rat* [18] all have elements of adaptivity. In most cases this adaptivity involves scaling their operation in order to vary the amount of resources consumed The range of acceptable adaptation is application specific and user dependent. Video in a tele-medicine application has tight constraints due to the need for observing fine detail, on the other hand video conferencing has looser constraints. The latter application could adapt its operation within a fairly wide range of spatial and temporal scales and still be of acceptable quality to the user. Adaptive applications could also adapt their operation to deal with other variations in QoS characteristics of networks. As an example *vat* dynamically adapts to the delays introduced in the network by varying its play-out buffer length. As well as possessing adaptive mechanisms the application needs to know when to trigger these mechanisms. Some applications monitor their environment in order to obtain this information, while others depend on the user to explicitly activate the mechanism.

2 Motivation and Contribution

QoS management architectures have been proposed which address the provision of QoS support in networks and end-systems in order to support multimedia applications. Most of these are predicated on the availability of guaranteed services [17, 5, 21]. However it is becoming clear that these types of guaranteed services are not going to be ubiquitously available or be a cost effective solution. Multimedia applications themselves may have fluctuating resource needs. To deal with this adaptive QoS architectures [10, 15] have been proposed where the applications dynamically adjust their resource usage and possibly renegotiate when their resource requirements change.

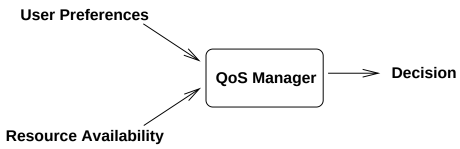

Figure 1: QoS management model

Many of the QoS architectures proposed have concentrated on the implementation of QoS mechanisms in networks and end-systems as well as their coordination in order to provide QoS support to the application. In an heterogeneous environment as described above the QoS assurances provided by the underlying resources may vary. Therefore in such an environment the role of QoS management is to optimally configure the adaptive application making use of the services provided by the underlying resources. In fact we can represent the role of the QoS manager by Figure 1, where it takes adaptive decisions based on user preferences and resource availability. These decisions should take place with minimum user intervention with respect to specifying the user's requirements.

There have been many QoS controlled multimedia applications designed which enable the user to control the QoS of the application. Most of these applications quantise application QoS parameters into a set of values which are then presented to the user. In [9] user QoS for video is parametrised as fast, fast normal, slow, very slow for frame rate and very high, high, medium, low very low for resolution. Similarly in [1] five bands are defined based on a combination of frame rate and resolution. Another approach presented in [20] is termed "Query by example" where a pre-defined set of parameters which correspond to a known experience (for example audio quality is quantified as telephone or CD quality) are defined. The user is also given the opportunity to view samples of these predefined qualities. However in all these approaches it is assumed that the user selects a single point of operation, if this point cannot be sustained the QoS manager will come back to the user to obtain another operating point. This approach is suitable in the guaranteed approach where a single point of operation is negotiated. However in an adaptive approach the operating point may change during operation. Therefore in our approach the user policy specifies a range of operating points which have varying degrees of value to the user. The QoS manager need not keep interacting with the user but control the adaptive application by moving between these points as required. At any time the user may modify the user policy by changing the values assigned to the operating points.

Our contribution is a software framework that facilitates the development of adaptive applications. The framework permits the developer to specify application-dependent rules which govern the behaviour of the application. These rules specify how the user's preferences influence the QoS of the application. We propose cost as the primary control of the user's QoS. The rules also specify the adaptive actions to be taken if resource availability changes. The framework also supplies generic and re-usable resource management modules, that provide abstract interfaces to a variety of network resource classes. We demonstrate the usefulness of our framework with a real application.

2.1 SuperNOVA Application

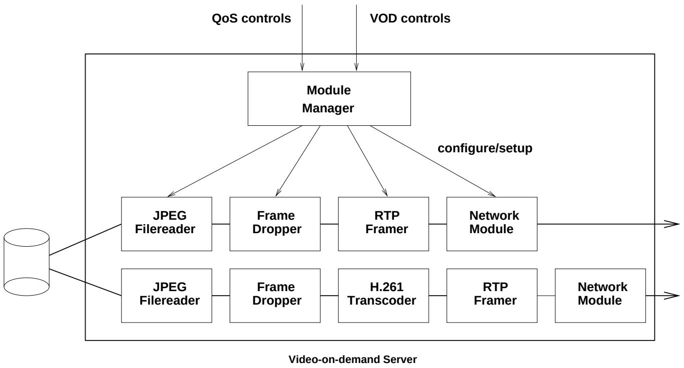

Figure 2: SuperNOVA application

We have built an VoD application called SuperNOVA. This application delivers streams of RTP packetised video and audio. The media clips are stored on disk as high resolution Motion JPEG (MJPEG) files. A block diagram of the VoD server is shown in Figure 2. It incorporates media modules which can be connected into a pipeline. This pipeline can deliver video at varying qualities (In the current prototype we do not scale the audio). Frame-dropper modules enable temporal scaling. A transcoder module converts MJPEG to H.261 on the fly. In the process of transcoding it is also possible to re-quantise the video to a lower resolution. Thus in SuperNOVA we can vary the frame rate and quantisation of the video as well as choose between MJPEG and H.261 codecs which have different bandwidth requirements.

In order for this application to operate optimally it needs to be supported by a QoS framework. We propose a QoS framework to support adaptive applications such as SuperNOVA. This framework implements the QoS management model outline in Section 2. In Section 3 we present an overview of this framework. In Section 4 we identify the generic services that need to be provided by the networks. Since the ultimate arbiter of QoS is the user, in Section 5 we set out the requirements which enable the user to govern the QoS seen by the application. Finally in Section 6 we provide some implementation details of this framework with specific reference to SuperNOVA.

3 QoS Framework

Figure 3 shows a functional block diagram of the framework. The roles of the QoS manager and the resource manager are now discussed in context of this framework, identifying the design decisions taken in the placement of these components.

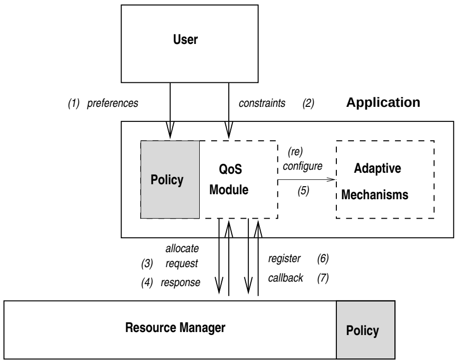

Figure 3: Distributed QoS framework

3.1 QoS Manager

The QoS manager is responsible for configuring the application to its optimal operating point taking into account user preferences and resource conditions. In order to be able to reconfigure the application it requires application specific knowledge about the adaptive mechanisms available in this application and the best policy to apply under various resource conditions. This knowledge is not only domain specific, ie. the requirements for video conferencing differ from that of video on demand, but also may be dependent on user preferences. Thus we choose to implement the QoS manager as an application level entity closely tied to the application. In Section 6.1.1 we show how a generic QoS module can be extended with application specific functionality.

In a distributed application, QoS management can be distributed between client/server. The adaptation can take place at the server (suitable for video-on-demand) or client, or in a network node in the case of multicast applications.

3.2 Resource Manager

The underlying network may provide different classes of service, however by identifying the generic functionality provided by any resource, we can present the QoS manager with a standard interface to the network resource. This interface allows the QoS manager to allocate resources on behalf of the application. The resources also implement a register/call-back interface which allows the QoS manager to be informed when resource conditions have changed. The resource manager implements policies to determine how resources should be shared. These policies would be usually controlled by the administrator of that domain.

It is not our intention to develop new resource management mechanisms. Instead we focus on how the resource management capabilities of new or existing networks can be utilised by the application level framework. h

3.3 Operation

The operation of an application implemented using this framework is explained with reference to Figure 3 and a timing diagram of interactions in Figure 4

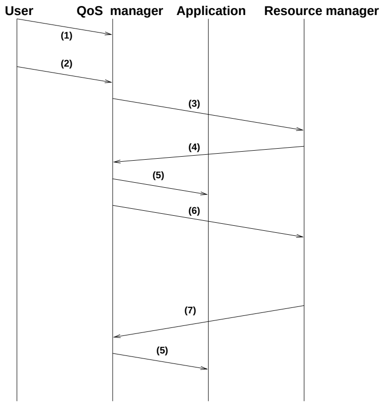

Figure 4: Timing diagram of QoS interactions

The user may express his QoS preferences (1). This would influence the QoS manager's choice and configuration of adaptive mechanisms. The user constrains resource usage (2) to reflect the maximum price he/she is prepared to pay. Based on these constraints and QoS policy, the QoS manager determines an optimum operating point for the application and determines the required resources for this. Based on these requirements it requests a resource allocation (3). The resource manager responds (4) with the actual amount allocated (or available) which may be less than the amount requested. The QoS manager now configures (5) the application's operating point, as determined by user constraints and resource availability. The QoS manager needs to be informed when resource conditions change, so that it can alter the application's operation to react to this. Therefore it registers (6) with the resource manager to receive a call-back when application defined thresholds in resource levels are reached. During the application operation the QoS manager may receive a call-back (7) to indicate that resource conditions have changed. The QoS manager then reconfigures (5) the application according to the QoS policy which specifies how this condition should be handled. The user may also change his QoS preferences or constraints during operation. This would result in new operating point being selected.

4 Resource Management

There are many approaches to providing QoS support within networks. In each approach, the network resources have been classified according to the type of service they provide. The mainstream approaches are specified by the ATM Forum and the Internet Engineering Task Force (IETF) in the intserv and diffserv initiatives.

In both the ATM approach and the intserv approach QoS is specified on a per-flow basis. ATM uses the UNI interface to transmit the application's requirements to the network while intserv proposes using RSVP. ATM uses sender oriented signalling while RSVP is receiver oriented. The diffserv approach on the other hand does not consider traffic on a per flow basis, instead providing the means for aggregated flows to achieve differentiated services by using the defined service classes. There is increasing evidence that both the intserv and ATM approach would not be scalable in the wide area as it requires maintenance of state on a per-flow basis. However it may be used in an intra-net or local area situation. It is likely therefore, that no single approach will prevail. Rather the user would have access to these different types of service depending on the situation. For example in a local area or Intranet, the user may have ATM type service classes whereas in the wide area the Internet model will prevail.

Common to all these approaches is the notion of service classes which will be costed differently. From an end-system perspective we can view the network as providing different classes of service. We attempt to identify the functionality that should be provided by the service and provide a generic interface to them. The generic interface allows the application to choose the resource class it requires based solely on cost and availability considerations.

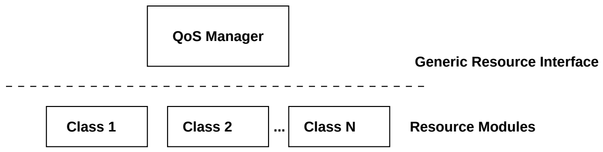

Figure 5: Resource modules

Thus we can represent all resources as a set of service classes as shown in Figure 5. The QoS manager selects the class of service on behalf of the user depending on its availability, application requirements and cost. We design a resource module for each resource type with variants for each of the different service classes. The resource module also encapsulates access to the resource itself, serving as the network end-point or interface to the CPU scheduler. In the generic interface to the resource we need to either provide a interface to the underlying resource's capability or provide the required functionality by some external means, for example an external QoS monitor could be incorporated into the resource module in order to provide the functionality not available in the resource itself.

4.1 Resource Management Functionality

In order to be able to provide a generic interface to the resources we need to identify what resource management functionality needs to be provided.

4.1.1 Allocation

Allocation is the function whereby the resource manager allocates resources based on a request made by the QoS manager.

In order to make an allocation request the QoS manager needs to specify the application's requirements in terms of certain parameters. In the guaranteed model of QoS management this specification is very detailed. For example for network resources it requires the specification of parameters such as minimum, maximum and average rates, burst sizes, end-end delay, loss rate and jitter and for CPU reservations the specification of parameters such as period, deadline and scheduling time. This approach presents a problem for two reasons. Firstly the nature of the application makes the accurate specification of all these parameters both impossible and unnecessary. Secondly it is unlikely that, for example, a network service provider would allow a end-system QoS manager such fine grain control over QoS parameters in the network. Indeed as we said before, the end-system QoS manager will be presented with a set of resource classes with certain characteristics. Therefore in our model we simplify the QoS specification to specify only an amount and define all other parameters as properties of the resource class. In order to specify the amount we use two parameters; maximum, and average. However the resource may interpret the allocation as required, for example by using only the average value.

In the guaranteed model of QoS management the act of allocation implies a contract between the QoS manager and the resource manager where the resource manager guarantees to reserve the requested resources on behalf of the QoS manager. However for heterogeneous environments we need to redefine this meaning. Thus "allocation" in such an environment could mean a hard guarantee, a soft guarantee or even just an indication of how much of the resource is currently available. In all cases the resource manager returns an allocated amount which may be less than or equal to what the QoS manager requested.

Allocation Policy Allocation policy determines how resources are partitioned when resource shortages arise. Associated with each type of resource would be a policy which determines both how the resources are divided between the different classes, as well as between competing requests within a class. Generally we can say that this policy would be determined by the administrator of that resource. However there may be a possibility for the user to modify this policy by overlaying a user policy on top of the system policy. For example in a workstation used by one user it should be possible to install a customised allocation policy for end-system resources.

4.1.2 Policing

Policing is the function whereby the resource manager ensures that the application stays within its declared allocation. Policing could be implemented within the resource, for example networks would incorporate traffic shaping and a CPU schedulers would demote any usage exceeding the allocation to best effort. Regardless of whether policing is implicit in the resource or not it is useful to provide some feedback to the QoS manager regarding the amount of resource it is using. Thus we implement a policing module at the access point to the resource. This enables the application to either redefine its allocation or adapt.

4.1.3 Monitoring

Monitoring is the converse function to policing. Here the resource informs the QoS manager regarding the state of the resource. In best effort and controlled load type of resources, there is no hard guarantee given to the QoS manager. Therefore the resource should inform the QoS manager when QoS parameters change. This helps application to deal with the changes in a controlled manner. This feedback takes place in two ways:

Firstly in event of an overload the resource manager always informs the QoS manager forcing it to take action. In networks overload means congestion. Congestion in the network is possible in the case of both best effort and controlled load type services. Congestion is usually implied by the presence of packet loss. Due to the increasing deployment of routers that deploy mechanisms such as Random Early Dropping (RED), the presence of loss could also imply incipient congestion. Certain network classes could also have an explicit congestion indicator such as the explicit forward congestion notification (EFCN) in ATM ABR or the proposal to include an explicit congestion notifier (ECN) in IP. However the congestion indicator is implemented it is required that the resource module compulsorily indicates overload to the QoS manager and force it to take action. By reacting to overload the application becomes both "net friendly" as well as being able to adapt its QoS according to user preferences rather than losing packets indiscriminately.

Secondly the application may have adaptive mechanisms to deal with the variation of other QoS parameters in the network. These may include the actual percent age of loss, jitter and delay. Thus it can register with the resource manager to be signalled when these parameters reach thresholds that are define ed by the application. For example an application with redundant audio coding would express an interest to know about low levels of loss not amounting to an overload. This would enable it to react to this loss by turning on redundancy. Therefore the

QoS manager should be able to register to receive certain optional QoS indications of interest to the application.

5 QoS Specification

In the previous section we have outlined the functionality that is provided by the resource manager. The role of the end-system QoS manager is to utilise these services to configure an optimal operating point for the application. In order to be able to do so the QoS manager requires a QoS specification. A complete QoS specification consists of several components: a QoS management policy which determines how the QoS manager makes application dependent choices; a QoS mapping which expresses the user/application needs and their translation to lower level resource requirements and finally the specification of QoS constraints which constrain the QoS manager's decisions. Each of these aspects of QoS management arise from factors pertaining different layers in an end-end system. In the current context we can broadly divide them into three layers namely user, application and resource.

5.1 QoS Management Policy

Within the end-system the QoS management policy defines the rules that govern the QoS manager's selection of an optimal operating point for the application. An operating point is defined in terms of the settings of the adaptive mechanisms in the application.

Some examples of adaptive mechanisms for typical audio and video applications are shown in Table 1. It can be seen that activating certain mechanisms have a direct influence on user perceived quality. For example in an audio application changing the audio codec while reducing/increasing resource usage, may also affect the audio quality as perceived by the user. In a video application changing the video frame rate or resolution will be perceptible to the user. Other mechanisms cannot be perceived by the user. For example the application could switch video codecs to tradeoff CPU usage for bandwidth usage, and the user will not perceive anything as long as the visual quality remains a constant. Some mechanisms may be perceptible to the user, but the user will not wish to control their activation directly. For example the turning on of audio redundancy may be perceptible to the user in the form of a better quality under lossy conditions, but the user would wish to leave that decision to the QoS manager.

Thus it can be seen that adaptive mechanisms fall into two categories. Those which are perceptible to the user, or those which the user would wish to control, and those which can be activated without interaction from the user. Therefore QoS management policy can be sub-divided into two categories, user policy - which determines how the application's operating point is optimised to maximise the user's satisfaction with it, and resource policy - which specifies how the best operating point for the current resource conditions is determined.

5.1.1 User Policy

User policy determines the course of action adopted by the QoS manager when a user indicates a change in current delivery "quality" is necessary. In other words this requires the selection of an operating point such that the user's satisfaction is maximised.

Contrary to appearance, user policy is not a simple issue. There are a number of interrelated factors needing consideration:

5.1.2 Combination of Multiple Quantities

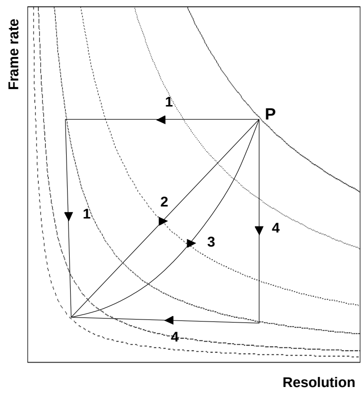

Figure 6: Degradation path

User policy defines the user's relative satisfaction with the quality of each the possible operating points of the application. An ordering in terms of user satisfaction may be straightforward in certain applications. For an audio application where the only mechanism available is changing the codec, an ordering of the codecs according to decreasing audio quality is possible. However in an application where multiple quantities, and hence mechanisms contribute to the user perceived quality the situation is more complicated. This is illustrated in Figure 6 showing a video application which can scale its frame rate as well as its resolution. Point P represents the maximum quality. Path (1) shows an ordering of operating points (degradation curve) if frame rate is always considered to be more important than resolution. In path (4) resolution considered to be more important than frame rate and in path (2) both are considered to be equally important. Path (3) represents an arbitrary combination.

5.1.3 Content dependence

	T_{lo}	T_{hi}
$A_{lo}V_{lo}$	Logo/test pattern	Station break
$A_{lo}V_{hi}$	Snooker	Sporting highlights
$A_{hi}V_{lo}$	Talk show	Advertisement
$A_{hi}V_{hi}$	Stand-up comedy	Music video

Table 2: Video Classification Examples

The determination of a degradation path as shown above would be dependent on the content of the media itself. Apteker et al [2] have conducted a study in which they classified video content in terms of their content. They used three dimensions. The temporal nature of the data (T), the importance of auditory information (A) and visual components (V) to understanding the message. Using this scheme, they classified movie clips into 8 categories by assigning two levels (lo and hi) to each of the dimensions.

Video	
Mechanism	User Perception
frame rate	visual quality
size	visual quality
colour	visual quality
resolution	visual quality
CODEC	-

Audio	
Mechanism	User Perception
Packet size	interactivity
Redundancy	-
CODEC	aural quality

Table 1: Mapping QoS Mechanisms to User Perception

Table 2 shows some example videos classified according to this scheme. They then carried out a watch-ability test on video in each of the categories at three different frame rates. The conclusion was that watch-ability at the three frame rates was dependent on the clip content. While this study was only for frame rate, it is likely that a combination of parameters would result in a similar outcome.

5.1.4 Context dependence

The user policy may also be dependent on the context in which the application is being used. In *vat*, there are two modes of operation which determine the play-out buffer strategy to be using depending on whether the application is being used in an interactive discussion - in which case a low delay is desirable; or for listening to a lecture - in which case a high delay is acceptable as a means of reducing loss.

5.2 Resource policy

The QoS manager should also react to changes in resource availability and resource related QoS events emanating from the resource manager. The occurrence of these events are beyond the control of the user. However the QoS manager needs to define some rules which govern how it reacts to these events. This is termed the resource policy. For example on receipt of a congestion indicator, an audio application may choose to turn on redundancy to deal with certain levels of loss. However in other applications the event action may simply be to degrade "quality" in order to consume less of that resource. Similarly an application may have a specific action to deal with changes in resource availability. For example a video application may switch to a codec with a more aggressive compression scheme but higher CPU requirements for decoding, when available network bandwidth is less than a certain value.

5.3 QoS Translation

At the resource level QoS is expressed in terms of parameters such a bandwidth, delay and jitter for network resources and period and duration for CPU resources. At the application level, QoS is expressed in terms of application configuration parameters such as frame rate, video resolution and size for video and quality (codec) and interactivity (sample size) for audio. An adaptive application is by definition able to operate at different operating points, each having differing resource requirements. These operating points are defined using application parameters such as frame rate and resolution. However in order for the QoS manager to allocate resources on behalf of the application, a translation between application QoS and resource level QoS is required. For an adaptive application with many operating points this translation should exist for all its operating points.

There have been two main approaches to QoS translations. In one approach attempts have been made to derive a mapping function between application level parameters and system or resource level parameters. For example in [9] mapping function which converts application QoS to transport QoS for an application known as XMOVIE is presented. However these functions are only valid for specific applications. Others have assumed that this mapping can be obtained via measurement. In [10, 17] the translated values are measured in pre-runs and stored in a database. Similarly in [1] 5 bands of video based on user perceived QoS were determined and the required resources to support these qualities were measured. In general, the issue of providing a mapping for all types of multimedia applications is non-trivial. Certain applications can easily determine their resource needs - an audio application uses a constant network bandwidth which is dependent only on codec selected. However determining resource usage for other applications such as variable bit rate video application is more difficult. Therefore in general it should be assumed that this mapping may be partial or inaccurate. Therefore QoS frameworks that require an accurate mapping [17, 5] to exist cannot be applied generically to all types of applications. Rather the QoS framework should explicitly deal with the question of partial/inaccurate mapping [15].

We have taken the approach that a translation should be provided by the application designer in either form. However this translation may be inaccurate Therefore provision is made for measurements to be used to refine the mapping in future runs. Further the application adapts to the resource allocation rather than expecting the initial resource allocation to be accurate.

5.3.1 Translation of User level QoS

There is wide variance in the literature as to what constitutes user level QoS. In some QoS architectures the user QoS is simply resource QoS [5], while in others it is the application QoS [17]. In [9] it is a combination of application and resource level parameters.

The issue here is not user interface design. Rather, it is concerned with the provision of an intuitive and simple measure of "satisfaction" with the perceived quality. We argue that user level QoS should represent the user's ultimate satisfaction. Thus we represent user QoS as a single parameter, "satisfaction". The translation of application level QoS parameters to this "satisfaction" value is based on the user QoS policy. We use this single value as an internal mechanism for ordering operating points as well as a means of providing feedback to user.

5.4 Constraints

The availability of resources and the allocation policy would provide a constraint to the QoS achieved by the application.

In addition it is likely that the user would wish to constrain resource usage in order to reduce cost. Without the notion of cost there is no incentive for the user to specify anything other than maximum quality. Pricing policies for allocating network resources have been proposed in [11, 7]. In [11] the price is based on demand. In [7] the concept of graduated prices for multi-class services is proposed. Since we are operating in a heterogeneous environment the latter form of pricing policy seems more suitable. The concept of cost can also be extended to other resources such as CPU usage on a server. Therefore cost can also be used to combine resource usage into a single parameter to be controlled by the user.

5.5 User control of QoS

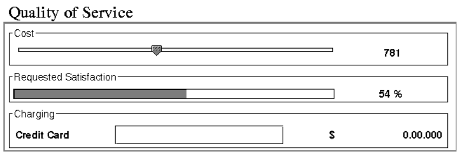

Figure 7: User control of QoS

In the previous sections we have described the components of QoS specification pertaining to the user level. These were the user QoS policy which determined how the user's perception of quality was taken into account (Also represented as a translation from application level QoS to a single satisfaction value at the user level), and user level constraints which is represented by cost. We need to design a suitable graphical user interface to enable the end user to manipulate these values. For acceptability and ease-of-use the fewer control quantities the better, a single "knob" being optimal. Therefore we propose the following simple interface shown in Figure 7. Here we use the cost constraint as the users primary means of controlling QoS. The "satisfaction" meter provides feedback to the user by displaying the satisfaction as calculated according to user policy.

We have defined user policy as that which embodies user QoS. User policy may be specified by the application designer for certain applications. For example for VoD it is possible to pre-determine a translation based on perceptual tests. However in other applications it may be necessary for the user to modify user policy at run-time. This enables the user to take context dependent actions. Therefore we propose an additional interface with a suitable GUI which lets the user modify user policy. This is application dependent and left up to the application designer.

6 Implementation

We have developed a prototype implementation of our QoS framework using Java. The object oriented nature of Java allows us to build components that can be customised by the application developer by virtue of extending the base component. Thus the framework is a set of Java classes for use by the application developer. We have currently implemented only network resource modules (ie. only one type of resource). In the following sections we identify the main software components in the framework and discuss how the generic components can be extended with application specific information with special reference to the SuperNOVA application.

6.1 Application

The adaptive application first needs to be implemented by the application designer. Depending on the types of media it supports it would include adaptive mechanisms that would enable it to adapt to changing resource conditions. In order to interact with the QoS manager it needs to implement the following interfaces. A `setOperatingPoint` interface enables the QoS manager to configure the application with the optimal operating point. `OperatingPoint` is passed opaquely to the application and is interpreted in an application specific way. A `setEndPoint` interface enables the application to use the network endpoint allocated by the QoS manager. Instead of developing an application from scratch it is also possible to wrap an existing application to provide these interfaces.

6.1.1 QoS Manager

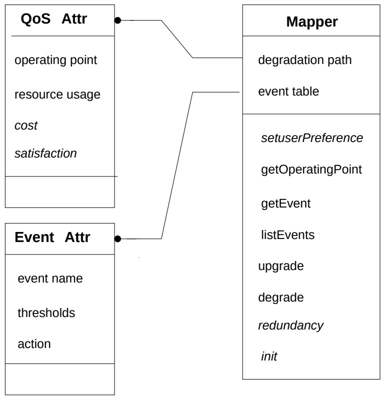

Figure 9: Mapper class diagram

The QoS manager performs the role of a coordinator between the user application and resource manager. This functionality is generic. However it also needs knowledge of the application specific QoS policy in order to make optimum decisions, this information needs to be provided by the application designer. Thus the QoS manager is implemented as two components, the generic QoS module performs the coordination function, while the mapper module encapsulates application specific policy and needs to be pre-configured by the application designer.

The implementation of the QoS manager can be best explained by considering its operation in two phases, ie. the

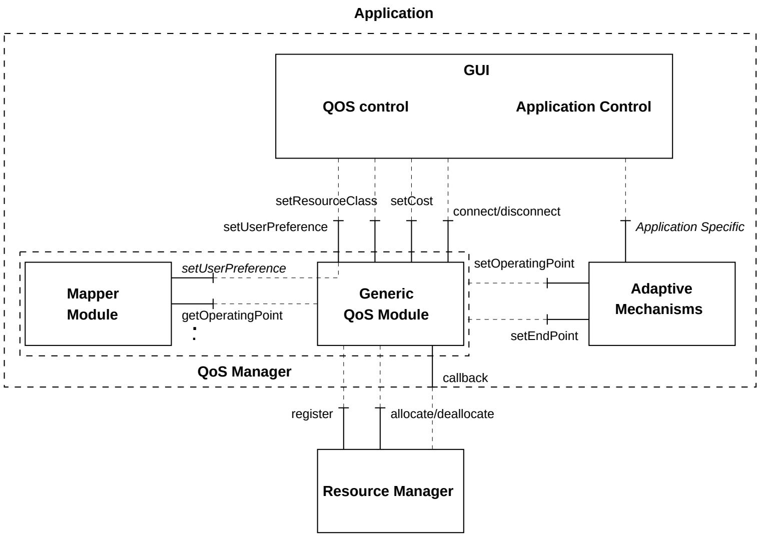

Figure 8: Framework implementation diagram

configuration phase and the operating phase. In the configuration phase the application specific mapper module needs to be configured. Figure 9 shows the class diagram of the mapper module. It contains a `degradation_path` which is dynamically built at run time. This is an ordered list of operating points according to cost. In order to build this table, the following information is needed.

- A QoS translation which is a list of all operating points and their corresponding resource requirements. An initialisation method which sets up this translation needs to be provided. This could be a mapping function or a database. In SuperNOVA we measure consumption at highest bandwidth and resolution. Since these two parameters are independent, and assuming a linear relationship between any parameter and bandwidth, we can obtain the network bandwidth for any given combination.

- A cost function which is used to calculate the cost of each operating point. This is dependent of the resource class which can be selected by the user using `setResourceClass`.

- The user QoS policy determines the choice of operating point if multiple operating points have the same cost. In our framework this is implemented via the `satisfaction` value which represents an ordering of operating points according to increasing "Quality". This can be initialised by the application designer or set by the user. To enable the latter the `setUserPreference` method needs to be implemented and a suitable GUI front end for it provided (See Section 6.1.2).

Using this information a `degradation_path` can be calculated. The `getOperatingPoint` selects a operating point corresponding to a particular cost. The `degrade` and `upgrade` methods move up or down the `degradation_path` from the current operating point.

The resource policy is implemented via the `event_table` This lists QoS events, thresholds at which the application should be informed and the action to be taken. An initialisation method for the `event_table` needs to be provided. The default action is upgrade or degrade. If any other action is specified this needs to be implemented by the programmer. For example, a redundancy method could be implemented which returns a new operating point with redundancy turned on. In SuperNOVA the event policy defaults to degradation. Therefore in event of loss we degrade to the next level of service.

6.1.2 Graphical User Interface (GUI)

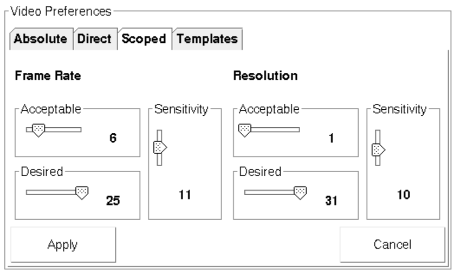

Figure 10: Optional QoS controls

The application designer also needs to provide a suitable GUI for the application. This consists of two parts: a QoS management GUI (which interfaces with the QoS module) and an application dependent component, which interfaces with the application in an application specific way. For a VoD application such as SuperNOVA the latter GUI would display a list of videos available at the server as well as standard VoD controls.

In Section 5.5 we have described the simple QoS control GUI which is generic to all applications. This allows the user to use a slider to constrain cost. The user may also wish to select a resource class via the `setResourceClass` interface. These operations would be sufficient for the novice user and represent a simple easy to understand user interface. The framework provides a hook for the application designer to provide a suitable GUI for enabling the user to have a more fine grain control over QoS. A more experienced user may wish to modify user policy. A possible interface would allow the user to select from a list of predefined templates which correspond to policies suitable for different classes of clips. As a step towards developing such templates for we have designed an additional GUI (See Figure 10) meant for expert users (or content providers). In SuperNOVA we have derived mechanisms for mapping multiple application parameters (such as frame rate and resolution) into a combined satisfaction value [19]. Using this interface the content provider may set minimum, desired and sensitivity (importance) values for each of the application level QoS variables.

6.2 Resource Manager

6.2.1 Interfaces

As detailed in Section 4 we define the generic functionality that needs to be provided by the resources. The following is the generic interface that should be presented by any resource module in this framework.

- `RsrcAmt resamt allocate(RsrcAmt reqamt)`
  The resource manager attempts to allocate `reqamt` on behalf of the application. `resamt` returned is the actual amount allocated.

- `deallocate()`
  Deallocate resources

- `register(QoSParam type, Callback fn, int lowThrs, int highThrs)`
  The QoS manager registers to receive a call-back when the QoS parameter `type` reaches a pre-defined threshold. The QoS manager always registers to receive call-backs when overload occurs. Registering to receive other call-backs is optional.

6.2.2 Network Resource Module Implementation

In order to demonstrate how the generic interface operates, we have implemented resource modules for two different classes of network services. To this end we have implemented resource modules for a best effort class (UDP/IP) and a guaranteed class (ATM CBR) respectively.

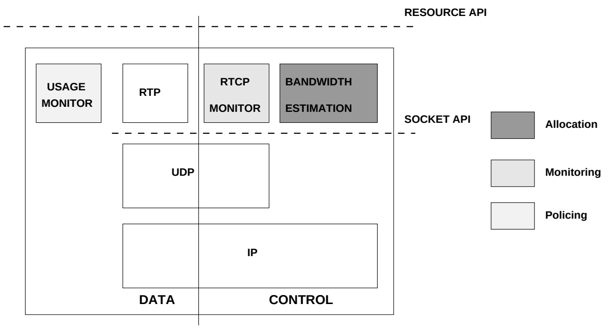

Figure 11: IP resource module design

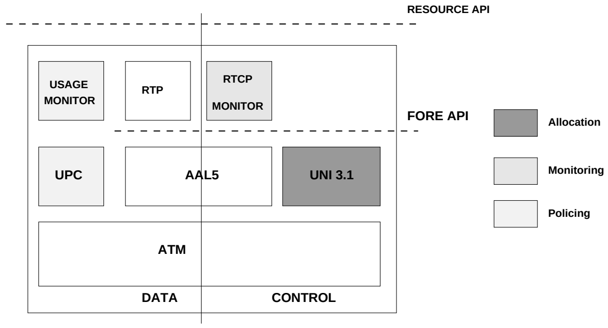

Figure 12: ATM resource module design

Figure 11 and Figure 12 shows how the functionality is implemented in the IP UDP and the ATM CBR modules respectively. Let us now consider the mechanics used to implement each of the functionalities.

Allocation In the ATM case the allocation function is implemented by using signalling via the ATM forum's UNI3.1 [3] during connection set up. No such allocation capability exists for UDP/IP, but it is possible to match the bandwidth allocation request to the capacity currently available. This provides the application with a starting point. Thus it can set its initial data rate in a manner that does not congest the network as well as obtaining the best QoS possible. The UDP/IP resource module implements a bandwidth estimation module. In order to implement this we use a series of probe packets, this is similar to the work described in [8] and [4].

Policing ATM CBR performs usage parameter control (UPC) to police the traffic to the parameters set up at connection time. However there is no means for the QoS manager to know that such policing has taken place. Since the application specification of its resource usage may be inaccurate we provide a simple measurement module for both ATM and IP. This maintains measurements of network usage and informs the QoS manager via the call-back interface if there is a large deviation from the requested allocation.

Monitoring There is no support in either protocol stack for per-stream QoS statistics. Since for streaming media we use the RTP protocol in both cases, we can access the functionality provided by RTCP. We have implemented RTP as a library and the resource module has access to the RTCP receiver reports generated at the receiver end. Using the fractional loss value we define a loss of 10% as indicative of congestion and inform the QoS manager via the call-back interface. In addition the QoS manager may register to be informed when loss, delay and jitter reached defined thresholds which are set by the QoS manager.

7 Conclusion

We have described our framework and its implementation as well as a sample a Video on Demand application developed using it. We feel that the building of this application demonstrates the viability of the framework. The video mapper module combines with a simple-to-use interface to provide sophisticated mapping between user preferences and resource management. In implementing this we have also presented a multi-level interface that provides the user with granularity of control to suit individual inclinations. Anecdotal feedback indicates satisfaction with the usability of this interface from a variety of users. Finally we feel that the generic resource management modules we have implemented are easily re-usable by other applications. However this has yet to be demonstrated by further application development.

There are a number of directions for future work. User perception tests are required to verify the user interface and the appropriateness of the mapping algorithm. The framework requires further validation through the development of different applications. We are investigating a distributed framework which could be used, for example, for application level QoS management of multicast applications.

8 Acknowledgements

The authors would like to acknowledge and thank Anthony Richards and Glynn Rogers of CSIRO for their discussion and ideas on user level QoS translation.

The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science & Tourism of the Commonwealth Government of Australia.

References

[1] M. Alfano and R. Sigle. Controlling QoS in a Collaborative Multimedia Environment. In *Proc. 5th IEEE International Symposium on High-Performance Distributed Computing*, Aug. 1996.

[2] R. T. Apteker, J. A. Fisher, V. S. Kisimov, and H. Neishlos. Video Acceptability and Frame Rate. *IEEE Multimedia*, 2(3), Fall 1995.

[3] The ATM Forum Technical Committee. *User-Network Interface (UNI) Specification Version 3.1*, 1994.

[4] J. C. Bolot and T. Turletti. A Rate Control Mechanism for Packet Video in the Internet. In *Proc. IEEE INFOCOM'94*, pages 1216–1223, Toronto, Canada, June 1994.

[5] A. Campbell, G. Coulson, F. Garcia, D. Hutchinson, and H. Leopold. Integrated Quality of Service for Multimedia Communications. In *Proc. IEEE INFOCOM'93*, San Francisco, Mar. 1993.

[6] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM Computer Communication Review*, 22:92–97, July 1992.

[7] R. Cocchi, D. Estrin, S. Shenker, and L. Zhang. A Study of Priority Pricing in Multiple Service Class Networks. In *Proc. SIGCOMM Symposium on Communications Architectures and Protocols*, Sept. 1992.

[8] M. Crovella and R. Carter. Dynamic Server Selection in the Internet. In *Proc. 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, August 1995.

[9] S. Fischer and R. Keller. Quality of Service Mapping in Distributed Multimedia Systems. In *Proc. IEEE International Conference on Multimedia Networking*, pages 132–141, Aizu-Wakamatsu, Japan, Sept. 1995.

[10] M. Fry, A. Seneviratne, A. Vogel, and V. Witana. Delivering QoS Controlled Continuous Media on the World Wide Web. In *Proc. IWQoS'96*, pages 115 – 124, 1996.

[11] E. Fulp, D. Reininger, M. Ott, and D. S. Reeves. Paying for QoS: An Optimal Distributed Algorithm for Pricing Network Resources. In *Proc. IWQoS'98*, 1998.

[12] IETF Secretariat. Differentiated Services (diffserv) Charter [web page]. `http://www.ietf.org/html.charters/diffserv-charter.html`, 1996.

[13] IETF Secretariat. Integrated Services (intserv) Charter [web page]. `http://www.ietf.org/html.charters/intserv-charter.html`, 1996.

[14] V. Kumar. *MBone: Interactive Multimedia On The Internet.* Macmillan Publishing (Simon & Schuster), 1995.

[15] K. Lakshman and R. Yavatkar. AQUA: An Adaptive End-System Quality of Service Architecture. *High-Speed Networking for Multimedia Applications*, Nov. 1996.

[16] S. McCanne and V. Jacobsen. vic: A Flexible Framework for Packet Video. In *Proc. of ACM Multimedia*, pages 511–522, San Francisco CA, Nov. 1995.

[17] K. Nahrstedt and J. Smith. A Service Kernel for Multimedia Endpoints. In *Proc. IWACA '94*, Sept. 1994.

[18] C. Perkins, V. Hardman, I. Kouvelas, and A. Sasse. Multicast Audio : The Next Generation. In *Proc. INET 97, Kuala Lumpur, Malaysia*, June 1997.

[19] A. Richards, G. Rogers, M. Antoniades, and V. Witana. Mapping User Level QoS from a Single Parameter. In *Proc. International Conference on Multimedia Networks and Services (MMNS '98)*, Nov. 1998.

[20] A. Vogel, G. v. Bochmann, R. Dssouli, J. Gecsei, and A. H. B. Kerhervé. On QoS Negotiation in Distributed Multimedia Applications. Technical Report #891, Université de Montréal, 1993.

[21] C. Vogt, L. C. Wolf, R. G. Herrtwich, and H. Wittig. HeiRAT – Quality-of-Service Management for Distributed Multimedia Systems. *Special Issue on QoS Systems of ACM Multimedia Systems Journal*, 6(3):152–166, May 1998.