Designing Quality of Service Based Multimedia Systems Using the Unified Modeling Language

Muhammad Abdulla, Duminda Wijesekera, and Hassan Gomaa
Department of Information and Software Engineering
George Mason University,
Fairfax, VA 22030-4444, USA
{amhmd,duminda,hgomaa}@ise.gmu.edu
home pages: http://www.ise.gmu.edu/∼{amhmd,duminda,hgomaa}

***Abstract*— This paper addresses the usage of UML for requirement specification and design of multimedia systems. It shows how modular specifications of components used in QoS cognizant services increase their potential to reuse. It models common components such as negotiators, monitors, and resource managers using UML models enhanced with OCL constraints. The paper also describes how generic designs can be used to derive schedules for QoS cognizant services.**

***Index Terms*— Multimedia System, Quality of Service (QoS), Negotiation for QoS**

I. Introduction

Systems for which *quality* is an integral aspect of their functional correctness are generally considered Quality of Service (QoS) cognizant systems. Multimedia systems, networks, systems providing object services ([HV97]), and other performance centric software systems are some examples of QoS based systems. Over the recent past there have been tremendous advances made in these areas individually, but not as much in exploring the common characteristics that are solely due to having QoS as their underlying requirement. As the Unified Modeling Language (UML) ([RJB99], [BRJ99], [Gom00]) is becoming the lingua franca of software design, it is worth knowing the degree to which QoS based systems can be modeled using UML. This paper is an attempt to answer that question.

One of the questions that any QoS based service designer encounters is the representation of QoS parameters. Because there are numerous services from different application domains that use QoS parameters, there is no generally accepted set of QoS metrics. Specifications of such systems should be precise and feasible. Specifying a television broadcast as one that makes most viewers happy is an example of a realistic but ambiguous specification. Specifying a zero transmission delay for network services is unambiguous but an unrealistic specification as no network can provide zero transmission delay.

In ensuring that a QoS aware system fulfills its correctness criteria, it must rely on resources of the operating platform during the delivery of specified services. This is due to the finite nature and limitations on resources of any given operating platform. In the past, QoS sensitive systems have obtained such assurances from underlying operating platforms by utilizing appropriate service reservation, scheduling and exception handling mechanisms; and based on them entered into QoS contracts with service suppliers and component managers. Such QoS contracts are binding agreements between service providers and consumers, and either party may need to know if the other party lives up to its obligation. In order to do so, some service providers and consumers resort to monitoring the provided and utilized quality in terms of QoS parameters and take corrective action in case of contractual violations. Another usage of QoS monitoring is the recent trend of pricing QoS aware services, based on provided levels of QoS. We explore the consequence of such QoS contracts and monitoring throughout the software life cycle.

As stated, the objective of this paper is to examine the extent to which QoS sensitive applications can be designed using the UML. We seek generic, precise and useful modeling artifacts to fulfill our objectives. By examining numerous applications that use QoS parameters, we have identified some common characteristics of most QoS sensitive applications. We show how to model these common characteristics with the UML notation. We show their usefulness by indicating how they can be used in influencing scheduling decisions made by underlying operating systems and networks. The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes UML modeling of QoS cognizant systems in the requirement specification phase. Section 4 describes QoS modeling in the analysis and design phase. Section 5 presents possible usage of UML specifications in multimedia scheduling. Section 6 gives our conclusions.

II. Related Work

There has been a considerable amount of work modeling resources using UML. Selic [Sel00] describes a general framework for modeling resources using UML. A high-level view of resource modeling and the suitability of UML models to describe such frameworks are shown. This can be used as a general guideline for specifying particular example in modeling QoS cognizant systems. As will be shown here, UML diagrams alone are insufficient for precise and detailed specifications. In this paper, we examine the way and extent to which UML diagrams can be used and the utilization of OCL for precise modeling.

Gomaa [Gom00] and Benyoucef et al. [BAVK01] use examples of electronic commerce applications. In these frameworks, the negotiation takes place at the beginning. Once the deal is made, there will be no more negotiation. However, in a multimedia system as described in our example, QoS parameters for each movie being serviced can change over time, necessitating the need to build a dynamic framework where the negotiation process can continue until the movie has been completely delivered. We show how such negotiation frameworks can be designed.

Blair et al. [BBBC98] present a comprehensive description of distributed multimedia systems and their formal specifications. However, our attempt is to describe generic properties of QoS cognizant systems within framework of UML rather than using formal methods. Consequently, we provide details of specifying negotiation and monitoring frameworks and place less emphasis in formalizing properties of multimedia system etc.

Menacsé et al. [MBD01] describe an approach for preserving QoS of e-commerce sites by self-tuning, which is accomplished through analyzing the service quality provided within a time window in the immediate past to adjust its future performance characteristics. The main focus of the work is to provide an overall optimization on the server for all requests. No negotiation between client and server are considered. However, in some QoS applications it is possible for requests to have different QoS requirements, for which the server should arrange resources accordingly. We show how to model and design such systems where each request from clients should be considered separately according to their QoS requirements.

III. Modeling QoS Aware Services

In this section, we explore the nature of QoS specifications, and examine some common characteristics of quality sensitive services. We then show how they can be modeled using specification mechanisms provided by the UML language. We show how appropriate UML models can be used to specify and design QoS cognizant software throughout its life cycle. We begin this process with the requirements gathering phase.

A. QoS Modeling in the Requirements Phase

As stated, quality of service specifications arise with precise specification of QoS metrics and their acceptable ranges. For example, in multimedia services frame rates (i.e. frames per second), end-to-end service delay, jitter (i.e. delay variation between frames in a stream), drop rate (i.e. the percentage of lost frames), and inter-stream synchronization between audio and video are some specifiable QoS parameters. Detailed descriptions and properties of such QoS parameters appear in [WCH] and [CCH00]. In network services, bandwidth, its allowable variations, end-to-end delay, and drop rates are specifiable parameters. Secondly, not only the parameters, but also their acceptable ranges matter and are application specific. For example, 30 frames per second is ideal for video transmission in North America, but a minimum of 18 frames per second is necessary in order to maintain the sense of *motion* for human viewers. Also three minutes of end-to-end delay with less than two milliseconds of difference between audio-video starting times (mis-synchronization) may be acceptable for human viewers, but a video based automated target tracking device may not tolerate more than half a second delay irrespective of audio-video mis-synchronization. Hence, all QoS parameters and their acceptable ranges should be specified.

Secondly, as stated in Section I, both service providers and consumers are aware of the service provider's limitations in not being able to always guarantee the contracted service quality due to unforeseeable circumstances beyond their own control. For example, in providing audio-video services, some service interruptions or degradations at the server's input/output services may require the frame rate to be lowered, or the quality of frames be degraded. In transport layer services of networks, congestion may result in temporarily lowering the bandwidth and increasing the end-to-end delay. Some applications may be able to withstand such deviations from contracted minimal requirements, while others may prefer a temporary suspension of services. Therefore, not only the acceptable QoS levels, but also a handling method for exceptions needs to be specified in service contracts between the providers and the consumers of QoS sensitive services.

Thirdly, the nature of the contractual agreement is a part of the specification. For example, one server may wish to set its own limits for a service, while another may be able to accept consumer requests and make counter offers based on its abilities and permissions. This is specially true in today's web based services, where new negotiation paradigms are being experimented with. Therefore, the negotiation framework need to be considered with QoS specifications.

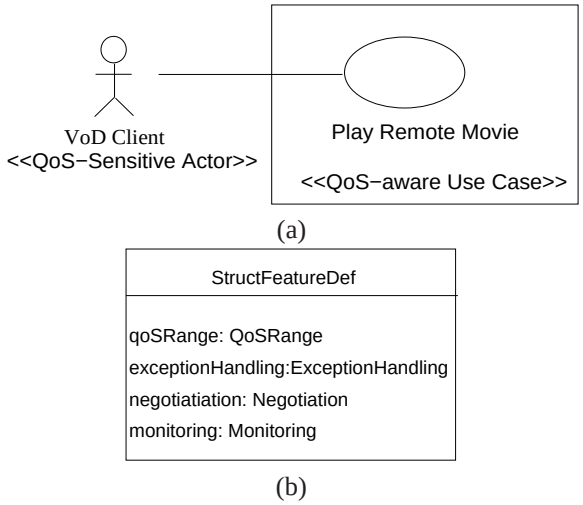


Fig. 1. UML Extensions for Use Case

Fourthly, to avoid exploiting QoS parameters either by the provider or the consumer of service, and to use as a condition for raising an exception or to aid in the continued negotiation process, most (but not all) providers and consumers of QoS aware services use monitoring. Accordingly, monitoring capabilities must be a part of the requirements specification of a QoS cognizant specification.

B. QoS Aware Use Case

In design methodologies that use the UML design language, Use Case models are used to specify the user's requirements, sometimes referred to as stakeholders interests in the system. In order to utilize use cases for QoS based services, they need to incorporate the following QoS requirements:

1) QoS parameters and their acceptable values or value ranges.
2) Exception handling
3) QoS negotiation framework
4) QoS monitoring

To accommodate these aspects, we extend use case models to support *QoS aware use cases* where an actor in a QoS aware use case is a *QoS sensitive actor*. As shown in Figure 1, these extensions are stereotyped ([RJB99]) as ≪`QoS aware use case`≫ and ≪`QoS sensitive actor`≫. In order to extend use cases and actors, we propose four enhancements to the `StructFeatureDef` class in the UML meta-model ([WK01]). The enhancements we propose are to include a range for QoS parameters, an exception handling mechanism and negotiator and a monitor, as shown in Figure 1(b).

We illustrate the use of theses aspects by an example specification and design of a *video on demand (VoD)* multimedia server. Our VoD service uses a client-server paradigm. Clients request movies with specified QoS parameters (e.g. frame rate, end-to-end delay, jitter, drop rate, and synchronization) for delivery. If the server cannot satisfy the required movie QoS, a negotiator in the VoD server bargains and negotiates acceptable QoS parameters with the client. In addition, the server also maintains a QoS monitor to provide QoS corrections to the service, while the service is being provided.

We will discuss the requirement analysis and design issues through *multiple view model* with UML. A multiple-view model captures different aspects of a software product line, for functional modeling, static modeling, and dynamic modeling. Using the UML notation, the functional view is represented through a use case model in the requirements phase, a static model view through a class model, and a dynamic model view through a collaboration model and a statechart model.

The use case model view addresses the functional requirements of a software product line in terms of use cases and actors. An actor is a user type. A use case describes the sequence of interactions between the actor and the system, considered as a black box. Use case descriptions for the VoD example are shown in Figure 2.

Now we give a typical usage scenario for our example VoD server. Suppose the server receives a request for a movie `Doll.mov` with the following QoS parameters.

- 30 frames/second (fps) for the audio with neither jitter nor drops.
- 30 fps for video with a jitter of less than 3ms. and a maximum drop rate of 0.1%.

The server checks and finds out that `Doll.mov` is available in its archive, and makes a counter offer of 25 fps but agrees to the other QoS parameters. Finally, the QoS negotiation results in, say, 28 fps, 0.01 drop rate for both audio and video streams, and a maximum jitter of 3ms for video and 0ms for audio streams. The server then begins to send the movie and starts monitoring its services. The monitor checks for the QoS periodically and informs the server of QoS violations. For example, if the drop rate of video streams increases to 0.02, the monitor will make the server renegotiate the QoS contract. Say the new negotiation ends up with an updated QoS contract of a maximum drop rate of 0.01 at 25 fps with less than 3 jitter rate for the video. This process can go on dynamically until the movie transmission is complete or the negotiation is unsuccessful.

IV. Software Architecture for QoS Systems

Media types, service components for each media type, and the structure of QoS systems should be considered at the architectural level.

It is natural to assign a *media component* to each of the media types of the QoS system such as audio, video, and

Use Case: Play Remote Movie
Scope: QoS Cognizant Multimedia Server System
Goal: Play a movie with specified QoS from a remote server.
Context of Use: Multi-client VoD system.
Primary Actor: VoD Client.
Main Sequence:
1. VoD Client requests movie with specified QoS on delivery.
2. VoD Server acknowledges requests and informs client if requested movie is available.
3. Server obtains available QoS parameters.
4. If required QoS parameters can be satisfied, the Server sends a positive acknowledgment message to the client.
5. The Server begins to send the movie and turns on the QoS monitoring, QoS negotiation, and QoS exception handling.
6. The Server sends an ending message at the end and close the connection.
Alternatives:
1a. Un-acknowledged client times out.
1b. Request for unavailable movies are denied.
1c. Syntactically incorrect or semantically invalid requests are refused.
1d. Defaults are substituted for unspecified or under-specified QoS parameters.
2a. If the service capacity is exceeded, the client request is queued.
2b. When feasible, queued requests are serviced.
3a. If service is overloaded, server returns service unavailable message to client.
4a. Service with QoS parameters outside the serviceable range (such as video frame rate higher than 50 fps or audio-video jitter larger than 5 ms), are refused.
4b. For service within serviceable range of QoS parameters, but cannot be met due to service loads, the server's negotiator enters into a QoS negotiation session with the client.
6. If there are any QoS violations reported from the monitor during the service, the server warns the client and enters in to QoS negotiation session again.

Fig. 2. Use Case Description

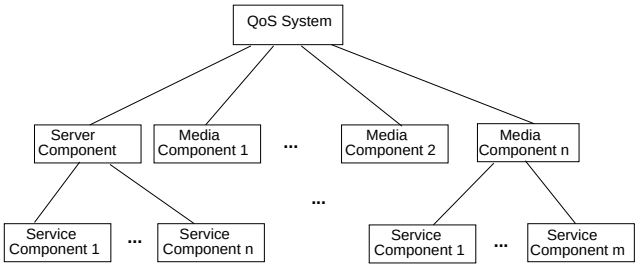


Fig. 3. General Architecture of QoS Systems.

text, etc. However, if there is a tight correlation between two components, it might be preferable to merge them into one. For example, for a VoD server with an optional language selection for audio and on-screen text, text and audio components can be merged as a single media component.

Service components for each media component should be assigned according to the requirements of the applications. Possible services for media components include negotiation, monitoring, synchronization, resource management, and real-time scheduling etc. Besides, a *server component* is needed for managing media components. Server and media components may or may not have the same service components. A general architecture of QoS systems is given in Figure 3.

V. QoS Modeling for the Analysis and Design Phases

The use cases proposed in the requirements gathering phase must be analyzed and corresponding software artifacts designed during the design phase of the software life cycle. Based on this analysis, we propose that the QoS cognizant VoD server consists of three major service components as follows:

- QoS negotiator
- QoS Monitor
- Resource manager

The first two components, the QoS negotiator and the QoS monitor emerge as direct consequences of the requirements, as the requirements specify the need for negotiation and monitoring. The third component, the *resource manager* is necessary because proper management of system resources is critical for the performance of QoS cognizant services. The resource manager is consulted by the negotiator in order to find out if the available amount of system resources in making admission control decisions and making appropriate counter offers to the requested QoS parameters.

In particular, QoS cognizant services such as multimedia, by their very nature require the combination of many sub-services such as video and audio that may be cognizant of sub-component QoS parameters. An example is that a movie QoS may be specified in terms of audio QoS, video

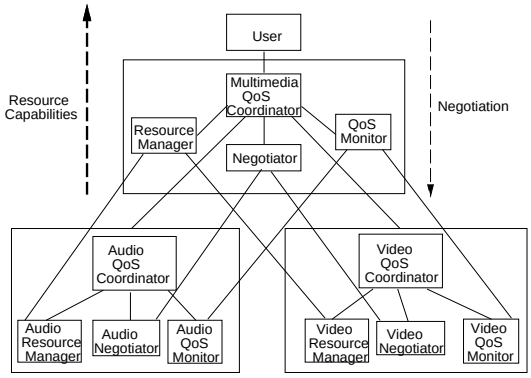
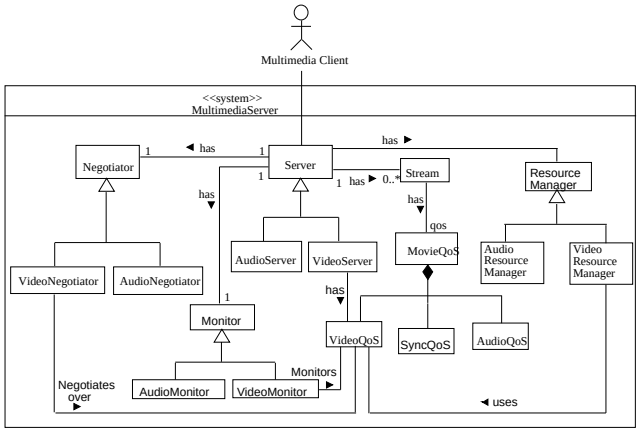
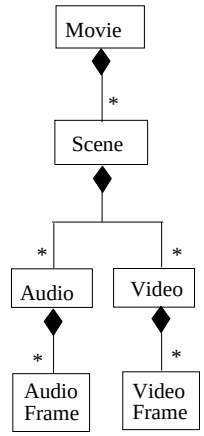


Fig. 4. Static Structure of Multi-layered QoS Negotiation.



(a) Class Diagram of Audio-Video Client-Server Application



(b) Movie Structure

Fig. 5. Server and Media Class Diagrams.

QoS and synchronization QoS parameters. Therefore, making an estimate of available service capacity requires that these components compute their respective capacities and provide estimates so that the available capacity for the complete service can be estimated.

What emerges from our analysis is the realization that both the multimedia QoS server and audio, video server components share the same structure of having negotiator, resource manager, and monitor components. The negotiator negotiates with higher or lower layers over QoS, the monitor checks QoS parameters against the service, and the resource manager handles allocation of resources such as CPU, buffer, etc. Components of the higher layer negotiate with corresponding components in the lower layer to get the available QoS capabilities. They are shown in Figure 4. Secondly, QoS parameters need to be specified using QoS parameters of sub components and parameters qualifying their combination, which we refer to as the *synchronization* component.

A. Static Modeling

The static model is used to depict the static structural aspects of a software product line by modeling classes, their attributes and relationships between classes [Booch99]. Objects in the collaboration model are instantiated from classes in the class model.

Following the generic model shown in Figure 4, a class diagram for the VoD server is given in Figure 5(a). The server and its subsystems share a general structure: Each of them has resource manager, negotiator, and monitor parts whose functions are described above. The monitor class for example will be responsible for checking QoS parameters of multimedia components such as audio, video, and synchronization.

A QoS specification for a movie is shown in Figure 5(b). In the example specification, the movie consisting of an audio stream, and a video stream. Frame rates and drop rates of individual streams are parameters of individual components and allowable audio-video mis-synchronization limits in milliseconds are metrics of the combinations. Synchronization components are implicitly assumed and are used to describe inter-component synchronization between audio and video streams and intra-component synchronization, such as maximum jitter within video frames [CT02].

The class diagram given in Figure 5(a) has associations between VideoMonitor and VideoQoS, etc. The constraint that VideoServer will only use VideoQoS also can be expressed in OCL as follows:

Context: VideoServer
server.stream.qos −> forAll (oclType = VideoQoS)

Other constraints corresponding to relationships between QoS types and the subcomponents of the negotiator, monitor, and Resource Manager can be specified similarly.

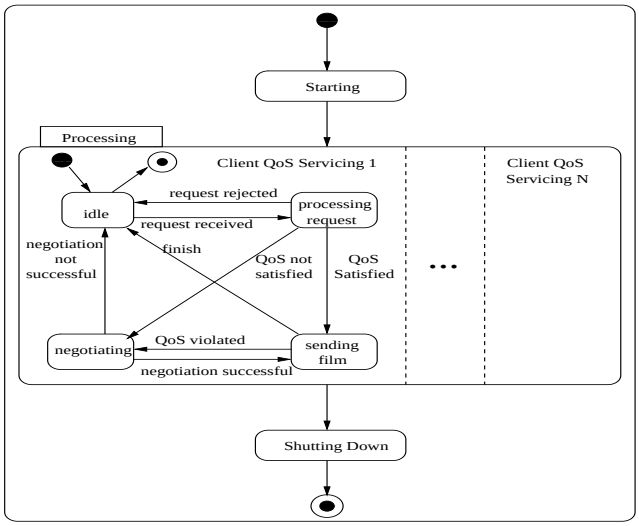


Fig. 6. Statechart for the Server.

B. Dynamic Modeling

As discussed earlier, a dynamic model view is represented through a collaboration model and a statechart model. The statechart model view, along with the collaboration model view, addresses the dynamic aspects of a software product line. A statechart is developed for each state dependent object in the collaboration model. Each state dependent object in a collaboration diagram is specified by means of a statechart.

The statechart diagram for the VoD server shown in Figure 6 describes the state dependent behavior of the server, where the `Processing` substate includes several concurrent states as a form of an `and`-state, each of which specifies the state dependent behavior of client QoS request servicing. The Statechart assumes that there is a fixed upper bound of `N` to the number of requests that can be processed by the server.

The collaboration model view addresses the dynamic aspects of a software product line. It is used to depict the objects that participate in each use case, and the sequence of messages passed between them [BRJ99], [Gom00].

The collaboration diagram shown in Figure 7(a) shows the initialization phase of the system. The multimedia QoS coordinator initializes Resource Manager, Negotiator, and Monitor only once for the first request and other objects are initialized for each request. When a new request is received, the Multimedia QoS Coordinator will start Audio QoS Coordinator and Video QoS Coordinator, each of which will initialize corresponding resource managers, monitor, and negotiator. Figure 7(b) shows the interactions between objects for a successful service including a scenario in which specified QoS is violated. In this phase, the Resource Manager allocates resources for the current transaction by communicating with Audio and Video Resource Managers. The Monitor checks for QoS violations in audio, video, or in audio-video synchronization. Negotiator will be responsible for building new service agreements with the client in the case of QoS service degradations.

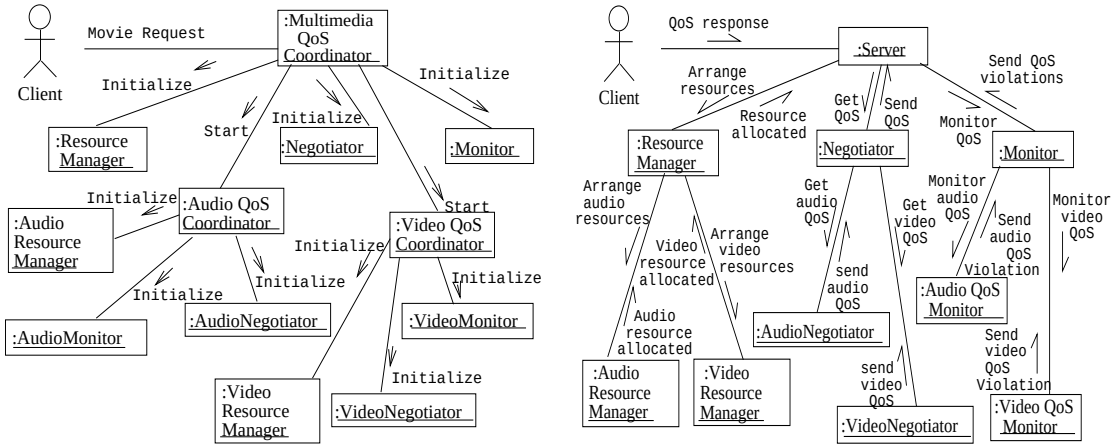
A more detailed interaction between different components of the video server is given in a sequence diagram as shown in Figure 8. The diagram depicts a transaction between the server and the client, with a QoS violation. First, the client asks for a movie with some QoS parameters (sequence 1.1). The server obtains and offers its available QoS (1.2–1.12) and, if accepted by the client (2.1), begins to send the movie (2.2) and asks the monitors to monitor the QoS parameters (2.3–2.7). In the case of a QoS violation (2.8), the server informs the client (2.9) and negotiates for a new QoS (2.10–3.1). This process continues till either the negotiation is not successful or the movie ends (3.2). Due to space limitations, interactions between resource managers and other components are not included.

Behavior of proposed components can be further specified by writing appropriate OCL expressions. For example, the video monitor and the synchronization monitor can be specified in OCL syntax as follows.

Context: VideoMonitor
Operation: checkQoS (s: Stream)
if (s.videoFrameSpeed $\neq$ 30)
  return false
if (s.videoJitter $>$ 0.003)
  return false
if (s.videoDropRate $>$ 0.001)
  return false
return true

Context: Monitor
Operation: checkSynch (s: Stream)
if (s.AudioFrameNo $<$ s.VideoFrameNo - 1)
  return false
if (s.AudioFrameNo $>$ s.VideoFrameNo + 3)
  return false
return true

The OCL constraint for the `VideoMonitor` states that maximum speed should be 30 fps, video jitter (i.e. delay variation between successive video frames) should be less than `3`ms and the drop rate should be less than `0.1`%. The OCL constraints of the `Monitor`'s checkSync method states that audio frames should not be behind video frames by more than one frame, and video frames should not be behind audio frames by more than three frames. As shown in Figure 8, if there is any violation of QoS parameters, say an increase in video drop rate that exceeds 0.1%, the Audio QoS Monitor



(a) Initiation Phase (b) Transmission Phase

Fig. 7. Collaboration Diagrams of the Multimedia System

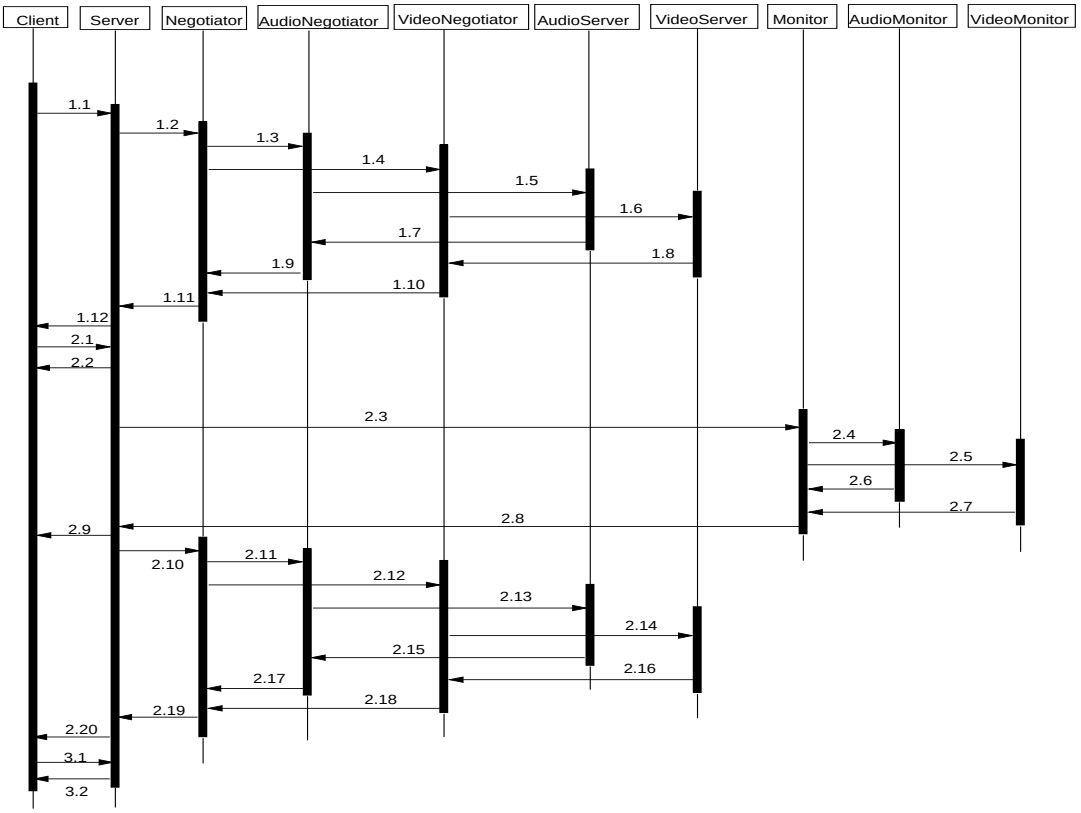


Fig. 8. Sequence Diagram of Audio-Video Client-Server Application with a QoS Violation.

will send a message to Monitor, which in turn informs the Server. The Server will ask the Negotiator for a new set QoS parameters with the Client.

As shown in this previous section, low-level QoS parameters such as delay, latency, jitter, etc. are more easily specified in OCL than high-level QoS characteristics such as availability, guarantee, and mean time between failures. Ongoing work is addressing how the latter parameters can be specified as combinations of the former parameters.

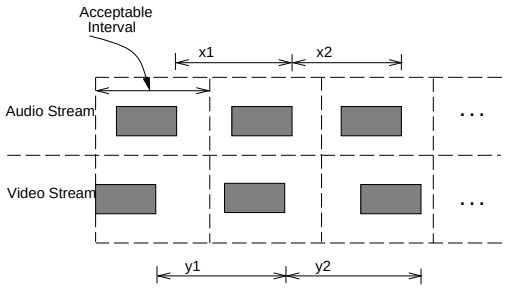


Fig. 9. Audio-video Scheduling

VI. Using UML Designs for Scheduling

Specifying QoS parameters at requirements and design phases can help to derive decisions on scheduling as shown in Figure 9. Assuming, for example, a set of QoS parameters are specified as follows:

Context:AudioStream
self.jitterRate = 0
self.dropRate = 0
Context:VideoStream
self.jitterRate $\leq$ 0.003
self.dropRate $\leq$ 0.001
Context:Stream
self.frameRate = 25
self.consecutiveSyncDrift $<$ 10

How scheduling can be done for the constraints given above can illustrated using Figure 9. The scheduler can derive that the length of the ideal presentation interval should be between 40ms, according to the constraint that the frame rate for the stream is 25Hz. Given that the audio jitter should be equal to 0, we can deduce that no difference in the presentation time is tolerated. Therefore, the presentation interval between audio packets, x_i, should be equal to the ideal presentation interval. From the constraint that the video jitter should be less than or equal to 0.003(s), we can find out that difference between length of presentation interval for video, y_i, should be no more than 3ms.

Constraints for drop rates indicate that audio and video drop rates should not be more that 0% and 0.1%, respectively. Consecutive synchronization drift ([WSNF99]), the largest difference in presentation between two different media components, is given to be 10(ms). This constraint asserts that the variation in presentation time between audio and video frames should not exceed 10ms. More constraints can be specified for other requirements about media presentation and synchronization. Further details on deriving scheduling from QoS specifications can be found in [PSW00] and [WSNF99].

VII. Conclusions and Future Work

Modular specification of components used in QoS cognizant services increase their potential for reuse. We have shown how some common components such as negotiators, monitors and resource managers can be specified using UML models enhanced with OCL constraints. As UML is emerging as the standardized design language, these components can be reused by different applications. We have shown how such generic designs can be used to derive schedules for QoS cognizant services.

Design models can be used in test case generation. However, design models alone cannot express some of the design constraints of the system. Therefore, generating test cases from design models and OCL expressions is useful to have a more complete test of the system under development.

After basic functional requirements are met, validity of the software under development will be directly dependent upon the performance of the system [WV00]. Gomaa and Menascé [GM00], [MG00] and Grassi et al. [GM01] discuss a UML approach based on performance analysis at architecture level. However, there is limited literature on performance testing using UML specifications. Future work includes using OCL to test the functional and performance characteristics of software systems.

References

[BAVK01] M. Benyoucef, H. Alj, Vezeau, and R. Keller. Combined negotiations in e-commerce: Concepts and architecture. *Electronic Commerce Reasearch, vol. 1*, 2001.

[BBBC98] G. Blair, L. Blair, H. Bowman, and A. Chetwynd. *Formal Specification of Distributed Multimedia Systems*. UCL Press, London, UK, 1998.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[CCH00] Andrew Campbell, Geoff Coulson, and David Hutchison. *A Quality of Service Architecture*. Department of Computing, Lancaster University, 2000.

[CT02] Stefan Conrad and Klaus Turowski. Temporal OCL: Meeting Specification Demands for Business Components. 2002.

[GM00] Hassan Gomaa and Daniel Menascé. Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures. *Proceedings Workshop on Software Performance, ACM Press, Ottawa, Canada*, 2000.

[GM01] Vincenzo Grassi and Raffaela Mirandola. UML Modelling of Performance Analysis of Mobile Software Architecture. *UML 2001, The Unified Modeling Language*, 2001.

[Gom00] Hassan Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley Object Technology Series, 2000.

[HV97] Michi Henning and Steve Vinoski. *Inside CORBA: Distributed Object Standards and Applications*. Addison-Wesley, 1997.

[MBD01] Daniel A. Menascé, Daniel Barbará, and Ronald Dodge. Preserving QoS of E-commerce Sites through Self-Tuning: A Performance Model Approach. *EC'01*, 2001.

[MG00] Daniel Menascé and Hassan Gomaa. A Method for Design and Performance Modeling of Client/Server Systems. *IEEE Transactions on Software Engineering, Vol. 26, No.11*, 2000.

[PSW00] Raymond A. Paul, Jaideep Srivastava, and Duminda Wijesekera. Test and Evaluation of Distributed Information Ssystems Network. *Journal of the International Test and Evaluation Association*, 2000.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[Sel00] Bran Selic. A Generic Framework for Modeling Resources with UML. *IEEE*, 2000.

[WCH] D. G. Waddington, G. Coulson, and D. Hutchison. *Specifying QoS Multimedia Communications within Distributed Programming Environment*. Department of Computing, Lancaster University.

[WK01] Jos Warmer and Anneke Kleppe. *Unification of Static and Dynamic Semantics of UML*. UML 2001-The Unified Modeling Language, Fourth International Conference, Springer Press, 2001.

[WSNF99] Duminda Wijesekera, Jaideep Srivastava, Anil Nerode, and Mark Foreti. Experimental Evaluation of Loss Perception in Continuous Media. *IEEE Multimedia Systems*, 1999.

[WV00] Eliane J. Weyuker and Filippos I. Vokolos. Experience with Performance Testing of Software Systems: Issues, and Approaches, and Case Study. *IEEE Transactions on Software Engineering*, 2000.