Management of Application Quality of Service

Patrícia Gomes Soares Florissi and *Yechiam Yemini*

Distributed Computing and Communications (DCC) Lab
Computer Science Department Technical Report CUCS-022-94
Columbia University, New York, NY 10027
{pgsf,yy}@cs.columbia.edu

Abstract

This paper proposes a new language for the development of distributed multimedia applications: Quality Assurance Language (QuAL). QuAL abstractions allow the specification of Quality of Service (QoS) constraints expected from the underlying computing and communication environment. QuAL specifications are compiled into run time components that monitor the actual QoS delivered. Upon QoS violations, application provided exception handlers are signaled to act upon the faulty events. Language level abstractions of QoS shelter programs from the heterogeneity of underlying infrastructures. This simplifies the development and maintenance of multimedia applications and promotes their portability and reuse. QuAL generates Management Information Bases (MIBs) that contain QoS statistics per application. Such MIBs may be used to integrate application level QoS management into standard network management frameworks.

1 Introduction

Services provided by traditional network stacks embody generic transmission management (monitoring, analysis, and control) mechanisms. The communication timing constraints of distributed multimedia applications undermine the use of such generic approaches. Consider, for example, a real time video transmission. First, loss control through message retransmission as employed by a connection oriented service (such as TCP [comer91]) is not suitable in this case. Retransmission causes an additional communication delay that results in abrupt disruptions in video flow. Second, video transmissions require the management of jitter, a *Quality of Service* (QoS) constraint not required by traditional data transfers. Third, some video applications recover from loss by replicating previous frames, while others recover by reducing the display rate. Therefore, the handling of violations is dependent on the specifics of the application being designed. It is thus important to establish effective technologies to ensure application customized management of the several QoS required by distributed multimedia applications. In what follows, the major questions addressed by this paper are identified.

How should programmers specify the QoS requirements of applications? Consider a speech communication application. How can a programmer specify a QoS constraint on the level of jitter that may be tolerated? How will such specification be compiled into effective run time mechanisms to enforce such constraint? A central difficulty is that QoS constraints typically relate to temporal behavior of programs and their communications. Therefore, a language to specify QoS must express constraints on real time behavior of a program and its communications. These constraints, furthermore, must be evaluated in real time.

How can the actual QoS delivered be monitored? Network entities use a *best-effort* approach to deliver QoS, i.e., there is no guarantee that services will deliver the QoS requested. Applications must be able to capture from the communication streams the actual QoS delivered. For example, when a connection is established with maximum end-end delay of 20ms, message delay is the measure to be monitored. Messages must be instrumented with extra information that enables the receiving end to reconstitute the actual end-end delay for each message. But, how can applications specify what must be monitored?

Furthermore, how can the specification be implemented by automatically instrumenting message streams and by accessing the information provided by the underlying services?

How can applications detect and respond to QoS violations? Violations and associated handling mechanisms are dependent on the specific application being developed. For example, a video application may decide to ignore previous frames upon loss, while a data transfer application may decide to request retransmission. The former has strict end-end delay requirements while the latter needs to deliver accurate data. How can they recover from violations in a customized, fast, and graceful manner? Furthermore, it is desirable to support these features without severely sacrificing performance.

How can external network management systems share control of QoS with applications? An accurate network management must include applications and their QoS related events. Consider an application that is displaying video at 100Mbits/s. Assume now that the bandwidth cannot be delivered because one of the links failed. Network management entities must be informed that application connections are not delivering the negotiated QoS so that they can act by finding an alternative path to re-establish the negotiated rate. This information is richer than normal link failure detection performed by network managers because it enables the latter to track which applications were affected by the surge. Application and network manager interactions complete information on the actual QoS failure and affected entities. To achieve this exchange, how can application internals be disclosed to management entities without incurring extra development overhead?

This paper overviews the *Quality Assurance Language* (QuAL), a new programming language that explicitly incorporates data types and constructs to abstract QoS related events occurring in a distributed system. A distributed application is viewed by QuAL as a set of autonomous processes that communicate by message exchange. QuAL extends the process model to associate QoS constraints to communications and computations. QuAL specifications are compiled into run time components that monitor the actual QoS delivered. Upon QoS violations, application provided exception handlers are signaled to act upon the faulty events. QuAL also provides means to automatically generate *Management Information Bases* (MIBs) that contain data associated with the performance of the QoS demanding computing and communication activities of applications running on a system.

This paper is organized as follows. Sections 2 and 3 describe QuAL abstractions to support the assurance of communication and processing QoS. Section 4 describes how QuAL automates the generation of management instrumentation of applications. Section 5 reviews related work. Finally, Section 6 concludes.

2 QuAL Supports Communication QoS

QuAL provides the means to specify and manage two classes of QoS communication constraints: network level and application level. Network level QoS measures are related to network dependent communication properties, such as maximum propagation delay in a link. Application level measures, on the other hand, represent application dependent constraints that must be monitored and enforced on communicated streams, such as message generation rate. QuAL also provides operators to enable dynamic re-negotiation of QoS parameters. A re-negotiation can happen at any time during process execution and it will occur concurrently with the sending and arrival of messages.

The examples described in this paper use Concert/C [auerbach92] as the base process oriented language and adopt the following conventions. Keywords and constructs in QuAL are written using **bold** face, in Concert/C are underlined, and in C [kernighan88] are written using plain text.

2.1 Specifying and Monitoring Network Level QoS Measures

QuAL provides the following abstract QoS attributes for the specification of network level QoS measures: loss tolerance, permutation tolerance, maximum end-end delay, maximum inter-message delay, average transmission rate, peak transmission rate, and recovery time. Consider, for instance, two processes that want to communicate video frames in real time, as illustrated in Example 2.1. The receiver process defines the input port (inport for short) `video_input` that receives messages of type `video_frame_t`, whereas the sender process defines the output port (outport for short) `video_output` that sends messages of the same type (the code for the processes are omitted in the example due to space limitations). The sender process samples a video camera and sends the sampled data through `video_output`. The receiver process, on the other hand, displays locally

the data coming through video_input. The example shows the specification of the network level constraints that each process requires from the communication.

QuAL extends the declaration of a port with the **realtm** clause that contains the specification of the QoS measures of interest. Most specifications of QoS measures in QuAL are expressed using an *interval of acceptance* that must contain the QoS delivered by the underlying system to applications. In Example 2.1, for instance, the message transmission rate of video_input must be in the interval between 10 and 15 messages/sec.

```
% Receiver Process
realtm { loss 6;          /* The percentage of messages lost during transmission
                          must not be higher than 10^-6 */
     permt;               /* Permutation is allowed in the transmission, i.e., messages
                          need not to be delivered in the order they were sent */
     rate  sec 10 -
           sec 15;        /* Mean transmission rate is between 10 and 15 messages/sec */
     peak - sec 20;       /* Peak transmission rate is 20 messages/sec */
     delay ms 35;         /* Transmission delay must be less than 35ms */
     inter-delay ms 33;   /* Inter-message delay must be less than 33ms */
     recovery sec 3;}     /* Any recovery must take less than 3sec */
  handlers { net_handler
        {manage_conn;};}  /* Port that receives network level QoS violations */
  receiveport             /* Keyword that identifies a port declaration */
    {video_frame_t}       /* Type of messages exchanged through the port */
    video_input;          /* Port identifier */

% Sender Process
realtm { loss NULL;
     permt NULL;          /* No constraints regarding loss or permutation */
     rate - sec 25;       /* Message mean transmission rate is 25 messages/sec */
     peak - sec 30;       /* Peak rate is 30 messages/sec */
                          /* No constraints regarding delay or inter-message delay */
     recovery sec 4;}     /* Any recovery must take less than 4sec */
  receiveport {video_frame_t}
    *video_output;        /* The symbol * distinguishes an outport from an inport */
```

Example 2.1: Specifying Network Level QoS Measures

The binding mechanism in QuAL guarantees that only ports with *compatible* QoS attributes are connected. Two ports have compatible network level QoS attributes if the intersection of their acceptance intervals is non null for each QoS attribute specified. This is the case for video_input and video_output in Example 2.1. Their intersection defines a mean rate between 10 and 15 messages/sec, a 20 messages/sec peak, recovery time less than 3 sec, loss no higher than 10^{-6}, and arbitrary permutation.

It is the responsibility of the runtime to map abstract QoS demands into lower level service requests, sheltering heterogeneity at the network layer. For example, the runtime maps transmission rate constraints into protocol specific requests to allocate bandwidth and buffers.

In QuAL, the choice of the communication protocol for a connection is delayed until run time, when two ports are actually bound. In this manner, the choice can be based not only on the static aspects of the communication (such as loss tolerance) but also on properties only known at run time. For example, properties such as what protocols are supported by the communicating machines and whether communication will occur over a local or a wide area network can only be determined during run time.

The language runtime is also responsible for monitoring connections based on their QoS demands. More specifically, the runtime adds control information (such as message sending time) to messages being transmitted, enabling the computation of the mean transmission delay, mean jitter, mean transmission rate, and peak transmission rate. If any QoS violation occurs, an exception is raised by the runtime. QuAL raises exceptions by sending *exception messages* that report the exception. Exception

messages are sent to designated exception handler ports, specified in the **handlers** clause of a port declaration. In Example 2.1, an exception message is sent to port `manage_conn` whenever a message propagation delay is higher than 35ms or the transmission rate is lower than 10messages/sec. The receiver process is responsible for checking `manage_conn` and for controlling the message display time based on the violations signaled.

2.2 Specifying and Monitoring Application Level QoS Measures

QuAL provides mechanisms to assist applications in managing (that is, monitoring and controlling) the actual QoS delivered by peer applications and by the network. QuAL uses a *contract identifier* to represent a set of constraints that a port must comply with. A contract identifier in QuAL denotes communication management in the same way a function name denotes the computation performed by a function. Consider, for example, a broadcast of JPEG coded video to three destinations (A, B, and C) that have different amounts of communication and processing resources available, and therefore, different transmission requirements. The communications between the sender and the receivers A, B, and C could be managed according to the contracts `low`, `med`, and `high`, respectively, that represent the management that provides low, medium, and high quality transmission.

From the application level point of view, the runtime creates *Communication Monitoring Processes* (CMPs) for each communication, as illustrated in Figure 2.1. Processes are represented by big rectangles, inports by small black rectangles, and outports by small black circles. The straight arrows represent information flow between processes. A message sent by an application level process is first intercepted by a CMP that checks a particular contract. Only the messages that comply with such constraints are sent to a second CMP that monitors the network level QoS constraints discussed in Section 2.1. This latter CMP then delivers data to the network. For example, the communication with receiver A is managed according to the contract `low` by CMPlow and by the network level constraints of the connection between the sender and receiver A by CMPnlow.

A CMP checks compliance with a contract by calling an associated *monitoring function* defined by the programmer. Monitoring functions have no access to the contents of a message. Instead, they receive as arguments an index that identifies the order of a message in the stream being communicated, and the sending and arrival times of the respective message. A monitoring function returns a value that indicates whether the message identified by the function arguments complies with the constraints being monitored or if an exception should be raised. Messages that do not comply with the contract must be removed from the communication. Based on the value returned by the function, a CMP generates the appropriate exception messages and decides where to send the message next (either to an exception handler port or to the CMP that checks network level constraints). The concept of CMPs, however, is just an abstraction provided to application developers. For performance reasons, the QuAL runtime implements the monitoring functions in the same physical address space of applications.

To adjust the quality of a video stream according to the resources available, the monitoring function that checks the contract `low` could implement, for example, *media scaling*. Scaling consists of sampling a message stream and transmitting only the fraction sampled, assuming that the sampled data represents a good enough approximation of the original information. In this case, the monitoring function would monitor the rate in which messages are generated by the sender and decide to uniformly drop messages to reduce the rate. Similarly, the monitoring function associated with the contract `med` would do the same, but the sampling frequency would be higher if more resources are available. On the other hand, the monitoring function associated with the contract `high` would not drop any message, and only monitor the generation rate. The function could raise exceptions whenever the rate violates the high quality transmission requirements.

QuAL can also help in managing inter-stream constraints, that is, constraints that involve more than one stream. Consider, for example, a receiver process with two connections that must be synchronized, one for audio and the other for video. In QuAL, the language runtime generates a single CMP that intercepts the traffic of both streams and it is thus able to compare the inter-stream dependencies.

To summarize, application level constraints in QuAL can be related to a single stream (intra-stream constraints), or to a group of streams (inter-stream constraints). Application level constraints in general can be associated to outports and to inports. Constraints associated to outports are checked as messages are sent whereas the ones associated to inports are checked when

messages are received. The details regarding the syntax for the specification of contracts and associated monitoring functions can be found in [florissi94].

3 QuAL Supports Processing QoS

QuAL adds real time language constructs to enable the expression of processing timing constraints, such as deadlines to process messages. Example 3.1 illustrates the real time processing of video in the context of a multimedia conferencing application. QuAL introduces the **`within`** *real time block* that consists of a sequence of instructions and execution timing constraints associated with it. A **`within`** block is only executed if there are enough processing resources available for its execution according to the timing constraints specified. Otherwise, control is passed to the statement following the **`until`** condition. The

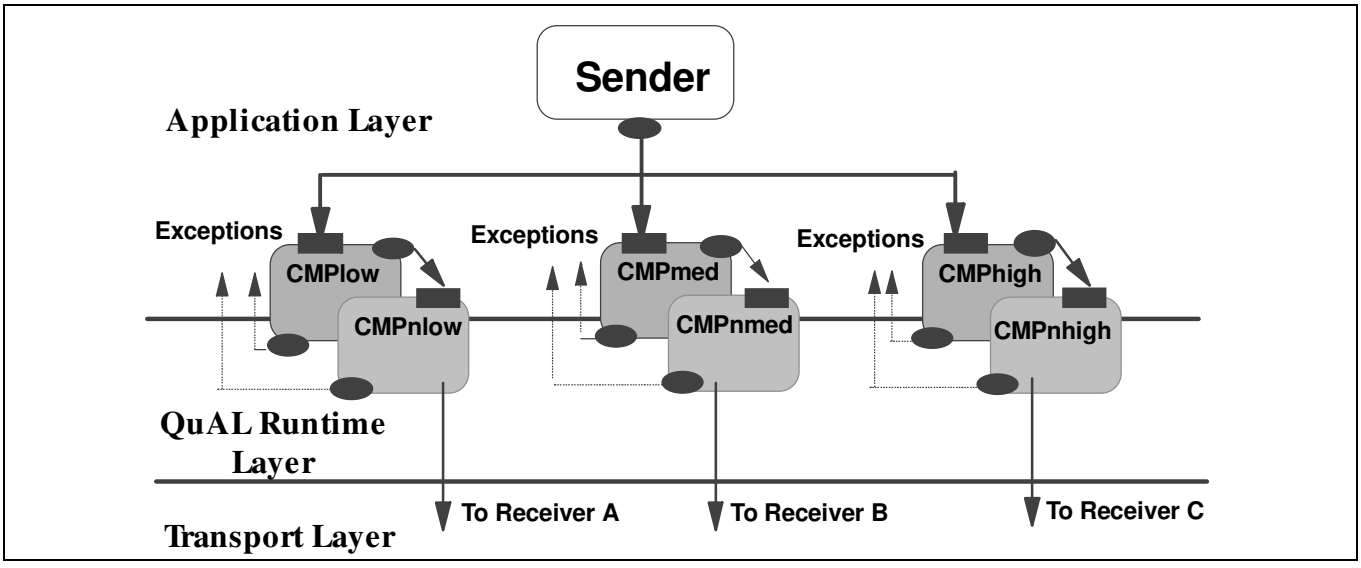


Figure 2.1: Broadcasting Data in QuAL

`do` block represents the task that must be performed in real time repeatedly until the **`until`** condition is true.

The timing constraints determine if the block must be executed *periodically, sporadically*, or *aperiodically*. Furthermore, timing constraints can be *hard or soft*. Periodic activities are those which have to be processed at regular intervals, and must be completed before the next is due. Sporadic activities are asynchronous activities that have minimum inter-arrival periods between occurrences. Aperiodic activities are asynchronous and have no minimum inter-arrival period between occurrences.

Hard constraints must be met or else the application will deliver wrong results. Soft constraints indicate ideal targets which should be met when the system is not overloaded, but that can be missed occasionally. In Example 3.1, the timing constraints specify a sporadic soft mode of execution, with a maximum of 30 activations/sec. The **`do`** block will execute for the first time only after the **`after`** expression is satisfied, that is, there is a message in `video_inport`. From that point on, every execution is triggered by the **`atEvent`** condition. It is not known during compile time how long the display operation will take. Consequently, a **`timeout`** block is used. That is, either the display is executed in 1/60sec, or an exception is raised causing control to be passed to the **`expired`** block. In QuAL, exceptions are also raised if the timing constraints of a real time block cannot be met. That is, if video messages arrive but the **`do`** block cannot be executed in 1/30sec, control is passed to the **`miss_deadline`** block.

4 QuAL Automates Generation of Management Instrumentation of Applications

Figure 4.1 illustrates the mechanism used in QuAL to manage applications. Each dashed square delimits a machine. QuAL application A is executing in a workstation that is being managed by a remote *Simple Network Management Protocol* (SNMP)

[stallings93] manager. Application A was instrumented at compile time to generate management information during execution that reflects the performance of its activities. An SNMP agent running on the managed workstation is responsible for collecting the information generated by the QuAL application. For example, the compilation of a connection establishment for port `p` generates code that causes application A to report the event to the SNMP agent. More specifically, the application informs the agent about the QoS requirements of the port, and the time in which the connection was established. Similarly, further instrumentation causes the application to also report when data is sent through a port, or whenever a deadline for executing a real time block is missed.

```
main() { ...
  /* Real time block */
  within (                              /* Timing constraints */
     sporadic; soft;                    /* Mode of execution */
     period sec 30;                     /* Maximum number of activations */
     after(select(video_inport));       /* Initial condition */
     atEvent(select(video_inport));)    /* Condition for other executions */
  do {
     timeout(sec 1/60.0)            /* Timeout block */
       { ... }                      /* Retrieve and display a video frame */
     expired { ... }                /* Discard frame that could not be displayed */
   }
  miss_deadline { ... }             /* Discard frames that could not be displayed */
  until (false);                    /* Termination condition */
  ...}
```

Example 3.1: Handling Video in Real Time

The SNMP agent that collects the management information generated by QuAL applications is part of the language runtime. Such agent maintains the information collected into a QoS MIB, i.e., an information store that contains data related to application QoS. Objects in the QoS MIB are defined using the subset of *Abstract Syntax Notation One* (ASN.1) [stallings93] defined by the SNMP framework. QuAL runtime SNMP agents are also responsible for answering QoS MIB access requests from external management entities that comply with the SNMP protocol. In Figure 4.1, for example, the SNMP manager sends requests to the SNMP agent running on the managed workstation to access information about application A. The manager then analyzes the information and reports on the status of application A by generating messages on a window in the manager station. The manager can inform, for instance, that allocated connections are being poorly used or that the application is failing to meet their deadlines.

The QoS MIB is subdivided into the following groups:

- *application:* information about the QoS demanding activities of QuAL applications running on the system
- *outport:* information about the connections to the QoS demanding outports of the applications running on the system
- *inport:* information about the connections to the QoS demanding inports of the applications running on the system

4.1 Application Group

The application group augments the *Network Service Monitoring MIB* [freed93] (NSM MIB) to include information about application QoS. NSM MIB consists of a table which has one row for each managed application that is currently in execution. The objects in this table provide a description of the managed application, information about its status, and statistics on the health of its non QoS demanding communicating activities. A NSM MIB entry indicates, for example, the name of the managed application, time when the application started, and last time when a connection establishment failed.

There is one entry in the NSM MIB for each QuAL application in execution. Furthermore, each entry is extended to include objects in the application group with information about the QoS demanding activities of applications. By QoS demanding activi-

ties is meant the computations and communications in QuAL that have QoS constraints associated to them. Thus, blocks of instructions that are not part of a QuAL **within** block or communication through ports that do not include the QuAL **realtm**

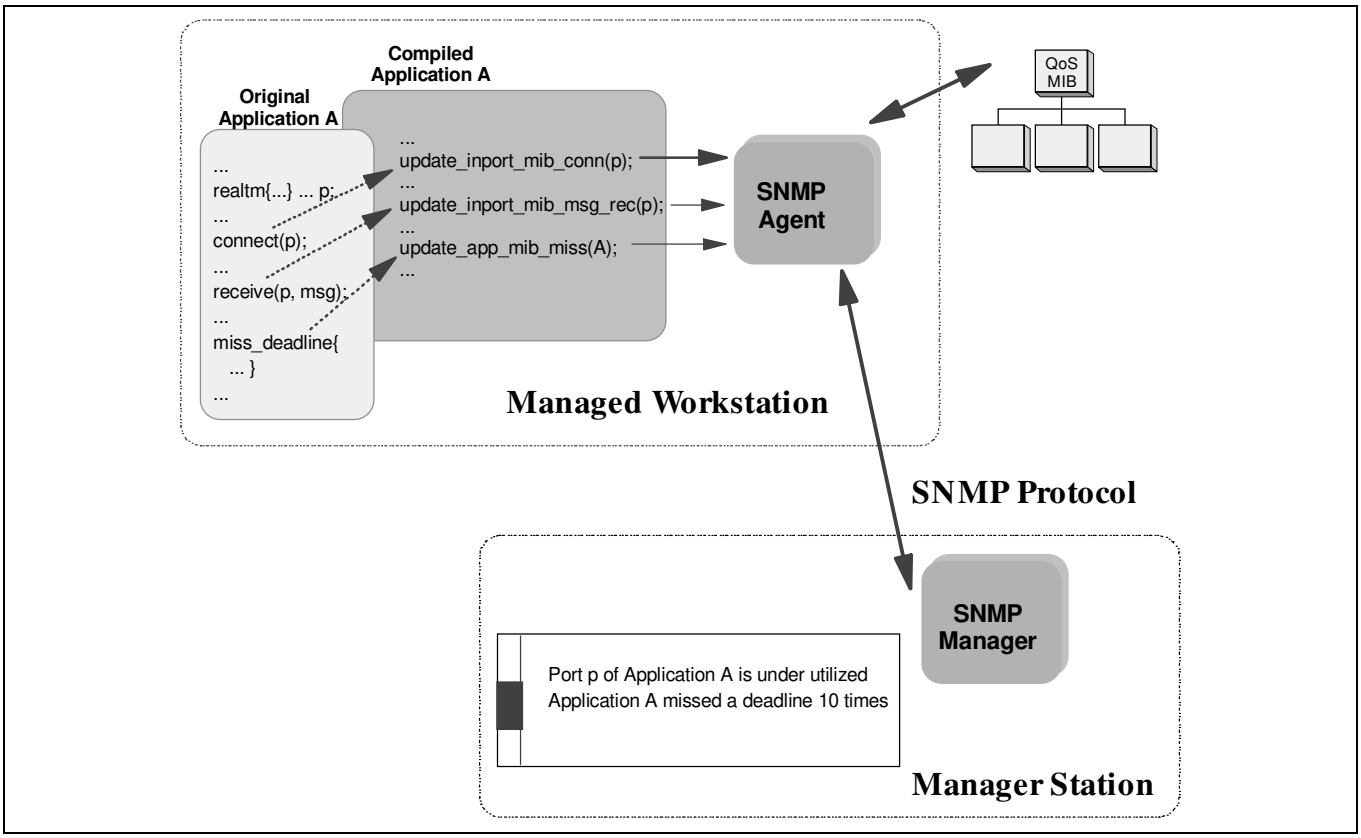


Figure 4.1: Application Management Automation in QuAL

clause are considered to be non QoS demanding.

In the application group, the information related to the QoS demanding computing activities is based on the performance of applications while executing QuAL **within** blocks. Furthermore, the information regarding executions in the soft mode is maintained separated from the information regarding the hard mode, enabling a more accurate analysis of the processing requirements of applications. The following objects provide information about the soft mode of execution. qAAccSft[1] is a counter accumulating the number of times the application started the execution of a **within** block in the soft mode. qALstSft and qALstSftRj are time stamps that indicate the time when the last execution of a **within** block finished and the last time when the application failed to start the execution of such a block because there were not enough processing resources available. qAAccExc is a counter accumulating the total number of exceptions received by the application indicating missed deadlines. qALstExc is a time stamp that indicates the last time when an exception was raised. Similar objects provide information on the performance of applications while executing in the hard mode. In addition, when the managed application is executing in real time, additional objects provide information about the current execution. More specifically, the qACst object is an integer indicating the estimated cost in milliseconds of the **within** block executing, qAPrd is another integer that indicates the maximum number of times that the block can be activated per second, and qAExc is a counter accumulating the number of exceptions raised during the current execution.

[1] The name of QoS MIB objects starts with either qA, qO, or qI depending on whether the object being named belongs to the application, outport, or inport group, respectively. The initial q indicates that they are related to **Q**uAL applications.

Application group objects provide general information about the QoS demanding communication activities in a way very similar to how NSM MIB objects describe the non QoS demanding ones. qAInQoSCnn and qAOutQoSCnn are objects of type gauge that indicate the current number of connections to inports and outports, respectively, that belong to the application being managed. qALstQoSInCnnRj and qALstQoSInCnn are time stamps that indicate the last time when an inport connection request was rejected and the last time when an inport connection was established. Analogously, qALstQoSOutCnnFl and qALstQoSOutCnn indicate the same information for outport connections. The outport and inport group objects provide further information about each connection of the managed applications running on a system.

4.2 Outport Group

The outport group consists of the qOTable, which has one row for each connection to an outport that belongs to a QuAL application described in NSM MIB. A qOTable entry provides information about the identity of a connection, its QoS requirements negotiated with the service provider, its health, and its traffic activity.

A connection is identified by the objects qOLclAdd and qORmtAdd of type IpAddress that indicate a local and a remote IP address, and by the objects qOLclPrt and qORmtPrt of type integer that identify a local and a remote transport-layer port number. The objects qOLclQAdd and qORmtQAdd are of type integer and identify the QuAL level addresses of the local and remote ports connected. It is important to note that a single QuAL level port address can be associated with several connections in a broadcast situation. The object qOLclAIndx of type integer identifies the process that owns the connection and it can be used to identify the entry in NSM MIB that contains more information on the application that is sending data.

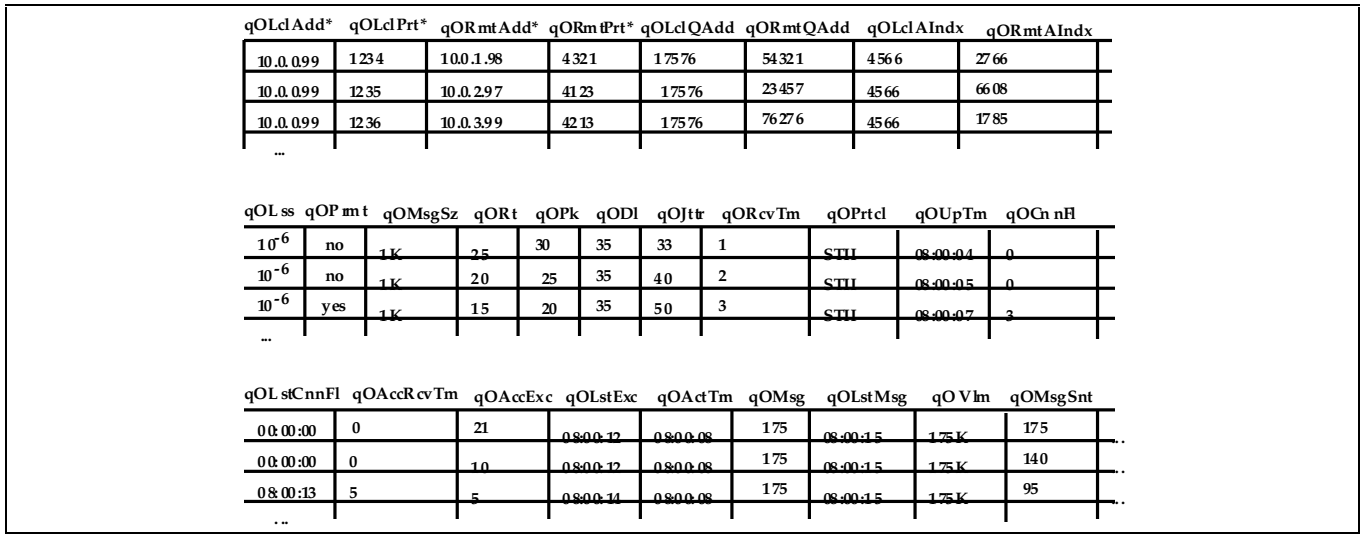
The following objects of type integer describe the QoS demands of the connection being monitored. qOLss indicates the maximum percentage of message loss tolerated. qOMsgSz indicates the maximum size of messages that will be transmitted expressed in bytes. qORt and qOPk indicate the mean and the peak transmission rate expressed in number of messages/sec. qODl and qOJttr indicate the maximum propagation delay and the maximum inter-message delay expressed in milliseconds. qORcvTm indicates the maximum time tolerated for recovery from failures expressed in sec. In addition, qOPrmt has either the value “yes” or “no” indicating whether permutation is tolerated or not, and qOPrtcl of type OBJECT IDENTIFIER identifies the communication protocol being used.

The health of a connection is described by the following objects. qOUpTm, qOLstExc, and qOLstCnnFl are time stamps indicating the time when the connection was established, the last time a connection problem occurred, and the last time an exception was raised due to QoS violations in the communication. qOCnnFl and qOAccExc are counter accumulating the number of connection problems and the number of exceptions raised since connection establishment. Finally, qOAccRcvTm is a counter indicating the total amount of time spent recovering.

The communication management mechanism offered by QuAL allows application defined monitoring functions to sample the information generated by a process, selecting among the data generated the samples that are to be sent. Section 2.2 discussed such mechanism in more details. To capture not only the behavior of the data generation process, but also the behavior of the data sending process, outport group objects maintain information associated with each process separately. The traffic of data generated is characterized by the following objects. qOActTm and qOLstMsg are time stamps indicating when the connection became active and when the last message was generated. qOMsg and qOVlm are counters accumulating the number of messages and the volume of data generated. Similar objects provide information about the traffic of messages sent.

Figure 4.2 depicts three outport table entries that describe connections belonging to the application A. The entries were vertically partitioned in three parts. The first part shows, for each entry, the value of eight objects (from qOLclAdd to qORmtAIndx). The description of each entry is continued in the second part, where the value of eleven objects are shown (from qOLss to qOCnnFl). The third part shows the value of the rest of the objects (from qOLstCnnFl to qOMsgSnt). The value 4566 in the field qOLclAIndx can be used to locate in NSM MIB the entry with more information on application A. Application A is broadcasting information and has three connections for a single QuAL level outport. Outport 17576 is sending video to the inports 54321, 23457, and 76276 that belong to the applications with index 2766 running on machine 10.0.1.98, 6608 running on machine 10.0.2.97, and 1785 running on machine 10.0.3.99, respectively. Each connection involved in the broadcast has different QoS demands. For example, the connection between outport 17576 and 54321 requires a mean transmission rate of 25

messages/sec with a peak rate of 30messages/sec, whereas the connection between outport 17576 and 76276 requires a mean transmission rate of 15 messages/sec with a peak rate of 20messages/sec. The receivers of the broadcast have different amounts of communication resources available for the transmission of video, as illustrated in Figure 2.1. The connections also differ on



qOLclAdd*	qOLclPrt*	qORmtAdd*	qORmtPrt*	qOLclQAdd	qORmtQAdd	qOLclAIndx	qORmtAIndx
10.0.0.99	1234	10.0.1.98	4321	17576	54321	4566	2766
10.0.0.99	1235	10.0.2.97	4123	17576	23457	4566	6608
10.0.0.99	1236	10.0.3.99	4213	17576	76276	4566	1785
...							

qOLss	qOPrmt	qOMsgSz	qORt	qOPk	qODl	qOJttr	qORcvTm	qOPrtcl	qOUpTm	qOCnnFl
10^{-6}	no	1K	25	30	35	33	1	STII	08:00:04	0
10^{-6}	no	1K	20	25	35	40	2	STII	08:00:05	0
10^{-6}	yes	1K	15	20	35	50	3	STII	08:00:07	3
...										

qOLstCnnFl	qOAccRcvTm	qOAccExc	qOLstExc	qOActTm	qOMsg	qOLstMsg	qOVlm	qOMsgSnt	
00:00:00	0	21	08:00:12	08:00:08	175	08:00:15	175K	175	..
00:00:00	0	10	08:00:12	08:00:08	175	08:00:15	175K	140	..
08:00:13	5	5	08:00:14	08:00:08	175	08:00:15	175K	95	..
...									

Figure 4.2: Example of an Outport Table Entry

the time tolerated for recovery from connection failures. The three connections use the ST-II [topolcic90] transport protocol.

The connections were established between 8:00:04am and 8:00:07am, and became active at 08:00:08am. Only the connection with outport 76276 had problems. The connection failed three times since initialization, the last failure was at 08:00:13am, and 6sec were spent in recovering. So, the average recovery time was 2sec (lower than the maximum recovery time indicated in the QoS constraints.) The application that owns port 76276 seems to be located in a troubled location. First, the bandwidth required is very low, and second, the number of failures seems to be high. However, a low quality connection in this case seems to be better than no connection at all, especially for video transmissions. On the other hand, the QoS of the connection between outport 17576 and inport 54321 seems to be the most violated, with a total of 21 exceptions signaled. The last exception for the connection between 17576 and 76276 was at 08:00:14am, right after the connection went down for the last time. So, the exception is probably related to the connection failure. The connection between 17576 and 54321 sends all the data generated, whereas the sampling rate for the connections between 17576 and 23457, and 17576 and 76276 are 80% and 60% , respectively.

4.3 Inport Group

Message loss and permutation during transmission complicate the capture of management information at a receiving end. In order to capture these features, separate statistics are maintained for messages that arrive *out of sequence* and messages that arrive *in sequence*. A message arrives out of sequence when it arrives late, that is, it arrives after the arrival of messages sent after it. A message arrives in sequence when it arrives before all messages that were sent after it. Loss or permutation is detected whenever a message arrives in sequence but before the arrival of messages sent before it. The messages that did not arrive yet are the ones considered to be lost or permutated. Consider, for instance, that the message arrival sequence of an inbound connection is the following, where messages are identified by their index: 1, 2, 5, 7, 4, 8, 3. The flow of messages that arrive in sequence consists of the messages 1, 2, 5, 7, and 8. The flow of messages that arrive out of sequence consists of the message 4 and 3. Loss or permutation are detected for the first time when message 5 arrives, and again when message 7 arrives. The messages 3, 4, and 6 are assumed to be lost. If they ever arrive, they are considered to arrive out of sequence.

Another reason for distinguishing between the flow of messages that arrive in sequence from the one that arrive out of sequence is that the semantics of messages that arrive out of sequence depend on the application being monitored. Some applications (e.g., video transmission) discard messages that arrive out of sequence, whereas other applications (e.g., management applications that perform commutative operations on the contents of the data received) use messages that arrive out of sequence. By partitioning the information, analysis can be performed on each flow in an application dependent manner.

The inport group consists of the qITable, which has one row for each connection to an inport belonging to a QuAL application defined in NSM MIB. A qITable entry contains information about the identity of a connection, its QoS demands, and its health, in the same way a qOTable entry maintains this information for outport connections. However, the traffic behavior is represented by a description of the traffic of messages that arrived in sequence, the traffic of messages that arrived out of sequence, and the traffic of messages processed. Furthermore, a qITable entry also indicates the average propagation delay and jitter of each traffic.

5 Related Work

QuAL comprises work in three distinct areas: distributed computing, characterization and handling of QoS, and real time programming. Regarding distributed computing, QuAL is anchored in the process model and its abstractions can be added as an extension to any process oriented language. References [soares92] and [florissi94] overview current approaches for programming distributed applications, and discuss the advantages in choosing Concert/C as the base language for the first QuAL prototype.

Regarding QoS provision, QuAL provides a general purpose application level abstraction for the negotiation, establishment and management of QoS dependable communications. The model guarantees that connections are opened only between senders and receivers that have matching QoS requirements. Furthermore, the specification of QoS parameters in QuAL is independent of the communication protocol used, and also independent of the nature of the data being transmitted (voice, audio, or data). This approach is general, does not limit the domain of applications, and bridges heterogeneity at transport and session layers. Thus, it has significant advantages in relation to frameworks that are specialized for certain application domains [cohen81] [cole81] [keller93], and to approaches that expose programmers directly to transport layer and session layer service interfaces [topolcic90] [anderson90].

QuAL language constructs for the expression of processing constraints target handling of real time demands in a generic and robust way, that enables graceful recovery from degradation. A survey of real time languages can be found in [halang91] and an analysis of the approach used in QuAL to handle real time features can be found in [florissi94].

6 Conclusions

This paper describes the Quality Assurance Language (QuAL) that eases the management of Quality of Service (QoS) for distributed multimedia computing and communication applications. QuAL provides language level abstractions for the specification, negotiation, and management of communication and processing QoS constraints. The communication QoS constraints include network level constraints (such as maximum propagation delay) and application level ones (such as message delivery rate and synchronization of streams). The processing QoS constraints specify the execution timing demands of activities (such as deadlines to process messages).

QuAL exposes application developers to a single, generic abstraction for QoS specification that is independent of the underlying service providers (transport layer and Operating System). QoS parameters specified are used by the language runtime to allocate communication and processing resources that can best deliver the QoS required.

The language runtime system is responsible for monitoring the delivery of QoS in an application customized manner and to signal QoS violations. Violations are signaled through messages sent by the runtime to application defined ports. QuAL applications control the delivery of QoS by specifying how to treat messages that violate QoS constraints and by supporting dynamic re-negotiation of QoS demands. In this manner, QuAL promotes a fast, customized, and graceful recovery from violations.

QuAL applications can be automatically instrumented at compile time to generate Management Information Bases (MIBs) during execution. MIBs contain statistics on the performance of applications running on a system. The runtime includes a man-

agement agent that answers access requests to the generated MIBs according to the SNMP standard. As a consequence, external management entities can share QoS management with applications by accessing the generated MIBs.

QuAL abstractions can be incorporated as an extension to any process oriented language and are compatible with existing inter process communication abstractions, thus easing QoS specification and management without incurring a high learning overhead. QuAL provides generic abstractions for the specification of QoS demands that do not limit the application domain.

The first QuAL prototype being implemented extends Concert/C and uses ST-II transport protocol services for QoS demanding transmissions.

References

[anderson90] Anderson, D. P., Tzou, S., Wahbe, R., Govindan, R., and Andrews, M., "Support for Continuous Media in the DASH System," in *Tenth International Conference on Distributed Computing Systems,* Paris, 1990.

[auerbach92] Auerbach, J., "Concert/C Specification," Tech. Rep., IBM T. J. Watson Research Center, Yorktown Heights, NY, November 1992.

[cohen81] Cohen, D., "A Network Voice Protocol NVP-II," Tech. Rep., USC/Information Sciences Institute, April 1981.

[cole81] Cole, E., "PVP - A Packet Video Protocol," Tech. Rep., W-Note 28, USC/Information Sciences Institute, August 1981.

[comer91] Comer, D. E., Stevens, D. L., *Internetworking with TCP/IP Volume 1* Prentice Hall, NJ, 1991.

[florissi94] Florissi, P. G. S., "QuAL: Quality Assurance Language (thesis proposal)", Tech. Rep. CUCS-007-94, Columbia University, March 1994.

[freed93] Freed, N., and Kille, S., "Network Service Monitoring MIB", Internet Draft, November, 1993.

[halang91] Halang, W. A. and Stoyenko, A. D., *Constructing Predictable Real Time Systems*. Boston/Dordrecht/London: Kluwer Academic Publishers, 1991.

[keller93] Keller, R. and Effelsberg, W., "MCAM: An Application Layer Protocol for Movie Control, Access, and Management," in *First ACM International Conference on Multimedia,* Anaheim, 1993.

[kernighan88] Kernighan, B. W. and Ritchie, D. M., *The C Programming Language*, second ed. Prentice Hall, NJ, 1988.

[soares92] Soares, P. G., "On Remote Procedure Call," in *Second CASCON International Conference*, Toronto, Canada, November 1992.

[stallings93] Stallings, W., *SNMP, SNMPv2, and CMIP*. Addison Wesley, 1993.

[topolcic90] Topolcic, C. "Internet Stream Protocol", Requests for Comments 1190, October, 1990.