An End to End Price-Based QoS Control Component Using Reflective Java

Jérôme Tassel, {jtassel@jungle.bt.co.uk}
Bob Briscoe, {rbriscoe@jungle.bt.co.uk}
Alan Smith, {asmith@jungle.bt.co.uk}
BT Advanced Research & Technology, UK

Abstract

The main objective of the model we describe in this paper is to allow easy, flexible addition of quality of service (QoS) control to Java Internet applications. In this work the QoS is expressed in terms of network and host resources, the network QoS being controlled with RSVP. Flexibility is provided by a prototype product from the ANSA research consortium; Reflective Java which uses the Meta Object Protocol (MOP) to separate functional requirements (what the application does) from non-functional requirements (how it does it). This protocol permits the design and implementation of a generic QoS control element which can be added to an application for which QoS control is required. Alternatively, an existing application with rudimentary QoS control can be modified to use a set of QoS control classes designed by a specialist intended to reconcile competition for QoS between applications. The QoS control element we have designed also has scope for QoS adaptation, moving decisions on the introduction of QoS control from build-time to run-time when best-effort degrades below a useful point. Charging is also considered in this work.

Acknowledgements

This work was funded by BT as part of an industrial placement agreement for Jérôme Tassel from the MSc in Distributed Systems course at the University of Kent at Canterbury (UKC). Bob Briscoe and Alan Smith are researchers in the Distributed Systems Group in BT's research labs. The authors would like to thank Steve Rudkin, Peter Bagnall and Andrew Grace at BT, Zhixue Wu working on the ANSA project and Andy King from UKC for reviewing earlier versions of this paper and their valuable advice.

1. Introduction

In this paper, we describe the design of a flexible, easy-to-add adaptive quality of service (QoS) architecture for multimedia, Java-based [19,20], Internet applications. Control is provided for QoS properties that we identified as crucial in specifying QoS for real-time applications. This set of properties can be divided into two groups :

- User requirements (prioritisation, quality perception, budget)
- Mechanisms (Network QoS, RSVP in this case, and Host QoS)

Flexibility and adaptability is provided by a prototype product from the ANSA research consortium; Reflective Java which uses a Meta Object Protocol [12, 13, 14, 15] (MOP) to separate functional requirements (what the application does) from non-functional requirements (how it does it) [15]. The generic QoS control element we have defined can be added to an application for which QoS control was not originally thought of or to replace a deficient QoS control or to provide QoS control as a result of adaptation to network conditions [28]. Applications which require such a control are emerging Internet collaborative tools with a multimedia interface using audio and video streaming facilities. The QoS control element we have built also has scope for

QoS adaptation and charging. We believe this model provides a flexible way to control the usage of the resources available to the user down to a fine level of granularity. A QoS control interface is available to the user giving him complete flexibility over the sharing of his resources among his application sessions, media sessions and streams. This interface might exist as an operating system component separately from any applications that might use it.

The first half of the document describes RSVP (Internet ReSerVation Protocol) and Reflective Java, focusing on the points of interest for this project. The second describes in depth the architecture we have designed and some of the implementation results and experiences we have had so far.

2. QoS Control on the Internet with RSVP

RSVP is used in this work as the network QoS control mechanism. The design allows for other mechanisms to be used instead, or as well. Effectively RSVP allows control of the quality of service of data streams over the Internet [2,4]. Internet applications are changing from simple remote procedure call (RPC) type text based point to point applications to real time, multi-user, multimedia applications [1]. The original design of the TCP/IP suite only provided support for a best effort delivery scheme, ideal for applications such as ftp, World-Wide-Web, Telnet, and e-mail but is very deficient for real-time delivery of data. The efforts of the Internet Engineering Task Force (IETF) are now focused on developing a new Internet architecture which can provide Integrated Services on the Internet (the ISA). This will allow a wide range of QoS to co-exist, some with hard or soft real-time delivery constraints and some with more elastic timing constraints. The main weakness of the IP protocol for guaranteed delivery on the Internet is the variable latency of packet processing time within the routers along the data path, due to queuing delays which increase the jitter of the data stream, thus jeopardising real time delivery of data. The objective of RSVP, as part of the ISA, is to provide some control over the routing queuing delays and therefore the QoS of data streams as they pass through the routers.

RSVP attempts to reserve bandwidth for real time streams across the routers by setting packet priorities. In the case of guaranteed delivery of streams, RSVP allows provision of a guaranteed set of resources for a data stream based on an IP address (which can correspond to a unicast or multicast address) and a port number. We now describe the major features of the RSVP protocol which are of interest for our work.

2.1 Receiver Initiated Reservation and Message Processing

An Internet communication application can be divided into two parts, the sending and the receiving part(s). The sending application acts as a source of data for the receiving application(s); there might be multiple receiving applications in a multicast communication environment. Multiple senders can be treated as separate sources and collected or synchronised independently if necessary. In order to accommodate heterogeneous receivers on the Internet, receiver initiated reservation has been chosen in RSVP. The receiver chooses the reservation it wishes to make for a given source data stream. It uses information disseminated by the sending application about the data it is producing to decide what reservation is required. Admission and policy control is

the subject of current research work [5] to provide administrative control of bandwidth sharing and to make sure that streams keep to their agreed traffic properties.
Two types of messages are defined in RSVP, namely 'PATH' and 'RESV' messages. PATH messages are used by the sending applications to disseminate information about their data streams, and RESV messages are used to establish the reservations by the receivers that get the PATH messages.

2.2 QoS Attributes

RSVP does not know about QoS attributes but just conveys them transparently [5]. There are two main types of service available for RSVP: a guaranteed service and a controlled load service In our work we use the guaranteed service. The reservation attributes in the PATH messages define the sending application requirements in terms of expected bandwidth usage. The reservation attributes in the RESV messages represent the required bandwidth reservation for the receiving applications. The attributes used for the control of the bandwidth are the generated traffic rate and peak rate, the token bucket rate, minimum policed unit (defining a minimum size for packets to be policed) and the maximum packet size (for the underlying network).

2.3 The RSVP API (RAPI) [3]

An application programming interface (API) is available to interact with the RSVP daemon on hosts which implement RSVP. Operations are available to *join* a session (an RSVP session), register as a *sender* and send PATH messages, *reserve* bandwidth, modify the reserved bandwidth and *release* a reservation.

Another important aspect of RSVP is that of *event messages*. These messages are asynchronous (but synchronous results are also provided by the API calls, which provides some basic error checking such as parameter checking). Errors and modifications might occur along the data path and messages are sent back through the RAPI in order to notify the applications using RSVP for them to take appropriate actions. Messages might indicate an arrival of PATH messages, RESV messages, path errors, reservation confirmation or not. These upcalls are associated with an application level method which is called on receipt of such events.

2.4 QoS Adaptation in RSVP

The PATH and RESV messages we describe above are sent periodically (the RSVP daemon handles this) so that if the properties of the sending application stream are modified or the route through the network changes the receiver can then adapt its reservation. An RAPI call is available to modify the attributes of the reservation.

Aside from RSVP, Reflective Java is the other key element of an architecture which needs to be understood and is therefore described in the next section.

3. Reflective Java

Reflective Java is a prototype product from the ANSA research consortium [16, 17] providing the facility to separate the functional aspects of an application from the non functional ones [11], this is done using the Meta Object Protocol (MOP) [12, 13, 14, 15]. This separates the roles of application developers, one focuses on what the application does and another on how it does it. The term MOP is unfortunate - it is not a communications protocol, more a design pattern allowing this separation. Examples of non-functional requirements are the well know transparency requirements of the distributed systems world: replication, concurrency, failure transparency, some of which have also been worked on in a very similar development called MetaJava [18]. This scheme means that some requirements can be added late in the life cycle of an application, even if they had not been thought about originally.

The following diagram illustrates how Reflective Java implements the Meta Object Protocol at run-time:

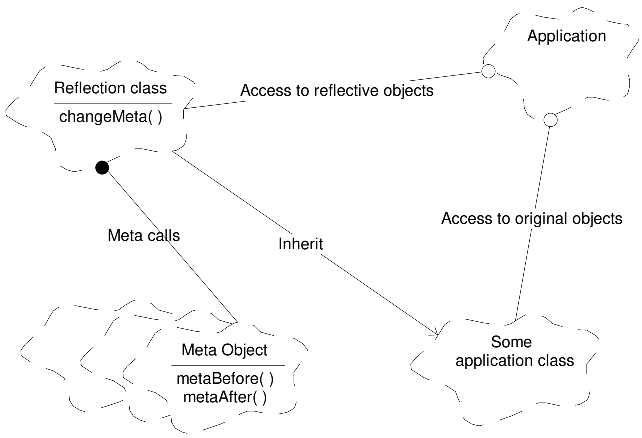

Fig. 1, The Reflective Java run-time model

The *meta-objects* implement some non functional requirements and are independent from any application which uses them so that they can be used by many applications. The meta-object developer can focus on designing meta-objects providing some functionality and those objects are later bound to application code. The meta-object defines two methods: *metaBefore* and *metaAfter* which implement some behaviour to be executed before and after the normal invocation of an application method. For example a locking meta-object providing concurrency control would provide some code to set a lock in the *metaBefore* method and unlock it in the *metaAfter* method.

The *Reflection class* is used to bind applications to meta-objects (bind the meta methods to the application class methods). A simple example is a locking meta-object and an account application class which uses this meta-object to provide concurrency control on the accounts. So for example a deposit method would be bound to the locking service provided by the meta-object. The original invocation of the deposit method would then become :

Application call deposit() → Reflective class calls metaBefore → Reflective class calls original deposit method through inheritance → Reflective class calls metaAfter

When the service provided by the meta-object is needed by the application class, the reflection class is used in the application code instead of the application class it inherits from.

As shown on the above diagram another important feature of Reflective Java is that the meta-object can be *dynamically* replaced by another one, implementing a new version of a service or a new service being more adapted to a new environment. This overcomes the drawback of having to manually update the code of the application to be able to make use of the reflection class as there is no need to further modify the application when the services from another meta-object are required.

In the build process, it is the reflective pre-processor which creates the reflective class from a binding specification file, which dictates which application method class should use the services of the meta-object. The following diagram illustrates the build process of Reflective Java.

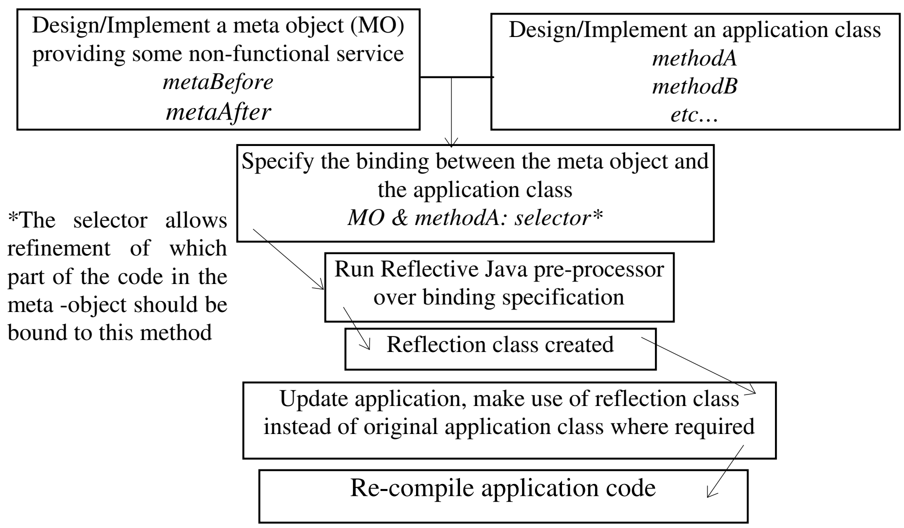

Fig. 2, Steps in using Reflective Java

This implementation of the reflection technique in Java is made possible by the dynamic class loading feature of Java. The possibility to download Java classes over the network make this scheme very flexible and place Reflective Java ahead of some other implementations of Reflection described in [12,13,14,15].

In the work presented in this paper we use the capabilities of Reflective Java to separate the QoS control of Internet applications from the mechanisms of those applications; sending data.

4. QoS Architecture

The aim of our architecture is to be able to provide a generic (adaptive) QoS control element, which allows the best use, in terms of user-perceived value, of the host and network resources [26]. However, it is of utmost importance that this element can be added to existing Internet communication applications written in Java by an independent developer. This implies it is not necessary to know the detailed internal mechanisms of the application and the integration could be reduced to a mechanical task in the future. Source code access is required, however.

Currently, for applications without any QoS control [29], as the network or host becomes congested the quality of the communication decreases due to the unreliable nature of the Internet best-effort data transfer protocols and the lack of resource control in most current operating systems. Today's Internet communication applications, written in Java or any other language, provide functionality both for sending the data and controlling the quality of the data sent. Typical examples are the vic and vat video/audio conference tools [6, 7] which provide a control panel for the properties of the multimedia streams. These applications do not make use of any network or host resource reservation protocols, although a recent prototype of vic now supports RSVP. Some others such as Vosaic or QuickTime Conferencing provide some form of QoS adaptation to network congestion in a best effort delivery environment but all in a different and non compatible manner and some are becoming increasingly monolithic and complex [31] as they try to introduce QoS control.

Moreover as QoS control is provided on a per-application basis the sharing of the host resources available to the user between users will not be optimal. Further, QoS adaptation can only be done on a per application basis, which again will not be optimal. Another downside of mixing the code of the application task and the stream control is that it is then more difficult to understand the code of the application and to modify just the stream control (non-functional) part or just the application task (functional) part.

Our component using reflection helps remove the limitations of the current ways of integrating QoS control with real-time multimedia streaming [8] applications by providing guaranteed network resources over the Internet as well as sharing efficiently the network and host resources among the users and giving a flexible control to the user over his own resources. Our model provides a clear distinction between the application and the QoS control. We also believe that adaptation can be more efficient as more host-wide information is available to the adaptation mechanism.

The diagram on the following page is a representation of the model we have designed. There are three major elements in this model:

1. The original application (which models typical Java Internet applications)
2. The QoS control architecture (which includes components to control host and network resources in order to share them between application streams)
3. The reflective architecture (which binds the two previous parts together)

In the model, different components control or give quotes for a unique type of resources (host or network). The QoS manager acts as a conductor of all of them. Resource control and quoting have been separated to permit them to evolve independently. The model we designed is for receivers but the same techniques could be adapted to senders. However we have only implemented the former. The following section describes the design choices for the model we created.

4.1 Original Application (1)

The aim of this part of the model is to represent a typical Internet application that uses some communication class to interact with the network. The original application classes altogether provide both the functional and the non-functional requirements of the application. The original application [29] (that is, without our QoS control) would only use the communication class and not the reflection class. The reflection class is the reflective element that allows us to include QoS control in the application with minimal modifications to the code.

At the application level, an important issue related to the reflective architecture was to decide which class in the application to make reflective. We were faced with two options:

- to make the communication class used by the application reflective
- to make a more high level application class reflective.

In order to make our solution reusable and portable we decided to take the first solution In the case of Java this meant making the Socket class reflective. This means that when the application code is to be modified there is no requirement to know the structure of the program but just to know where communication objects are instantiated and to replace the original communication class by the new reflective class we have designed. It also means that any application using the Java Socket class can re-use the work we have produced (in the other case a new binding specification would have been required for each new application).

This, of course, means that the application needs to be modified, which is even more constraining as it is done *manually* (however, a modified class loader could be used to avoid this). The main issue during this process is to decide *which communication objects require QoS control* as not all streams need timeliness; some are signalling streams for which a best effort delivery scheme is very well suited. We identified the following options:

- *Make all communication objects (Sockets) reflective but test the port number within the meta object class and set reservation only on some well known ports.* A port number is an obvious selection criterion as it is the only element of information guaranteed to be available when any general communication channel is created. This information can be passed to the meta-object which will then decide on the path to follow; interacting with the QoS manager or not. This has got some run time cost as all communication objects would be reflective while only some would really need to be. Also some applications use random ports in which case this scheme could not be used. However , the main advantage of this method is that it requires only a few easy changes to the application code.

- *Make all communication objects reflective but include heuristic logic in the meta-object to decide at run-time which calls do or do not require real-time control.* This has the performance cost drawback of the previous solution but avoids the need for port number conventions. It would be the only solution if the need for

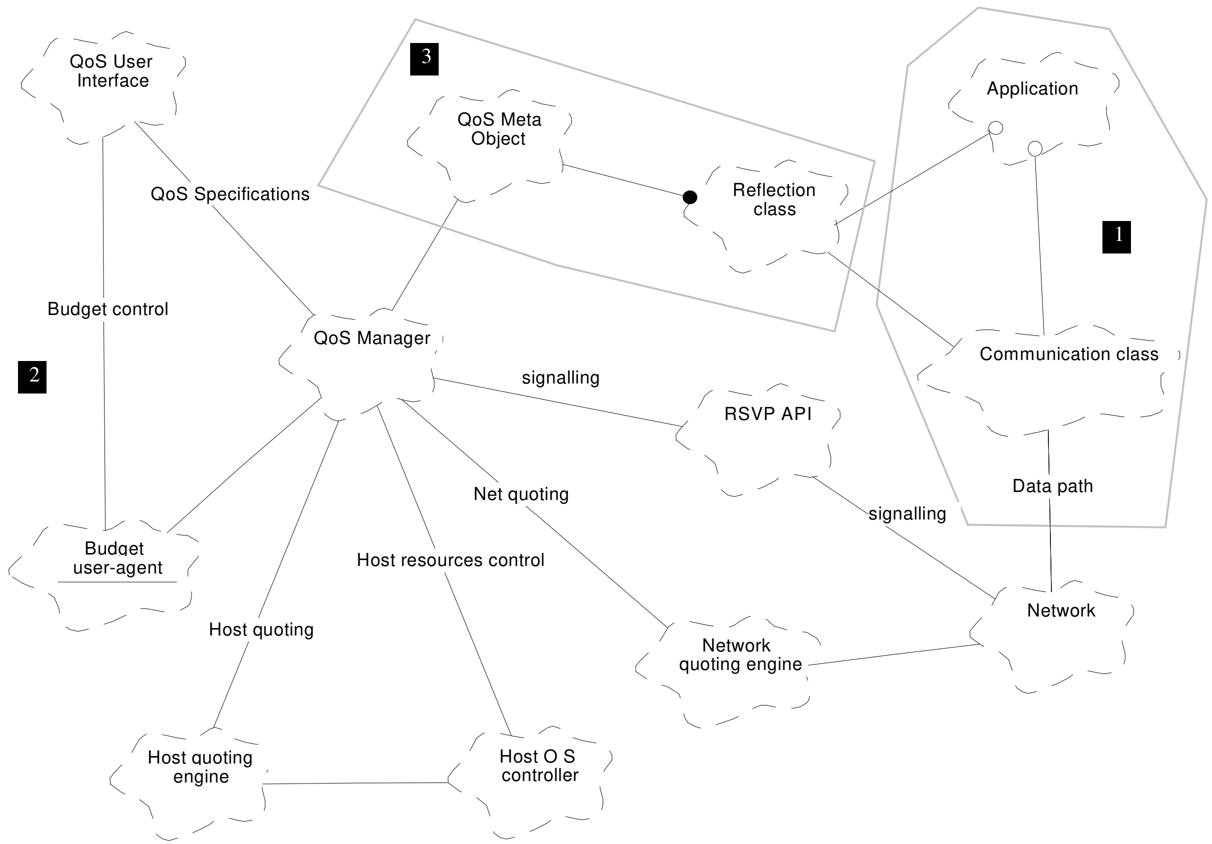

Fig. 3 , Reflective model for (adaptive) QoS control

real-time control for each class varied depending on run-time conditions. Another drawback is that it is not clear to us how one could define the heuristics.

- *Make all communication objects reflective as the channels which do not require real-time control will not receive PATH messages and therefore reservations will never be established*. This again has the same drawbacks as the previous solution and worse, RSVP sessions will be created which will never be used, but the solution again requires no conventions concerning port numbers.
- *Give control to the user to state whether or not a stream is to be timely*. This would present the user with a stream control panel. This of course is a too finely grained solution as the user should not be aware of the stream notion and will probably not know when a stream is to be a data stream which needs to be timely or a signalling stream which does not require real time control.
- *Go manually through the code and decide according to the context if the stream requires QoS control or not.* This means that the code of the application must be clear enough for the system integrator to decide if the data transmitted on a stream is real-time or not. In our case this is the solution we have chosen due the prototype nature of the target applications we use. However, this approach would not be appropriate if the need for real-time control varied with run-time conditions.

4.2 The Reflective Architecture (3)

The reflective part of the model allows us to link together the application services and the QoS control architecture we have designed. Two classes make up this part of the model, the *QoS meta-object* implements the QoS control which in our case is delegated further to the QoS manager. As a result the role of the meta-object is only to call the QoS manager. The call to the QoS manager is in the metaAfter method so that the creation of the socket is not held up, the metaBefore method is used to release reservations on destruction of a socket. Furthermore making a call to the QoS manager (and not directly to the RSVP API) does not tie down our Reflective model to the use of RSVP.

Notice that thanks to the dynamic facilities of Reflective Java the meta-object could be modified at run-time. For example, when the environment changes (from a fixed to a mobile network, or from a network using RSVP for bandwidth reservation to another using another reservation protocol [9, 10, 25]) a method is provided to switch to a new meta-object in response to such arbitrary events. This could be used to call a different QoS manager more appropriate for the new environment or having some extra-functionality. The QoS meta-object is generic, it is not designed with any host applications in mind. We have added a call to the QoS manager (described in the following section) in the metaBefore method and in the metaAfter call. The call in the metaBefore method is to set-up the required QoS for the newly created stream (the key used to identify each stream is “user name + application name + stream IP address + port number”) and the call in the metaAfter call is to release reservations.
The second class is the *reflection class,* which is a refined version of the original application communication class and binds together the meta-object methods to the appropriate methods of the original application class. The reflection class is specific to

the application class we decided to make reflective; the Socket class. It was created originally by the Reflective Java pre-processor but we had to make manual changes to it due to some limitations of ANSA's Reflective Java prototype (no support for a Reflective constructor or make classes from the Java APIs reflective).

4.3 The QoS Control Architecture (2)

The aim of the QoS control architecture is to manage resources on integrated service networks (focusing on Internet and RSVP). The resources our architecture controls are network resources (in terms of bandwidth), host resources (in terms of memory, CPU usage and hard disk space) and user resources (budget). Fig 3 illustrates the different objects of our architecture. The model we have designed is host oriented; the role of the QoS manager is to co-ordinate the services of the other QoS related objects in order to manage resources on behalf of a host. Therefore on a network using our model, each host would hold a copy of the full model described in Fig 3. Each user on the host could be running multiple applications using many streams, some of which would require QoS control. There would therefore be many instantiations of the QoS meta-object class and reflection class, one for each of the streams which required QoS control. But each user has access to only one QoS user interface, which integrates control over all the streams he owns. Also there is a single user budget agent per user.
Implementation issues regarding communications between objects running on different virtual machines (the user applications and the user QoS control interface for example) have been solved by using a product internal to BT: the JavaShell. With this, multiple applications can run on the same virtual machine and communicate with each other (in our case the QoS control objects, which are unique for a host, are static and can be accessed by all other objects). This avoids the need to differentiate between real host resources and those allocated to each virtual machine, a problem likely to disappear as the virtual machine becomes integrated with the operating system [21].

The architecture is not exhaustive in terms of how resources can be managed but objects could easily be added to control resources differently (e.g. administrative QoS management, or adding billing facilities). The next paragraphs discuss the different elements of this architecture.

Managing network resources

In order to control network resources for the application streams, the QoS requirements of those streams must be defined. As our QoS model can be added to an application which does not deal with QoS control (or very little), those requirements must be emanating from another source. As we use RSVP to manage the network QoS, we use the information provided in the TSpec element of the PATH messages to decide on what QoS properties a stream might require. Those requirements are then balanced by the QoS manager according to the user choice of priorities between streams (see below) and the resources available (on the network and the host) [22]. Juggling of user priorities is done in financial terms. The costs of network resources are therefore needed; these are provided by the network quoting object. The assumption is that, in the future, all QoS will have a price in order to put a brake on profligate users.

It is the QoS manager which issues demands for bandwidth reservation via the RSVP API; it holds the set of users, their applications, their streams and the network QoS for these streams. Notice that, because our implementation is in the Java language we had to implement a RSVP API in Java using Java native (JNI) calls [24].

Streams that are controlled can remain in a best effort mode until they require reservation, as the network or the host becomes congested. Another approach would be to provide them with a set of default resources as defined by the TSpec element of the RSVP PATH messages. The user by changing relative priorities between streams will then trigger modification of reservations.

A QoS monitoring object could signal the QoS manager when a stream is not performing well enough and then the QoS manager would provide some reservation for this stream. In our current implementation, the user can trigger reservation by using the QoS control interface we have designed, but integration of our automatic QoS monitor [23] is outside the scope of this work.

Also we decided to implement an application API for some of the QoS manager services so that QoS controlled directly from an application can also be taken into consideration. The QoS manager user interface shown in Fig 5 will be enhanced to make the distinction between application and user control of stream priorities.

Managing host resources

The role of the host controller object is to provide some admission control facilities regarding the usage of the host resources by the application streams on behalf of the users. We designed and implemented a host resources control object, which holds the amount of available resources on the host and registers their use. We also implemented a host quoting engine which returns a price from a specification of host resources. The host QoS requirements for an application stream is deduced from the required network QoS for the same stream. This might seem simplistic but it prevents overloading of the host. A more complex version of this object could be implemented but the services required would still remain the same as we have implemented. The QoS manager interacts with the host controller to set and release reservations and the user budget agent uses the services of the host quoting engine to authorise reservations.

Managing user priorities

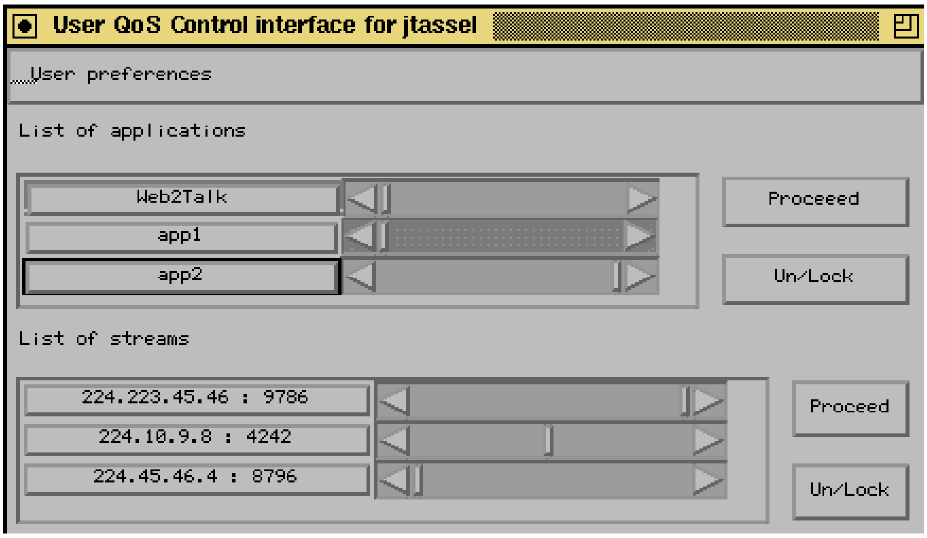

Fig. 4, The user QoS control interface
Note that the identification of streams by 'address:port' is merely a prototyping convenience - textual stream ids such as 'audio' or 'video' could have been generated relatively easily

The figure above shows the user QoS control panel we have implemented. This interface is available for each user to define the priorities between applications and streams within applications. The user can select an application in order to get the list of streams created by the application (the Sockets instantiated from the reflective class of the reflective part of the model). Then he can define which stream should have more priority over the host and network resources by positioning the scroll bars accordingly. These priorities have an impact on the distribution of the user budget over the current streams he is using. For users not wanting to have such a finely grained level of control on the streams, we have added a user preference which removes the list of streams from the control panel and only provides control over the list of applications. The resources are then distributed over the applications in proportion to the resources the PATH messages said were needed. Development of the GUI to express distinction between application requirements on priorities and user override is ongoing.

These priorities are then sent to the QoS manager which increases or decreases the resources reserved for the related streams in conjunction with the quoting engines, the resource controllers and the user budget agent (network and host). Streams being defined with a null priority are left in a best effort communication mode. This control could be part of a host control panel (similar to the one in MS Windows) and the QoS architecture could be part of the host operating system to provide a QoS management service to the user.

Managing charging issues

We have introduced a user budget agent in our design, which controls the budget of a user according to some rules predefined by him. The rules include the time constraints on reservation establishment and restrictions on maximum spending per stream and per application . When a new set of resources needs to be reserved on behalf of a stream, the QoS manager decides which attributes to reserve and mediates between the quoting engines and the user budget agent. The QoS manager requests a quote for the amount of resources it wants to reserve and sends this quote to the user budget agent which then can authorise or refuse it, according to the user spending rules.

It was a deliberate decision to normalise all QoS comparison into units of relative financial cost rate. This is the most convenient common unit both for the programmer and in terms of user understanding.

5. Conclusions and Further Work

One of the main achievements of our work is a component with a clear separation between the transmission mechanisms of an Internet application and the policies and mechanisms to control its QoS. This makes it very flexible in terms of later upgrades, configurability and adaptability. An Internet programmer can focus on the mechanisms of the application and a QoS specialist can focus on providing a QoS control element, thus increasing productivity. This separation also means that existing applications as well as forthcoming applications and the QoS control model we designed can easily be integrated. Re-usability is also an important achievement in our model as meta-objects are application independent. Our model also provides scope for adaptation to a changing environment. We only designed it for Internet (RSVP) but another one could be designed for a different environment (e.g. ATM, mobile networks) We have provided the run-time switching mechanism for this.

In our model, the presentation of QoS control to the user proved to be difficult. Our work has proved to us that relative priorities between cost rates are the most useful concept in this respect. Reflective Java proved to be relatively immature. Readers interested in a constructive critique of Reflective Java for QoS control can refer to [30] where a more in depth description of this work is also available.

The implementation of the model is still continuing as at this first stage we concentrated on getting the overall architecture working. We aim to provide better control in a multicast communication environment, provide more event-based reactions by the QoS manager and finally we would like to develop a reusable RSVP API in Java because the one we implemented for this work is very specific to our QoS manager. Another area of possible extension is in the run-time heuristic decision over whether to make a socket reflective (and thus give it timeliness control) and finally the QoS manager could be further separated to avoid over-centralisation.

6. References

[1] RFC 1633, Integrated Services in the Internet architecture: an overview. R. Braden, D.Clark, S.Shenker. June 1994. (33 pages) http://www.isi.edu/div7/rsvp/ietfpub.html

[2] RSVP: A new resource reservation protocol. Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker and Daniel Zappala. IEEE Network. September 1993. (9 pages) http://www.isi.edu/div7/rsvp/ietfpub.html

[3] RAPI -- RSVP Application Programming interface -- version 5. IETF RSVP Working group. Draft RFC, Expires November 1997. R. Braden, D. Hoffman. http://www.isi.edu/div7/rsvp/ietfpub.html

[4] Resource ReSerVation Protocol (RSVP) - Version 1 functional specifications. Internet draft. R. Braden, L.Zhang, S.Berson, S. Herzog, S.Janin. November 1996. (82 pages) http://www.isi.edu/div7/rsvp/ietfpub.html

[5] The use of RSVP with IETF Integrated Services. J. Wroclawski MIT LCS. Internet draft. Expired 3/97 http://www.isi.edu/div7/rsvp/ietfpub.html

[6] vic: a flexible framework for packet video, McCanne S and Jacobsen V, Proc of ACM Multimedia '95, Nov 1995.

[7] First IETF Internet Audiocast, Casner S and Deering S, Computer Communications Review 22, July 1992

[8] Specifying QoS Multimedia Communications within Distributed Programming Environments. Daniel G. Waddington, Geoff Coulson and David Hutchison. Computing Department Lancaster university. Proceedings of the 3rd International COST237 Workshop. Volume 1185, pp. 10-4-130. November 1996(20 pages)

[9] Supporting adaptive services in a heterogeneous mobile environment. Nigel Davies, Gordon S. Blair, Keith Cheverst and Adrian Friday. Distributed Multimedia Research Group at Lancaster University, ADAPT project. In proceedings of the 1st workshop on mobile computing systems and applications, Santa Cruz, CA, December 8-9, 1994. (5 pages)

[10] Quality of Service control for adaptive distributed multimedia applications using Esterel. Andrew Grace, Alan Smith. BT Laboratories. (8 pages) Second International Workshop on High Performance Protocol Architectures, Sydney, 11-12 December 1995

[11] A QoS Configuration System for Distributed Applications. Alan Smith, Andrew Grace, BT Laboratories. 5th IFIP International Workshop on Quality of Service (IWQOS'97) (4 pages) http://comet.ctr.columbia.edu/iwqos97/

[12] Concepts and experiments in computational reflection. Pattie Maes. AI-Lab, Vrije Universiteit Brussel. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) Proceedings, 4-8 October 1987. (8 pages)

[13] Computational Reflection in Class based Object Oriented Languages. Jacques Ferber, LAFORIA. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) Proceedings, 1-6 October1989. (8 pages)

[14] Workshop: Reflection and Metalevel Architectures in Object-Oriented Programming. Organiser: Mamdouh H. Ibrahim. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA/ECOOP) Report, 21-25 October 1990. (5 pages)

[15] Atomic Data Types. T.J. Stroud and Z. Wu. Department of Computing Science, University of Newcastle upon Tyne, UK.

[16] Design of Reflective Java. ANSA draft, Zhixue Wue and Scarlet Schwiderski, 23rd December 1996. APM.1818.00.06 (22 pages) Restricted to ANSA sponsors http://www.ansa.co.uk/Research/ReflectiveJava.htm

[17] Design and Implementation of a persistence service for Java. ANSA, Scarlet Schwiderski, 27th, January 1997. APM.1940.02 Restricted to ANSA sponsors http://www.ansa.co.uk/Research/ReflectiveJava.htm

[18] MetaJava: An efficient run-time meta architecture for Java. Jurgen Kleinoder, Michael Gölm. Friedrich-Alexander-University, Erlangen-Nurnberg, Computer Science Department, Operating Systems-IMMD IV, Germany, June 1996. TR-I4-96-03. http://www4.informatik.uni-erlangen.de/Projects/PM/Java/

[19] The Java language Environment, A white paper. James Gosling, Henry McGilton. Sun Microsystems Inc. October 1995. http://java.sun.com/docs/index.html

[20] The Java Platform, A white paper. Douglas Kramer. Sun Microsystems Inc. May 1996. http://java.sun.com/docs/index.html

[21] Secure computing with Java: Now and future. Sun Microsystems Inc, 4th June 1997. http://java.sun.com/docs/index.html

[22] QoS Mapping home page and set of supporting slides. Daniel G Waddington, BT-URI at Lancasteruniversity, End-system QoS in Multi-service Networks project. 6 December 1995 http://www.comp.lancs.ac.uk/computing/users/dan/uri/mapper/index.html

[23] Support Components for Quality of Service in Distributed Environments: Monitoring Service, D.A.Reed and K.J.Turner, Stirling University, UK, 5th IFIP International Workshop on Quality of Service (IWQOS'97) (4 pages) http://comet.ctr.columbia.edu/iwqos97/

[24] The Java native programming interface. Sun Microsystems Inc http://java.sun.com/docs/books/tutorial/native1.1/implementing/index.html

[25] Campbell, A.T., "QOS-Aware Middleware for Mobile Multimedia Networking" , Multimedia Tools and Applications, Special Issue on Multimedia Information Systems, 1997, (to appear) http://comet.ctr.columbia.edu/~campbell/andrew/publications/publications.html

[26] Aurrecoechea, C., Campbell, A.T. and L. Hauw, "A Survey of QoS Architectures", Multimedia Systems Journal , Special Issue on QoS Architecture, 1997, (to appear) http://comet.ctr.columbia.edu/~campbell/andrew/publications/publications.html

[28] Campbell, A.T., Coulson G., and D. Hutchison, "Supporting Adaptive Flows in Quality of Service Architecture", Multimedia Systems Journal, Special Issue on QoS Architecture, 1997, (to appear) http://comet.ctr.columbia.edu/~campbell/andrew/publications/publications.html

[29] Web2Talk, Internet telephony application. Roger Klein, Carsten Schulz-Key, Stephane Chatre http://www.CS.ORST.EDU/~kleinro/Web2Talk

[30] QoS Adaptation using Reflective Java Jérôme Tassel. M.Sc project dissertation, University of Kent at Canterbury UK. September 1997. To appear.

[31] Implementation of stream module in a Distributed Computing Environment. Rainer Aschwanden, BT Labs. September 1996.