

Controlling Quality-of-Service in a Distributed Video Application by an Adaptive Middleware Framework

David A. Karr
Titan Systems Corp.
Billerica, MA, USA
dkarr@titan.com

Craig Rodrigues, Joseph P. Loyall and
Richard E. Schantz
BBN Technologies
Cambridge, MA, USA
{crodrigu, jloyall, schantz}@bbn.com

ABSTRACT

We demonstrate a standards-based middleware platform, including the QuO adaptive middleware framework and the TAO CORBA A/V Streaming Service, for developing adaptive multimedia applications that are better architected and easier to modify than ad-hoc architectures and that can adapt to changes in resource availability to meet QoS requirements. These are presented in the context of an Unmanned Aerial Vehicle (UAV) video distribution application. We present experimental results showing how the UAV application uses adaptive behavior to meet timeliness requirements in the face of restrictions in processing power or network bandwidth.

1. INTRODUCTION

Middleware for Distributed Object Computing (DOC) is an emerging and increasingly accepted tool for development and implementation of applications. Various middleware tools address the concerns of distributed multimedia applications. The CORBA architecture includes Audio/Video (A/V) Streams [3], a standard for transmission of multimedia data between processes and hosts. TAO, an open-source, real-time CORBA ORB from Washington University in St. Louis, provides an implementation of A/V Streams, the TAO A/V Streaming service [2]. The QuO framework, developed by BBN, supports the adaptive modification of the operation of an application in order to maintain quality of service (QoS) under a variety of run-time conditions, and to obtain suitable tradeoffs of QoS parameters when system resources are insufficient to meet all demands [1].

In this paper, we apply QuO to the development of an Unmanned Aerial Vehicle (UAV) video distribution application, in which an MPEG video flow adapts to meet its mission QoS requirements, such as timeliness. We discuss three distinct behaviors that adapt to restrictions in processing power and network bandwidth: reduction of the video flow

volume by dropping frames, relocation of a software component for load balancing, and bandwidth reservation to guarantee a level of network bandwidth. We have developed a prototype application which uses QuO, the TAO real-time ORB, and the TAO A/V Streaming Service to establish and adaptively control video transmission from a video feed via a distribution process to viewers on computer displays.

The application, a standards-based middleware platform, demonstrates that adaptation can be effectively controlled by a superimposed QuO contract to regulate performance problems in the prototype that are induced by heavy processor load on the host of a critical component. Our experience also shows that the use of the QuO framework, in contrast to typical ad-hoc performance-optimization techniques, results in code that is clearer and easier to modify, and promotes re-usability of software under different sets of requirements.

2. THE UAV APPLICATION

As part of an activity for the US Navy at the Naval Surface Warfare Center in Dahlgren, Virginia, USA, we have been developing a prototype concept application for use with an Unmanned Air Vehicle (UAV). A UAV is a remote-controlled aircraft that is launched in order to obtain a view of an engagement, performing such functions as spotting enemy movements or locating targets. A UAV can receive remote-control commands from a ship in order to perform such actions as changing its direction of flight or directing a laser at a target. Several UAVs might be active during an engagement.

To support UAV operations, a video camera on the UAV produces a video stream that must be displayed with minimal real-time delay on consoles throughout a warship. There are several steps to this process:

1. Video feed from off-board source (UAV).
2. Distributor sends video to hosts on ship's network.
3. Users' hosts receive video and display it.
4. Users analyze the data and (at certain work stations) send commands to UAV to control it.

Our prototype simulates the first three of these steps. The fourth step, in which some users interact with the UAV in

real time, is manifested in a QoS requirement: at least on certain display consoles, images must appear a very short time after the camera records them; we cannot buffer or interrupt the stream in order to accommodate variable latency and periods of network congestion.

3. THE QUO ADAPTIVE FRAMEWORK

3.1 The Need for Adaptation

There are several reasons a software system may need to adapt. One reason is limited resources. Multimedia applications can be heavy users of processing power and network bandwidth. Adequate resources to support these needs may not exist on all hosts or at all points in a network. Scalability is also a major concern, for example, the addition of many distinct display consoles to support a single UAV. Finally, other applications may need the same resources being used by a multimedia application, and users require some control over the allocation of resources among competing tasks.

A second reason is that different users, who may interact with an application at various times, often have different requirements. For example, a user controlling the flight of the UAV, or performing other remote-control actions, may need extremely timely information, so it is better to omit data than to deliver it late. Simultaneously, however, another user viewing the same stream may be performing off-line analysis that requires complete data even if it is delayed.

A third reason is that QoS requirements can change dynamically over the course of execution. For example, it might be that landing a UAV requires a video stream with lower latency than is needed for normal flight. Moreover, high-level command decisions by the ship’s captain (such as raising the ship’s defenses) will require resources to be shifted toward applications that are the most critical under the new circumstances.

3.2 The Need for a Framework

There are a number of reasons to utilize a middleware framework in order to implement adaptive multimedia computing. These reasons, which are closely related, include *separation of concerns*, a concept in the study of *aspect-oriented programming*, which states that various concerns of the program—for example various QoS properties, as opposed to the specific functions performed by the software—are best treated separately by developers when possible rather than being “entangled” together throughout the source code. Frameworks also support clarity of code by presenting these different concerns separately, often in specialized formats. As a result of these clear demarcations, speed of development can be improved, especially when modifying existing applications in light of new QoS requirements, since existing QoS-related code can easily be identified and enhanced, supplemented, or replaced.

3.3 The Value of QuO

QuO provides a suitable framework to support adaptive QoS in multimedia and other distributed applications. QuO incorporates a suite of “little languages” designed to support QoS in DOC. Its elements include *system conditions* that measure the status of resources and of resource requests; *callbacks* that can allocate resources or alter exe-

cutation strategies; *delegates* that wrap program objects and can alter the execution of method invocations; and *contracts* that, depending on combinations of comparisons of the system conditions, invoke the callbacks and/or control the operation of the delegates. Contracts support powerful adaptive abstractions, in particular the concepts of “regions of operation” mapped from the system conditions, and a “state machine” model driven by events. QuO’s languages and compilers provide automated support for various object architectures, including CORBA (in C++ and Java), Java RMI, and method invocation on native C++ or Java objects (including local method calls).

4. THE DESIGN OF THE PROTOTYPE

Our prototype application illustrates the following adaptive strategies that can be controlled by the QuO framework:

- Send a reduced amount of data, for example by dropping frames of the video. The resultant video appears as if the camera had simply captured fewer images per second, without affecting the speed at which objects in the scene move.
- Move the distributor from an overloaded host to a different host where more performance is available.
- Use a bandwidth reservation protocol to ensure that the distributor is able to send the necessary data to the viewers through the network, even when the network is congested.

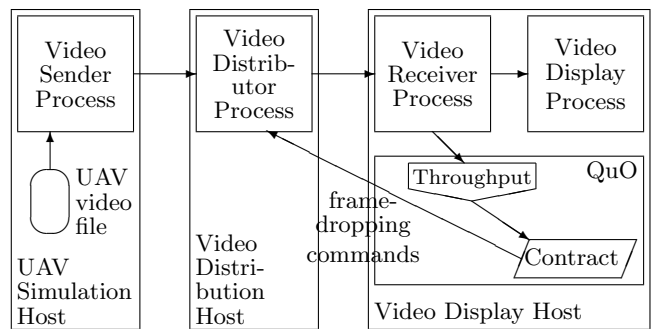


Figure 1: UAV Prototype Architecture

Figure 1 shows a recent version of the prototype architecture. It is a three-stage pipeline, with an “off-board UAV” (simulated here by a process on a separate host, reading data from a file) sending MPEG video to an “on-board” video distribution process. The video distribution process sends the video frames to video receiver processes, each of which is attached to a video display process that may have its own QoS requirements. Video transmission is accomplished via the TAO A/V Streaming Service, and all control paths are implemented by the TAO ORB. QuO controls adaptation by means of system conditions (measuring such things as video throughput actually achieved), contracts, and callbacks (for example to start or stop dropping frames).

The TAO A/V Streaming Service combines (1) the flexibility and portability of the CORBA object-oriented pro-

gramming model with (2) the efficiency of lower-level transport protocols. The stream connection establishment and management is performed via conventional CORBA operations. In contrast, data transfer can be performed directly via more efficient lower-level protocols, such as ATM, UDP, TCP, and RTP. This separation of concerns addresses the needs of developers who want to leverage the language and platform flexibility of CORBA, without incurring the overhead of transferring data via the standard CORBA interoperable inter-ORB protocol (IIOP) operation path through the ORB.

In ongoing work, we plan to increase the aspect orientation of the prototype application by generating QuO delegates to implement various aspects in addition to the adaptations discussed previously. These aspects include:

- Different transmission protocols. The prototype currently uses UDP, though we have also run it over TCP. These and other protocols could be made more readily interchangeable (with appropriate adjustments to adapt to their different QoS properties, of course).
- Headers added to transmitted frames. These are currently employed in an ad-hoc fashion to facilitate the handling of frames in the distributor and between adjacent stages. Abstracting this aspect would make it easier to alter headers to accommodate different transmission protocols and strategies.
- Measurement of latency and rate of frame delivery. Data such as these must be reported to system condition objects in order to control adaptation, but are measured by code in the path of the transmission.
- The frame-dropping mechanism. This must occur in the transmission path, because it takes the video data recorded each second as input and produces output with a reduced amount of data. We have begun work to separate this aspect and to improve the flexibility of where in the process it can occur.

5. RESULTS AND LESSONS LEARNED

An initial version of the UAV prototype application has been installed at NSWC Dahlgren and was demonstrated in the NSWC 2000 demo scenario. The current version is being provided as a source code release to researchers in related work, and is being integrated into the NSWC 2001 demo. The NSWC 2000 version illustrated the use of frame-dropping and migration of the distributor in response to excessive processor loads. The NSWC 2001 version illustrates bandwidth reservation as well.

5.1 Experimental Data

We performed the following experiment to test the effectiveness of the frame-dropping adaptation in the UAV application. The three stages were run on three Linux boxes, each with a 200MHz processor and 128MB of memory. The video transport was TCP sockets.

At time $t = 0$, the distributor started. Shortly after this, the video began to flow. At $t = 60$ seconds, $t = 62$ seconds, and $t = 64$ seconds, three special load-simulating processes were

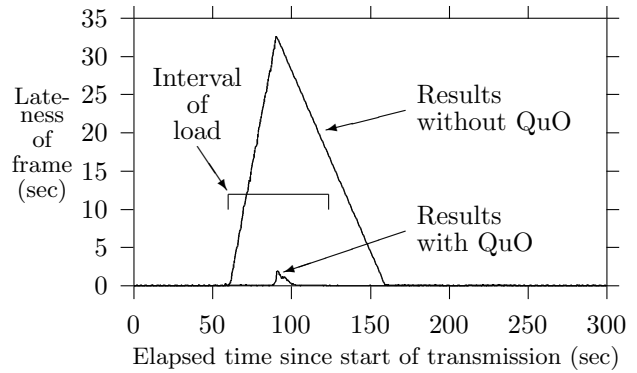


Figure 2: Lateness of frames received by viewer

started on the same host as the distributor, each attempting to use 20 percent of the maximum processing load (a total of 60 percent additional processing load). This reduced the distributor’s share of processing power below what it needed to transmit video at 30 frames per second. At $t = 124$ seconds, the load was removed. At time $t = 300$ seconds (approximately), the experiment terminated. The basic premise is that the full load was applied for a duration of one minute, starting after the pipeline had had time to “settle in,” and ending a few minutes before the end of measurement so we could observe any trailing effects.

This scenario was run twice, once without QuO attached and without any adaptation (the control case) and once with a QuO contract causing adaptation (the experimental case). For the purposes of this experiment, the only adaptation enabled was to reduce bandwidth by dropping frames. Because the video stream was in an MPEG-1 format, consisting of groups of pictures of 15 frames (in the typical sequence including one I frame, four P frames, and ten B frames), convenient levels of frame-dropping were to drop all B frames (and therefore transmit 10 frames per second), or to drop all P and B frames (and transmit 2 frames per second).

Figure 2 shows the effect of the increased load on the latency of the video stream. In this graph, the x -axis represents the passage of time in the scene being viewed by the UAV, and the y -axis represents the “lateness” of each image, that is, the additional latency (in delivery to the viewer) caused by the system load. That is, if all images were delivered with the same latency, the graph would be a constant zero. The label “Load” indicates the period of time during which there was contention for the processor. Without QuO adaptation, the video images fall progressively further behind starting when the contention first occurs, and the video does not fully recover until some time after the contention disappears.

The outcome of this experiment demonstrated that adaptation leads to improved performance of the application. The added latency caused by adverse system conditions (in this case, excessive CPU load) occurs in a sharply reduced magnitude and duration when adaptation is enabled, and the video image is continuously usable for its intended real-time purpose despite the fluctuation.

5.2 A Case Study

Our development experience serves as a case study in the value of QoS frameworks.

Reverse engineering of the third-party MPEG viewer used in the NSWC 2000 demo indicated that it was optimized to read input directly from a possibly slow disk (local to the display host) and display images via a possibly slow graphics card. It therefore employed extensive frame buffering (about 20 frames under normal operation) with algorithms to deal with starvation of the buffer, to “catch up” if a sequence of frames arrived late, and to drop frames when the graphics output was overloaded. These features raised the complexity of the code, but at the same time were unsuited to our requirements: in particular, when QuO directed the distributor to send a reduced number of frames, the viewer had a tendency to burst sequences of frames to the screen at its own predetermined rate rather than at the rate frames were sent or received. To control this burstiness, we added sophisticated code to constantly synchronize the frame rate of the viewer with the rate at which frames were received.

The end result was that adaptive strategies that took mere days or even hours to insert or modify in the components where QoS aspects were incorporated by the QuO framework, took weeks to add to this component with its ad-hoc entangled QoS.

In more recent versions of the UAV prototype application, the MPEG viewer was replaced by a different third-party implementation that does not have the unwanted QoS assumptions, and that displays the video frames at the rate received with minimal buffering. This has made the correct implementation of adaptive behavior much easier in the latest prototype. Moreover, since the viewer’s code was kept cleaner by not incorporating the ad-hoc QoS solutions of the older viewer, leaving open the possibility that these behaviors could be inserted via the framework if they were needed under other circumstances.

5.3 MPEG concerns

Another thing discovered during the project was that MPEG is not an ideal format for applications such as our prototype. A relatively minor concern was the apparent lack of support for any frame rates other than the standard modern film and broadcast formats. A larger concern with regard to the target application is the bidirectional interpolation of B frames between I and P frames, meaning that B frames cannot even be encoded until later frames are picked up by the camera, imposing a substantial latency between the video input and any kind of display. The latency can be reduced, but at the expense of bandwidth and/or loss of flexibility in frame-dropping patterns, by using an encoding of only I and P frames. A more flexible forward extrapolation scheme (allowing multiple P frames to be extrapolated directly from a single earlier frame) would likely work better, but does not appear to be supported by MPEG.

6. CONCLUSION

In this paper, we describe a prototype multimedia application, the UAV video distribution system, and the standards (MPEG video and CORBA A/V Streams transport) over which this prototype is implemented. We describe QuO, a framework for distributed object computing designed to enable adaptive optimization of distributed software systems. We demonstrate how QuO can interact with a multimedia DOC application in order to implement application- and implementation-specific adaptations to system performance issues. Further, we present empirical results that show that QuO can perform these adaptations effectively and efficiently. Moreover, our experience was that the use of the QuO framework made the implementation and redesign of adaptive behaviors easier for developers than ad-hoc methods typically used. These ad hoc methods involve extensive manual entanglement of code to perform basic functions with code to optimize performance under specific conditions, resulting in code that is difficult to develop, understand, and maintain. In contrast, the QuO-based adaptive behaviors are separated from the basic video functions, are easy to understand by inspection, and are easily modifiable.

7. ACKNOWLEDGEMENTS

This work is sponsored by DARPA and US AFRL under contract nos. F30602-98-C-0187 and F33615-00-C-1694. The authors would like to gratefully acknowledge the support of Dr. Gary Koob, Thomas Lawrence, and Mike Masters for this research. The authors would also like to acknowledge the contributions of Michael Atighetchi, Tom Mitchell, John Zinky, and James Megquier, Yamuna Krishnamurthy, Irfan Pyarali, Douglas C. Schmidt, the Naval Surface Warfare Center (NSWC) Dahlgren, VA (in particular Mike Masters, Paul Werme, Karen O’Donoghue, David Alexander, Wayne Mills, and Steve Brannan), and the DOC group at Washington University, St. Louis, and University of California, Irvine, to the research described in this paper.

8. REFERENCES

- [1] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proceedings of the 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, April 1998.
- [2] S. Mungee, N. Surendran, and D. C. Schmidt. The Design and Performance of a CORBA Audio/Video Streaming Service. In *Proceedings of the Hawaiian International Conference on System Sciences*, Jan. 1999.
- [3] OMG. *Control and Management of Audio/Video Streams, OMG RFP Submission (Revised)*, OMG Technical Document 98-10-05. Object Management Group, Framingham, MA, Oct 1998.