Quality of Service Specification for Resource Management in Multimedia Systems

Richard Alan Staehli
B.S., The Evergreen State College, 1982

A dissertation submitted to the faculty of the
Oregon Graduate Institute of Science & Technology
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

January 1996

The dissertation "Quality of Service Specification for Resource Management in Multimedia Systems" by Richard Alan Staehli has been examined and approved by the following Examination Committee:

Jonathan Walpole
Associate Professor
Thesis Research Adviser

David Maier
Professor

James Hook
Associate Professor

John Nicol
Principal Member Tech. Staff
GTE Laboratories, Inc.

Dedication

To my wife, who yearns for me to become a gardener.

Acknowledgements

The most important person to acknowledge is my advisor Jonathan Walpole. Without his encouragement and advice, I would not have entered the Ph.D. program at OGI. I have always felt strongly that taking time to hike, to climb, to ski and sail are as essential as any other pursuit in this life. Jonathan demonstrated that one can continue outdoor pursuits and still pursue science with a dedication that I had previously associated with digging a snow cave in a blizzard. In addition, many of the ideas in this thesis evolved slowly from our talks so that it is impossible to claim sole credit. Both Jonathan and Dave Maier have provided suggestions and feedback, beginning with my research proficiency paper five years ago and continuing up to our most recent publication. While this thesis represents work that I have developed and written on my own, I gratefully acknowledge the valuable criticism that Jonathan and Dave have provided.

I would also like to thank the students that have formed my at work family for these past years. Ravi Konuru and Harini Srinivasen were good friends through our first year of classes and the qualifying exams. They each inspired me by finishing their thesis work in good time and with good results. The other members of my "quals group" include Judy Cushing, Steve Rehfuss, Jon Inouye, Moira Mallison, and Bennet Vance. All have been valued friends and colleagues for sharing and learning the tricks of Ph.D. candidacy. I owe Judy a unique debt for having introduced me to computer programming when I took her Data Structures course at The Evergreen State College in 1982. Her letter of recommendation was probably a key to landing my first job out of college! Her creative energy continues to provide a fountain of opportunities for research and professional collaboration. I'd also like to acknowledge the many times that Bennet listened to my description of a problem that I could neither solve nor articulate well. It proved useful to know what parts of the problem I could communicate to him. In addition to these close friends that

took the qualifying exams with me, I need to mention Scott Daniels, who was like a big brother to me, Jenny Orr, and Nanda Kambhatla, who have always formed the core of the party crowd in CSE, such as it was. I regret that we didn't all have more time together for recreation.

I've deliberately delayed acknowledging the assistance of Jon Inouye. Of course, I need to thank Jon for the official LaTeX OGI Ph.D. thesis template. This is only one of many examples where Jon was so capable and willing to provide expert consultation on tools that I used without studying. I knew from previous industrial experience that an organization works best if it cultivates an expert or two for each tool. These experts take responsibility for maintaining and upgrading the tool and provide intelligent tutoring to other tool users. Jon Inouye has provided this expert service for LaTeX, HPUX, Mosaic and many other tools that have been essential in writing my thesis. His sushi-making skills and provisioning of the lab cookie jar are also much appreciated.

Other members of the faculty that have been of particular help include Jim Hook who taught a mean Automata Theory course in his very first quarter of teaching at OGI, and David Novick who was my first mentor and also advised me to enter the Ph.D. program. Harry Porter, a graduate student at the time, introduced me to object-oriented programming in his 1985 OOP class. Roger Lea, while only an adjunct faculty member for a brief time, gave me some of the most memorable advice about completing my Ph.D. work and its only through my own stubborn nature that I was unable to follow it well. I am also very grateful to Tom Little of Boston University for inviting me to meet with his research group in May of 1994 and to John Nicol of GTE Labs for inviting me to speak there during the same trip. These groups endured a painfully early version of my thesis talk. Of course, I am very grateful to the members of my thesis committee for each giving this thesis a careful reading.

Financial support for this work came from National Science Foundation grant IRI-9117008 for Multimedia Storage research and IRI-9223788 for Scientific Database research. Additional support came from Tektronix, Inc., the Portland Trailblazers, and from AT&T Bell Labs. I would particularly like to acknowledge Bart Locanthi from Bell Labs who built and loaned us our first digital video cameras. My first programs for capturing

and manipulating video images were executed on a MIPS Magnum running the Plan 9 operating system that Bart had installed and loaned to us. I deeply regret that we were not able to make a commitment to use Plan 9 for our other work. The opportunity to work with such a well crafted and documented set of tools was a great joy.

Special thanks to my wife, Jan, for providing both pressure to complete and respite from the task of writing this thesis. We have acted together to see this goal achieved, not knowing how it would change our lives.

Contents

List of Tables

List of Figures

Abstract

Quality of Service Specification
for Resource Management
in Multimedia Systems

Richard Alan Staehli

Supervising Professor: Jonathan Walpole

Digital multimedia systems are rapidly becoming ubiquitous with nearly all computer platforms offering support for audio and video. Multimedia computing promises to augment or replace most of the traditional broadcast and print media with more interactive and personalized information services. Unfortunately, today's real-time multimedia services are either tailored to a personal computer environment or are vulnerable to performance degradations in a shared environment. Designers are faced with two fundamental problems:

- choosing a digital representation for continuous media, and
- scheduling resources to approximate a real-time presentation.

While some multimedia systems take an ad hoc approach to these problems, an optimal solution requires a complete specification of presentation quality requirements.

This thesis offers the first complete framework for specifying presentation Quality of Service (QOS) requirements. Beginning with a formal definition of an ideal presentation, the thesis describes a quality estimation function based on error in the presentation

outputs. This approach allows device and data independent descriptions of multimedia services. We provide a detailed example of a formal QOS specification composed of orthogonal *content*, *view*, and *quality* descriptors. These descriptors are designed to support useful, complex multimedia presentations and to have a simple formal semantics. The practicality of the QOS specifications are demonstrated by a multimedia player that translates QOS requirements at runtime into acceptable presentations with near-optimal resource use.

Chapter 1

Introduction

1.1 Multimedia Systems and Resource Management

Multimedia systems facilitate better communication between people through the creation and exchange of multimedia information. A multimedia presentation uses complementary sensory channels to communicate more effectively and often more quickly than with a single media type, such as text. In particular, video and audio take advantage of our ability to recognize important information quickly through sight and sound.

Another important feature of multimedia systems is that natural sensory information can be recorded and reproduced without interpretation. Audio and video streams typically contain a large portion of irrelevant information that is not easily separated from essential information. Today's successful multimedia systems handle audio and video data types as *Binary Large Objects* (BLOBs) whose meaning is understood only by the user [68].

As audio and video capabilities are added to nearly every workstation and personal computer, the computer is assuming a new role as the smart user-interface to unified communication and information services. Computers add value to analog media through navigation support and information management services.

We define a *multimedia presentation* to be the digital output representation of multimedia information. This definition deliberately eliminates consideration of digital-to-analog conversion and display, because these mechanisms are typically not under software control. A presentation may include digital video and audio, as well as synthetic compositions such as slide shows and computer-generated music. We call these presentations *time-based* because they communicate part of their information content through presentation timing.

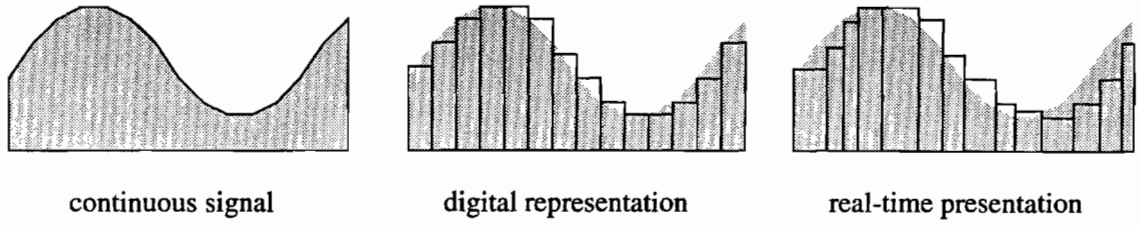

Figure 1.1: Loss of quality in digital representation and presentation.

Since digital video typically has higher bandwidth requirements than other data types, we use it for most examples with the understanding that analogies can be drawn for other time-based data types. This thesis concentrates on the problems of real-time multimedia presentations from stored data. However, since our QOS specifications make no assumptions about data location, our results may be applied to presentations of live data sources as well.

What is so new about time-based presentations? While presentations of text and numeric data are generally expected to be correct, for time-based presentations it is frequently impossible to compute and display correct results in real-time. For example, a stored video that is encoded at 2 Mbps cannot be played in real-time over a 1 Mbps network connection without skipping frames or otherwise losing part of the information stream. We refer to this conflict between resource capacity and presentation timing as the *real-time presentation problem.*

Some information loss is also inevitable in any conversion of continuous media between analog and digital representations. This loss occurs not only when the data is initially captured, but also in lossy conversions between digital encodings, such as when a 24-bit color image must be dithered for an 8-bit display. We refer to this conflict between preserving information and digital encoding as the *digital representation problem.*

Figure 1.1 illustrates the real-time presentation problem and the digital representation problem for reproduction of a continuous audio signal. These two problems are related through the resource requirements for data processing, storage and transport. For example, a real-time presentation may be achieved with limited resources by using a highly compressed digital encoding, such as MPEG, that sacrifices image quality for reduced

storage-volume and transport-bandwidth requirements [23, 74]. In fact, both problems have trivial solutions if the application allows arbitrary degradation of quality! But for acceptable presentations, a multimedia system must attempt to provide accurate timing and good image quality.

Since some loss of accuracy is inevitable, the goal is not to prevent loss, but instead to keep losses within acceptable bounds. This goal presents a new challenge for multimedia systems: to represent requirements for acceptable presentations and to manage resources to best satisfy those requirements. Of course, what is acceptable depends upon the user and the purpose for a particular presentation.

The requirements for acceptable timing and image (or signal) quality are referred to as *presentation Quality of Service* (QOS) requirements. Throughout this thesis we use the term QOS to refer to the presentation requirements. When we refer to the quality of service requirements for a particular resource we use the resource name as a qualifier, as in "network QOS".

Let a *presentation plan* be the combination of algorithms and resources used to implement a presentation. The same QOS for a presentation can be achieved by presentation plans that have very different resource demands. Figure 1.2 illustrates two alternative plans that produce the same output. The choice of whether to use compression over a network link depends on the relative scarcity of network resources versus processor bandwidth for encoding and decoding. Furthermore, the same resources may be used by presentation plans that have different output quality. For example, the compression in Figure 1.2 could produce high image quality with a low frame rate or vice versa with the same compression ratio and computational requirements. If resources are expensive, then the multimedia system should use no more resources than are needed to satisfy the QOS requirements. We refer to the determination of a presentation plan that satisfies QOS requirements as the *mapping problem.*

It is useful to consider the stored media in a multimedia system as a database and a time-based presentation as the result of a query on that database. Database technology offers many benefits for multimedia applications, such as high-level query languages, concurrency control for document editing, and device and physical data independence. But,

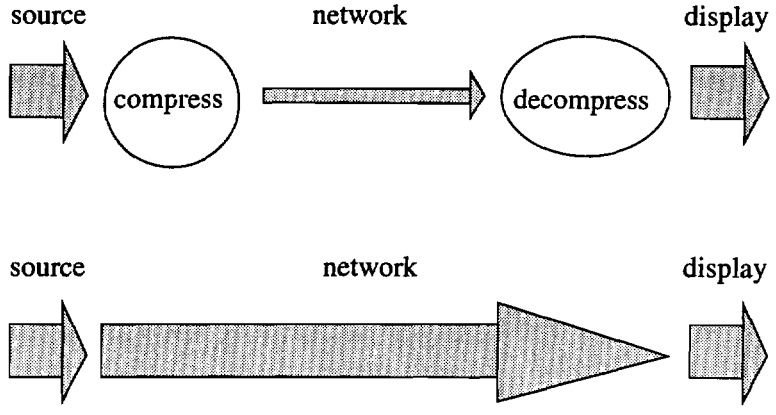

Figure 1.2: Use of compression trades CPU processing for network bandwidth.

current database systems do not adequately support time-based presentations. Relational data manipulation languages have demonstrated the value of letting the application specify *what* is wanted, and letting the database plan *how* to retrieve it. To support time-based presentations, a data manipulation language for a multimedia database should also allow the application to specify *when*, *where*, and *how precisely* the data should be delivered [50]. These constraints on delivery are an example of a QOS-based interface. The specification of QOS requirements is an issue for logical data modelling.

QOS management is both a problem and an opportunity. In the bad-old-days of analog media, the marketplace evolved a relatively small number of media to support the largest market segments. For example, the old analog phone system offered only two-way voice quality communications. The price of a phone call did not diminish when you needed only half the bandwidth. With digital technology, conversion between media formats is simply a matter of software and computer time; and of course computing time has become very cheap through technological advances and growth in the market for computers. Equivalently, digital technology makes it easy to offer the same media at virtually any resolution or sample rate. Just as ad-hoc queries are commonly used to select relevant information out of large relational databases, multimedia requests that specify QOS requirements can be used to browse any type of media without paying the resource costs for best-quality presentation.

Some examples help to illustrate that QOS requirements vary with the application. One of the early titles in the CD-ROM multimedia publishing market is a repackaging of the Beatles' film, "A Hard Day's Night" [77]. This CD-ROM includes the entire film as a compressed digital-video movie. The resolution and playback fidelity is well below the quality of a VHS home video, but the CD-ROM compensates by offering random access and hypermedia links between annotations and film segments. Despite the lower video quality, the compressed video on CD-ROM is better able to meet the needs of the film student than a lossless encoding requiring 100 times the storage volume and bandwidth.

Another example application is mobile computing, where the ability to view multimedia email and other documents over low-bandwidth connections and on low-resolution displays is important. Audio and video can be supported over 56-Kbps phone lines, albeit with reduced picture quality and frame rate [87]. In such applications, the timeliness of access to information may be more important than preservation of the source media quality. Network bandwidth, processing capacity and output device resolution may limit presentation quality, but they do not prevent us from providing real-time multimedia presentations.

As a third example, consider a video database that may be accessed concurrently by multiple users. The video will typically be encoded to support the highest quality playback, yet for tasks such as video editing and visual searches, it is possible to support many more concurrent sessions with lower quality. Despite this tolerance for lower quality, it is important to recognize the point at which poor quality impairs the usefulness of a presentation. An admission test can be invoked with each user request to determine if the request can be satisfied without excessively degrading the service to other users [5, 51, 71].

1.2 The Need for Presentation Quality of Service Specifications

How can a request for multimedia services express its QOS requirements? A multimedia system will need to interpret these requirements in order to schedule resources appropriately. A formal approach for specifying accuracy requirements in database transactions

has been described under the name Epsilon Serializability (ESR) [62]. ESR allows a query to specify an acceptable amount of error in data values so that the DBMS can relax some of the normal data locking requirements. In multimedia systems, the requirements for presentation fidelity are analogous to the requirements for accuracy in the result of a database query. An approach similar to ESR is needed for time-based multimedia presentations to allow relaxation of computation and resource scheduling constraints.

Consider the way that existing multimedia systems handle tradeoffs between QOS and resource use. Video on Demand (VOD) systems typically *guarantee* lossless data transport and strict presentation timing by making conservative resource reservations [47, 6, 84, 61]. This approach is designed to satisfy a specific application where the QOS requirements are high. A multimedia system designed to support a greater range of applications should also provide efficient support for moderate and low-QOS presentations. The Capacity-Based Session Reservation Protocol (CBSRP) allows the specification of discrete QOS classes based on sampling rate and spatial resolution [80]. Although the CBSRP definition of QOS does support dynamic control of resource usage, it does not constrain loss of information through quantization, temporal jitter, or synchronization errors. In the absence of a complete specification of QOS requirements, the implementation of CBSRP makes an ad hoc choice about how accurate the presentation timing must be and how much quantization error is allowed.

As an alternative to guarantees, *adaptive* approaches attempt to provide the best quality, but may degrade some aspects of presentation quality when resources are scarce. The Plateau group at Berkeley has described an adaptive algorithm for network video playback that attempts to decode as many video frames as possible while staying ahead of the display schedule [66]. If the decoder falls behind, some number of frames are skipped in order to reduce the decoder's processing load. Skipping frames is one way to trade presentation quality for resource savings. While this solution allows the video playback to maintain approximate synchronization across a wide range of display platforms, their algorithm for skipping frames is based on ease of implementation rather than a minimal degradation of presentation quality. Cen, et al., have demonstrated that an adaptive algorithm can achieve better perceived quality of MPEG video playback by intelligently

choosing the pattern of dropped frames at the source [13]. Others have shown that video resolution and picture quality can be varied dynamically to save bandwidth without dropping frames [21, 14]. As with a guarantee approach, adaptive algorithm designers are forced to make ad hoc choices regarding which aspect of presentation quality to sacrifice because of the lack of a complete specification of QOS requirements.

To date, researchers have found that presentation-level QOS requirements are difficult to define [58, 19, 33, 7]. Part of the difficulty is due to a confusion between specifying *what* presentation is desired and *how* to achieve that presentation. Presentation QOS requirements derive from *what* functionality is intended. On the other hand, resource QOS requirements derive from *how* the presentation is to be implemented.

What is missing in the literature is a method for specifying presentation QOS. The need for presentation QOS specifications is now well recognized [7], but there has been little discussion of how to address this need. The approach we favor is to divide the problem into two parts: modelling the measurable error in a presentation and empirically determining QOS requirements in terms of this model.

Since the purpose of a multimedia application is to communicate some content to a human user, the QOS requirements necessarily derive from the need to limit noise and other error in the communication channel. An approximate model of human perception can provide a useful tool for presentation QOS management. The degree to which different types of error interfere with the user's ability to understand the content can be determined by studies of human perception [75, 40]. This thesis focuses on how to model the user-perceivable error.

For a QOS specification to be useful for resource management, it must have formal semantics. Formal semantics allow a multimedia system designer to validate whether a particular presentation algorithm will be able to meet the specification. A formal specification is a prerequisite for reliable presentation quality guarantees, even when those guarantees are of a statistical nature [83]. Even if the goal is only to provide the best quality with the available resources, a formal semantics for presentation quality is needed to validate the optimality of a particular presentation. A formal semantics implies that QOS specifications are based on well-defined measurable quantities.

1.3 Scope and Contribution of this Thesis

Our research group intends to build multimedia systems that base their resource management decisions on the quality of service provided to the user. To advance this goal, we have surveyed techniques for QOS management in both research and commercial systems. The insights gained from this research suggested a new approach to specifying presentation QOS. This thesis describes the motivation for and the formal semantics of this new specification technique. To demonstrate the value of formal QOS specifications, the thesis describes the design and implementation of a multimedia player that minimizes the resources used for a presentation while satisfying user-specified QOS requirements. This resource optimization allows more concurrent presentations with QOS guarantees than are possible with an ad hoc approach to QOS management.

This thesis offers a formal QOS specification semantics that can be used to provide presentation guarantees. The key precondition for optimal resource management in multimedia systems is to identify a metric for presentation quality. We describe a three step method for defining such a metric. First, define an ideal presentation. Second, choose an error model that describe the difference between actual and ideal presentations. Third, define a quality estimation function in terms of the error model. We identify a completeness criteria for error models based on the ability to account for all error in a presentation. This method and the completeness criteria distinguish our work from other descriptions of presentation-level QOS parameters.

We also provide a particular example of a content authoring and playback model with formal QOS semantics. Our model contributes orthogonal definitions of *content*, *view*, and *quality* descriptors that together determine presentation QOS requirements. The error model we use to define QOS semantics offers a formal definition of error measures such as *jitter* and *synchronization* error in multimedia presentations. We show that our error model is complete.

A quality metric produced by our method affords a new tool for judging the strengths and limitations of any multimedia system. Not only does it offer a measurement tool, but by identifying the different facets of presentation quality, new opportunities for resource

optimization are more readily identified.

Finally, the prototype implementation of QOS-based resource optimization provides a concrete example of an architecture for translating presentation QOS requirements into resource guarantees. Such translation will be an important part of systems that allow a flexible range of service guarantees since users cannot be expected to understand the resource costs in a large distributed system.

1.4 Outline of the Thesis.

The next chapter surveys a broad range of techniques for QOS management and classifies them by the type of guarantees that they provide. This survey led us to the question of how to specify QOS requirements that would constrain the choice of QOS management techniques. Chapter 3 describes our architectural model for QOS specification, presentation planning and execution. This architecture clarifies the role for formal QOS specifications and provides a reference for comparisons with other systems.

The primary contribution of this thesis is the method and formal semantics for QOS specification given in Chapter 4. A practical implementation of a multimedia player based on formal QOS specifications is described in Chapter 5. Chapter 6 discusses related work in presentation-level QOS specification and Chapter 7 summarizes the major results and conclusions of the thesis.

Chapter 2

Real-Time Presentation of Stored Multimedia Data

We use the term Constrained Latency Storage Access (CLSA) to describe applications that have strict deadlines for the completion of some secondary storage accesses [70]. Examples of such applications are found not only in multimedia applications but also in real-time databases, which must satisfy strict constraints on transaction times [73, 1]. There has been some controversy about whether the timing requirements of multimedia presentations should be considered to be hard or soft real-time [28, 29]. A large body of knowledge exists on how to build hard-real-time systems, but it is generally expensive to assure that no deadline is ever missed [44, 37, 72, 67]. Instead, most existing multimedia systems are susceptible to some data loss and timing error. The majority of the techniques surveyed in this chapter describe ways to reduce the magnitude of these errors.

This chapter describes authoring tools that specify presentation goals and then classifies the well known techniques for meeting these goals in various computing environments. Each technique is characterized by its effect on presentation quality as a function of resource availability. The last section summarizes the results of the survey.

2.1 Identifying Presentation Requirements

Figure 2.1 shows an abstract model for authoring and playback in multimedia systems. In the first step, a presentation *author* creates a descriptor for some multimedia *content.* A *content descriptor* defines a multimedia document in terms of basic media types, layout, synchronization, and links for navigation between documents. A presentation algorithm

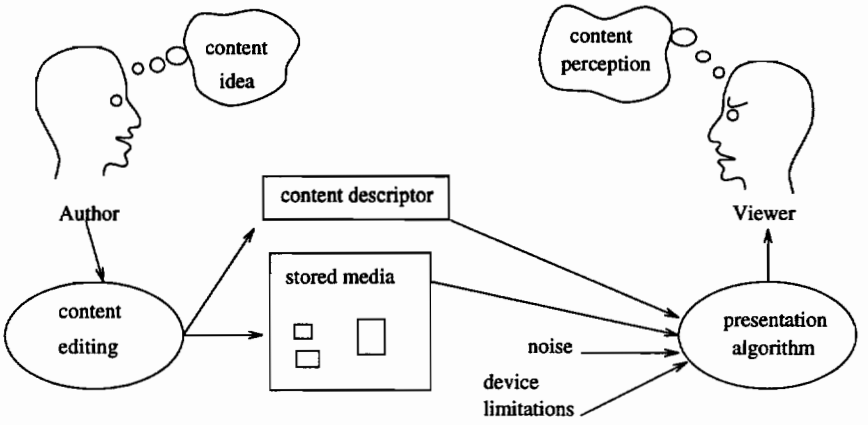

Figure 2.1: Authoring multimedia presentations.

reproduces content from a descriptor, but the presentation may be degraded by device limitations or noise in the computing environment. A viewer receives the author's intended message by filtering the noise from the perceived content.

The following tools offer varying amounts of support for presentation authoring and playback. All produce some form of content descriptor. But what is the correct way to interpret a given content descriptor during a presentation? The discussion of each tool considers this question and describes the implicit presentation semantics.

2.1.1 Continuous Media

Audio and video recordings of natural phenomena have a natural presentation semantics as well: the goal is to reproduce the original phenomena. To this end, digital recording tools produce a minimal content descriptor to accompany the encoded data that effectively specifies a normal presentation.

Audio and video are considered *continuous media* data types because they represent continuous natural phenomena. The amplitude of an audio signal varies continuously with time and the color value in a video varies continuously in both time and two-dimensional space. Digital recordings are created by periodically sampling an analog signal. The sampling process is characterized by the sampling frequency and the sample depth or number of bits used to represent a sample. For example, compact disk audio is sampled

at 44 Khz with 16-bits (65536 values) per sample. Digital television for studio work is sampled at 13.5 Mhz with 8-bit samples for the luminance signal [59].

A content descriptor for continuous media playback can be as simple as a file name and a few parameters that describe how the data was recorded. For example, Sun audio files contain a header with sample rate and format information [76]. For best reproduction of the original audio signal, the digital samples should be written to a digital-to-analog converter at the same rate and format that they were recorded. This timing requirement is typically met by periodic scheduling of a low-latency output task. In addition, synchronization between audio and video tracks that were recorded together should be preserved during playback.

How accurate does the playback really need to be? Small amounts of timing error in a presentation have an effect similar to that of signal noise as illustrated in Figure 1.1. Resampling and conversion of the data stream for a different audio device can also introduce perceived noise. But a digital recording already has a base level of *quantization noise* from the use of discrete values to represent analog samples. Small timing and data conversion errors are insignificant so long as they are masked by the quantization noise, but larger errors may be tolerable for some types of content and applications. Today's commercial tools attempt to provide the best playback quality possible for the recorded data and do not incorporate any other notion of QOS requirements.

2.1.2 Muse

The Athena Muse authoring environment offers four distinct representational approaches for specifying interactive multimedia learning environments: directed graphs, multidimensional spatial frameworks, declarative constraints, and a procedural language [32]. The directed graphs are useful for hypermedia style navigation. The spatial frameworks allow both specification of image display positions and placement of objects on a presentation timeline. Figure 2.2 illustrates the use of a timeline to specify synchronization in Muse. A timeline can be used to synchronize many objects including still images, text, video and audio segments. Declarative constraints, in the Muse system, are limited to bi-directional

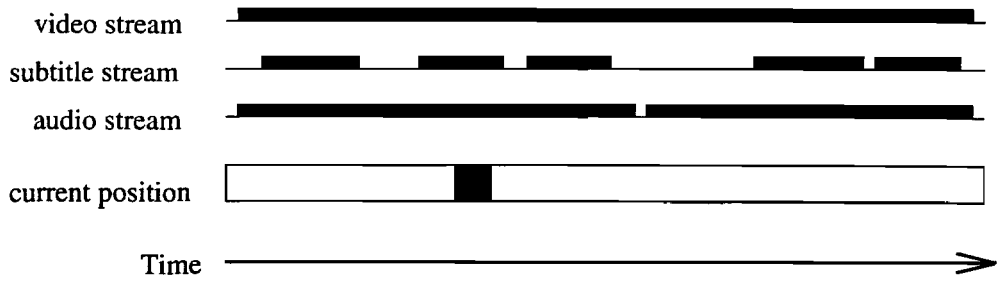

Figure 2.2: Use of a timeline for synchronization in Muse.

equality relations. For example, the scroll bar shown in Figure 2.2 is constrained to represent the current position in the display of a timeline. When the scroll bar is moved, the view from the timeline is updated and vice versa. Finally, the procedural language allows arbitrary computations to be embedded in the production.

The specification of temporal and spatial layout in Muse constitutes a set of presentation goals that the presentation engine should attempt to meet. Unlike the continuous media data types described above, the synthetic timing constraints of Muse composition are generally not periodic. The descriptor for a presentation must explicitly store each object presentation time specified on the timeline and the presentation engine must initiate presentation of the objects according to this schedule. Muse has an informal presentation semantics and has been successfully used for authoring educational materials for presentation on a modified Athena workstation.

2.1.3 Object Composition Petri Nets

Little and Ghafoor have described an interval-based descriptor for multimedia presentations called an *Object Composition Petri Net* (OCPN) [43]. Each media object is assigned to an output device and has a known display duration. An OCPN is constructed by specifying the temporal relation between two objects using one of the seven interval relations shown in Figure 2.3. Some relations require a delay parameter as indicated by the small arrows. Every OCPN can be viewed as an object with known duration for recursive composition.

OCPNs have an operational semantics that guarantees that no object is displayed

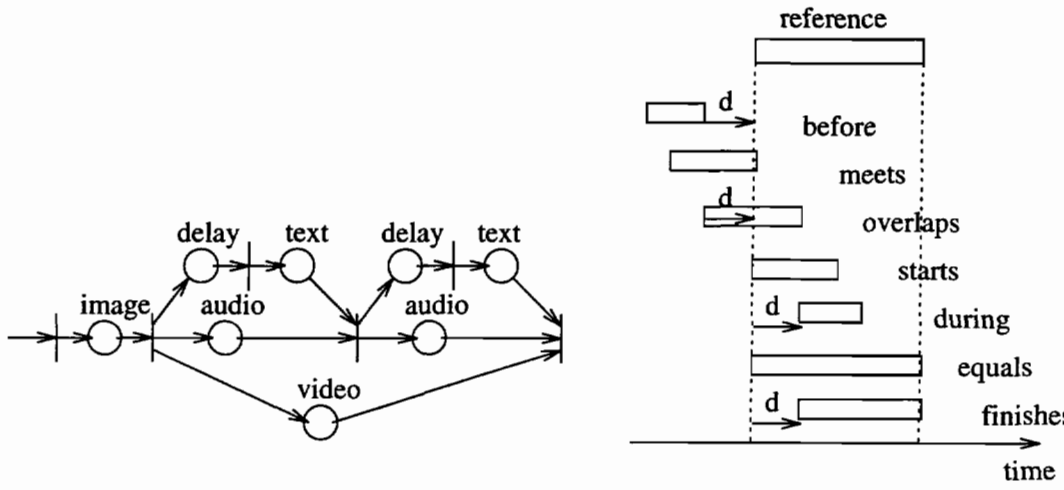

Figure 2.3: Interval relations in an Object Composition Petri Net.

until the previous transition in the petri net has been enabled. For example, when the image display process in Figure 2.3 completes, the subsequent transition (vertical bar) is enabled and the associated video, audio, and delay processes can begin. However, there is no guarantee that enabled processes will begin immediately or that they will execute for precisely the specified duration. As with authoring in the Muse system, OCPNs do not specify presentation accuracy. Little and Ghafoor have suggested a partial list of network QOS parameters for communication of multimedia objects [42]. In Chapter 4, we offer a formal definition of some of these parameters and show where they fit into a complete model for presentation QOS specification.

2.1.4 MAEstro

MAEstro is a set of UNIX-based tools for authoring multimedia documents [19]. It provides a timeline editor with a direct manipulation interface for synchronization of media segments. The appearance of the timeline editor is similar to Figure 2.2, but the MAEstro authoring model is more restrictive than Muse. For example, a presentation has only a single track for each media type so that it is not possible to specify two concurrent audio segments. Media segments are edited and played by a media editor that is registered for each media type. During a presentation, the timeline editor detects the start time for each

segment and sends a message to the appropriate editor to display the appropriate data.

MAEstro was designed to support network-based multimedia by delegating media handling responsibilities to the distributed media editors. As a concession to the difficulty of synchronizing distributed multimedia streams, MAEstro does not guarantee that media editors stay synchronized with each other. Instead, it has *rising-edge* synchronization which means that only the start time for each media segment is controlled by the timeline editor. In practice though, even the initiation of media playback is subject to delays for message passing, process scheduling and storage access. MAEstro is used for authoring multimedia documents with coarse-grained synchronization on UNIX platforms.

2.1.5 Algebraic Video

Weiss, et al., have described an informal semantics for an algebra of video composition operators [85]. An algebraic video expression can represent a segment of raw video or a composition of other algebraic video expressions. The algebraic operators are shown in Table 2.1. This video algebra allows users to specify presentations through content-based queries and simple composition operations. Content is described with text annotations using the `description` operator. Since any video expression may be annotated, and video expressions may share common video segments, annotation properties may overlap. The union and intersection operators support composition and decomposition, respectively, of overlapping video segments.

The output operators for algebraic video expressions specify a multimedia presentation, but do not describe presentation quality requirements. As with other authoring tools, the algebraic expressions describe presentation goals without providing constraints for an implementation.

2.1.6 MHEG

ISO's Multimedia Hypermedia Experts Group has defined the MHEG encoding standard for storage, exchange and execution of multimedia presentations [52]. The encoding supports spatial layout and synchronization of common media elements and also supports

Creation	
`create` *name begin end*	create a presentation from named video
`delay` *time*	create a presentation with empty footage
Composition	
$E_1 \circ E_2$	concatenation of E_1 followed by E_2
$E_1 \bigcup E_2$	E_1 followed by E_2, no duplication of common footage
$E_1 \bigcap E_2$	intersection with only common footage of E_1 and E_2
$E_1 - E_2$	difference with only footage of E_1 that is not in E_2
$E_1 \parallel E_2$	E_1 and E_2 start simultaneously and play concurrently
$E_1 \wr\wr E_2$	E_1 and E_2 play concurrently and end simultaneously
$(test)?\ E_1 : E_2 : ... : E_k$	E_i is played if *test* evaluates to i
`loop` E_1 *time*	repetition of E_1 for duration *time*
`stretch` E_1 *factor*	stretch duration of E_1 by *factor*
`limit` E_1 *time*	limit duration of E_1 to *time*
`transition` E_1 E_2 *type t*	transition effect between E_1 and E_2 for duration t
`contains` E_1 *query*	components of E_1 that match *query*
Output	
`window` E_1 *rectangle priority*	display E_1 with *priority* in *rectangle*
`audio` E_1 *channel f priority*	output E_1 to *channel* with *priority*
Description	
`description` E_1 *content*	annotate E_1 with description of *content*
`hide-content` E_1	hide the content annotations of E_1

Table 2.1: Algebraic video operations.

user interaction through hypermedia links, menu selections and data entry. The fundamental building blocks of an MHEG presentation are *content objects* that represent an atomic piece of a particular media type. Layout and synchronization of content objects is described with *virtual coordinates* and *virtual views* that must be mapped to real coordinates during presentation. State transitions in the playout of content objects, e.g. a completion event, can be used to trigger other presentation actions. The MHEG standard is rich enough to represent presentations created by many diverse authoring tools, including Muse, OCPNs, and MAEstro.

2.1.7 Discussion

Early multimedia systems, such as Muse and MAEstro, are designed to use the same presentation engine for both authoring and playback. The authors can know the limitations of the presentation computing environment and can tailor content descriptors appropriately [19]. A viewer that uses the same presentation engine and the same or similar computing environment is likely to perceive the presentation as the author intended. On the other hand, the MHEG standard is predicated on the exchange of presentation descriptors between heterogeneous computing systems. But distribution and heterogeneity make it difficult for the author to tailor the content for playback. When a viewer attempts to view a presentation over a heavily loaded network or on a machine with lower performance than the authoring platform, there is a large probability that the presentation will not be acceptable. Even when the presentation engine can adapt the presentation quality to the available resources, the engine has no information about how to balance the loss of playback quality between different aspects such as spatial resolution and frame rate.

Current authoring and playback tools specify only presentation goals and not presentation QOS. The presentation engines take an ad hoc approach to managing QOS tradeoffs. These tradeoffs are described in detail in the next section.

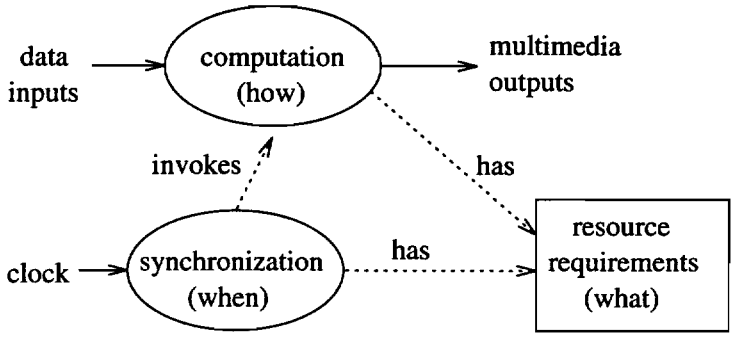

Figure 2.4: Abstract model of a real-time presentation.

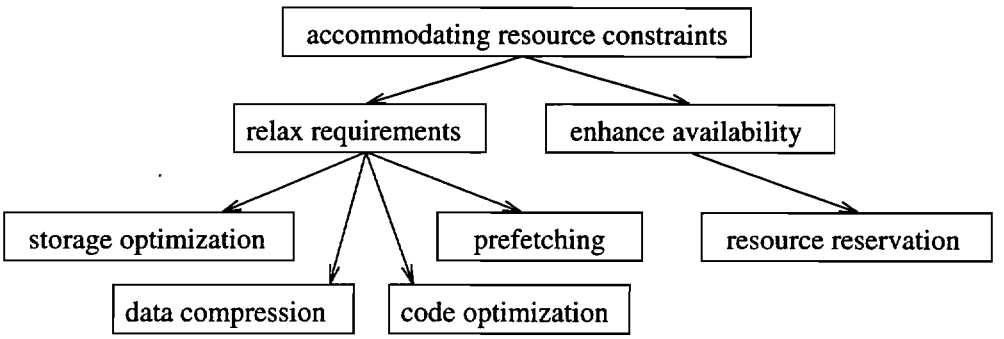

Figure 2.5: Overview of presentation techniques.

2.2 Presentation Techniques

Figure 2.4 shows an overview of the problems in multimedia system design. A presentation algorithm must solve three problems: how to compute multimedia outputs, when to compute them, and what system resources to use. The next sections survey how these problems are solved in existing multimedia systems. Section 2.2.1 discusses basic algorithms for computing multimedia outputs from stored data. Section 2.2.2 describes common algorithms for synchronization, i.e., control of when outputs occur. Resource constraints can be met by either reducing presentation resource requirements or increasing resource availability as suggested in Figure 2.5. Approaches for relaxing presentation resource requirements are described in Sections 2.2.3, 2.2.4, 2.2.5, and 2.2.6. Resource reservation techniques to assure resource availability are described in Section 2.2.7.

2.2.1 Output Computation

An important task in planning a presentation is to identify data sources and sinks and to determine what computation is needed to connect them. For example, presentation of an MPEG video requires reading the file, decoding the video stream, converting each frame to the color map and dimensions of the output window, and copying the data to the output framebuffer. These steps are typically organized as a pipeline of (possibly distributed) processes and some of the processing may be performed by specialized hardware. In any case, the choice of a computation algorithm directly determines the values that are output during the presentation. The computational steps in a presentation can be can classified in the following categories:

- storage access
- transport of data
- compression and decompression
- manipulation of content
- output device access

As discussed in the introduction, it may not be possible to represent the intended output values perfectly. For example, a black and white display cannot reproduce a color image, and a low resolution display cannot reproduce finely detailed images. Instead of a single correct computation for connecting sources to sinks, there are many possible computations that approximate the intended output.

The conversion from source encoding to an output representation can affect many aspects of presentation quality. In audio presentations, converting to a smaller number of bits-per-sample introduces more noise. Resampling an audio stream at a lower rate results in the loss of high audio frequencies. Even the volume of an audio presentation can be affected by data processing steps. For images – such as text displays, still pictures, graphics and video – the output computation affects color fidelity, brightness, contrast, resolution, image noise, visual artifacts, and overall image proportions. The presentation

planner must choose among the approximate computations, one which meets QOS goals with available resources. The following sections discuss techniques that expand the range of options.

2.2.2 Synchronization

If acceptable computations of presentation outputs have been identified, then the question is when to execute those computations. The schedule for when output events *should* occur is defined both by the authoring tools and by interactive events. Schedules may be classified in three categories:

- periodic
- scripted
- event-driven

A *periodic* schedule specifies a constant time period between output events. If the schedule is not periodic, but the output times are known in advance, we call it a *scripted* schedule. Finally, if output events are triggered by user interaction or other external events, we call the schedule *event-driven.* A presentation schedule may be constructed as a hierarchy of periodic, scripted and event-driven schedules. For example, an information kiosk might present video segments according to a scripted schedule while each single video segment has a periodic schedule for frame output events.

Event-driven schedules can be realized with an event-loop algorithm that invokes a handler for each external event. The handler completes its work quickly, possibly by forking another process that may run concurrent with the event loop. Typically, the initiation of a presentation in response to user interaction should occur as soon as possible, while the remainder of the presentation is defined by a periodic or scripted schedule. MHEG and other interactive multimedia documents support event-driven schedules, but it may be easy to derive a new scripted schedule from the MHEG specification in response to each user interaction.

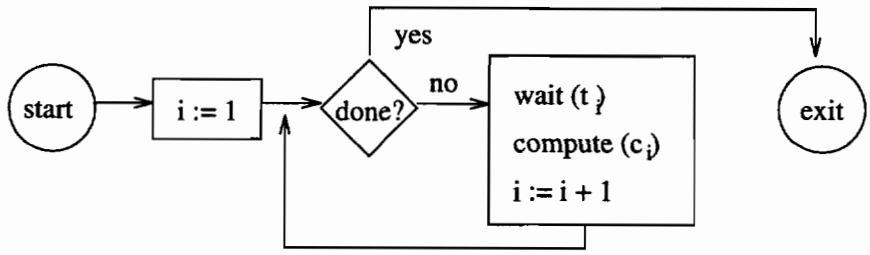

Figure 2.6: Clock-driven synchronization.

Figure 2.6 illustrates a simple *clock-driven* synchronization algorithm. The presentation is described as a sequence of pairs (c_i, t_i) where c_i is an output computation and t_i is the ideal time for the computation to occur. For each output computation, the algorithm waits for the clock to reach the ideal time before running the output computation. For periodic events, this loop can be implemented with a periodic timer interrupt. This algorithm provides two guarantees: that outputs are generated in order and that no output happens early. Unfortunately, it does not guarantee that any output is generated!

Despite these minimal guarantees, the clock-driven algorithm is perfectly adequate for presentations where the latency for output computations is easily bounded. MAEstro and many other multimedia systems take this approach in environments where the time required to access and process data for each output is negligible relative to the time between events [19]. But what should be done if the access and rendering time distort the presentation timing as shown in Figure 2.7? Not only does the title screen persist too long, but if the image is displayed after the presentation time for the end slide it gets immediately overwritten!

If process latencies are predictable, one way to correct the output timing is to modify the schedule of output times, compensating each by the anticipated latency [19]. If the latencies are variable, but bounded, it may be possible to hide the latency by prefetching from storage as discussed in Section 2.2.6.

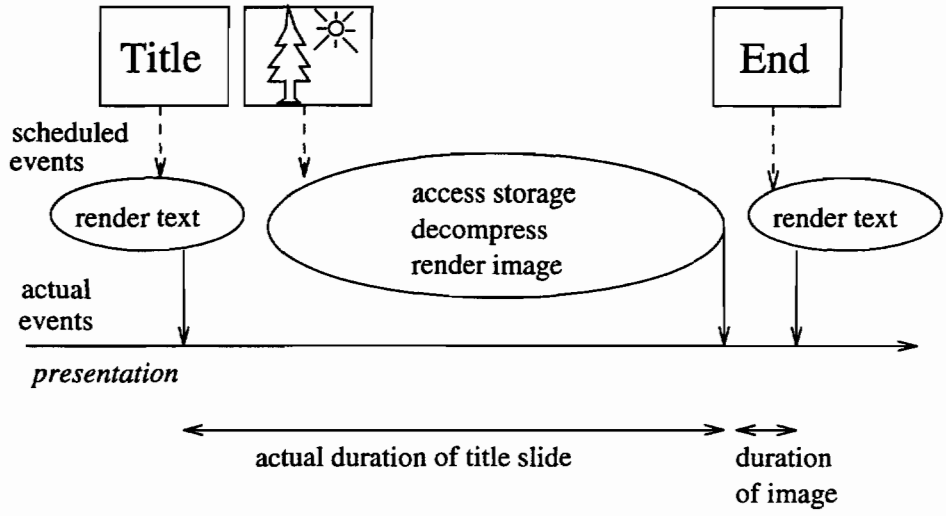

Figure 2.7: Output delay from process latencies.

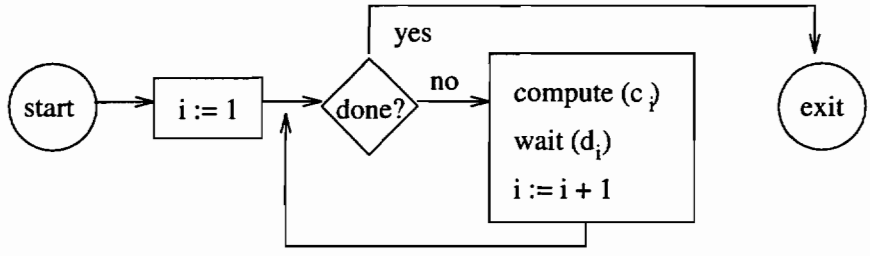

Figure 2.8: Duration-driven synchronization.

An alternative to the clock-driven algorithm is to wait explicitly for the intended duration of a presentation before triggering subsequent actions. Figure 2.8 shows a *duration-driven* synchronization algorithm that executes computation c_i and then waits for a duration d_i before continuing. A computation may fork a child process to execute a sub-presentation recursively and in parallel, but it must then wait for the child to complete before it completes itself. This type of algorithm is described for executing an OCPN [43]. Duration-driven synchronization guarantees that the output of every computation c_i is not overwritten early by the next output of the presentation thread. Since other presentation actions may be occurring in parallel, this synchronization algorithm only preserves a partial order of presentation events.

Duration-driven scheduling may be combined with clock-driven scheduling and event-driven scheduling. For example, most multimedia players handle user inputs that start, stop, reposition and change the rate of presentation. The event-handling loop interprets the user input and recomputes parameters for a clock or duration driven algorithm. In a duration-driven algorithm, some presentation computations may be implemented by a clock-driven process.

So far, we have assumed that the synchronization algorithms affect only the timing of computations and not the results. If it is more important to complete computations on time than to complete all computations, then a clock-driven synchronization algorithm may be used to skip computations. For example, instead of waiting for the next presentation time in sequence, the algorithm might skip all but the last computation whose time has past. This guarantees that every computation was current when it was initiated. If all computation latencies are bounded by a duration d then this algorithm guarantees that every computation completed is not more than $2d$ late. we call this type of algorithm *strict clock-driven* synchronization because it allows a strict limit on timing error with respect to the clock.

If some computations must be skipped, it is desirable to have more control over which to omit. In continuous media presentations bandwidth limitations frequently limit the sample rate that can be transmitted and decoded. Several researchers have described the use of *software-feedback* techniques to determine the available bandwidth and to adapt the scheduled sample rate accordingly [66, 13].

In a distributed system, there is no single clock that can be used to control synchronization. Our ability to synchronize processes executing on different machines is limited by the communication latency between the machines, and by the variation in clock rates. Clock rates can vary in the parts-per-million range with normal temperature changes [55]. These uncertainties make it difficult to assure that media streams from different sources begin at the same time and proceed at the same rate. One solution is to use a network clock-synchronization protocol, such as NTP, to synchronize distributed clocks within a few milliseconds [55]. However, in most presentations of stored data the output devices are attached to a single client machine. In that case, it is a simple matter to control

synchronization via a local clock on the client. Feedback algorithms have been described that coordinate timing between servers and the client [13]. Section 2.2.6 describes this type of coordination in more detail.

2.2.3 Storage Optimizations

Video and audio data can require large amounts of storage space. For many multimedia applications, the data is immutable and storage is optimized for read access. Many different storage architectures have been used for multimedia data, including arrays of fast magnetic disks and optical disk jukeboxes [9, 15]. The variations in throughput and latency characteristics of such systems is very large. CD-ROM drives with transfer rates of 1.2 MBytes per second and seek times on the order of 1 second are in common use on personal computer systems [82]. A throughput bottleneck in storage forces either a slow-down of the presentation or information loss through skipped data. Large storage access latency causes delay and jitter in a presentation.

Storage optimizations are targeted at reducing latency and increasing throughput. Storage latency is a function not only of physical device characteristics, but also the policies that dictate the placement of data and when it is moved. Common device characteristics to be considered include RAM access speed, bus contention delays, disk controller overhead, seek time, rotational delay, transfer rate, mounting time for off-line disks in a jukebox, and network communication delays. The policies of the storage system are evident in data layout and caching, the handling of resource contention (including CPU) in multitasking environments, decompression and other processing requirements. Data layout optimizations are discussed in this section. LRU and other common disk caching policies are ineffective for multimedia presentations since the media streams are accessed serially and the datasets are frequently too large to fit in main-memory. Instead, we discuss prefetching techniques for hiding latency in Section 2.2.6. Compression techniques for reducing storage bandwidth requirements are described in Section 2.2.5. Resource reservations to avoid contention are discussed in Section 2.2.7.

Data Layout

A careful layout of data in storage is an important part of many continuous media storage systems [47, 6]. The goal of data layout is to minimize seeks and rotational latency between reads. Seek time can be minimized by storing the data stream in contiguous storage locations. Rotational latency can be minimized by dividing the data into disk-transfer units and writing these units to the next available disk sector at the same rate that they will be read. Yu, et al. have described an optimal placement of audio data on disk that accounts for rotational latency [88].

A concurrent presentation of two or more streams, e.g. audio and video tracks, requires interleaved access to data for each stream. If the streams are to be played out synchronously they can be multiplexed and stored as a single stream [63]. During playback, a multiplexed stream must be de-multiplexed before the data is written to separate output devices.

If concurrent streams are not multiplexed, then the disk head must be scheduled to interleave reads from each stream. Figure 2.9 illustrates a cyclic disk schedule that reads two sectors for one stream, seeks to a second stream to read one sector, then returns to the first. This time sharing creates two problems for real-time disk access: increased jitter in the stream access and decreased disk bandwidth due to the overhead for seeks. While the disk is servicing one stream, data transfers for the other stream are delayed. When the disk scheduler switches streams it incurs the cost of seeking to the other stream. The jitter can be hidden from the presentation by introducing a buffer between the disk server and the display process. The disk scheduling and buffering requirements for continuous media data have been described by Gemmell and others [25, 2]. Some of these results are described in Section 2.2.6.

Data layout is further complicated by applications that read only a portion of a continuous media stream, e.g., only the low-frequency components of a compressed video. One approach is to split a single media stream into base-layer and enhancement-layers [14]. For low-resolution access, an application need only read the base-layer stream. For best resolution, an application must read the base-layer and enhancement-layer streams and

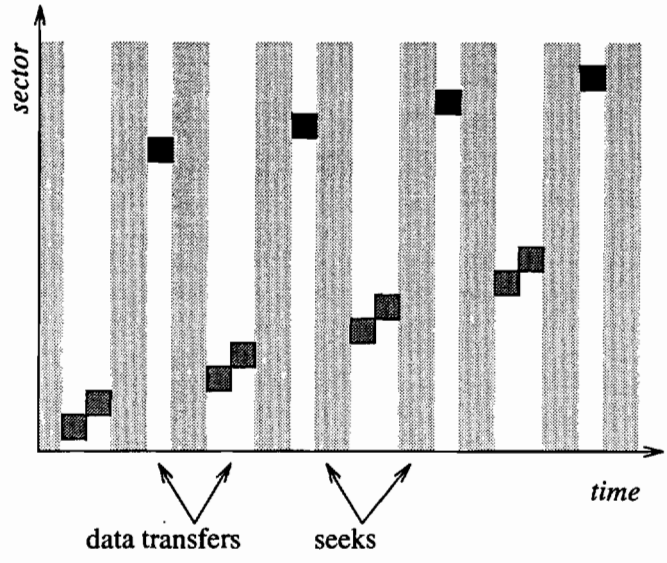

Figure 2.9: Interleaved disk scheduling for two streams.

combine them.

Disk Striping

Disk striping is a common technique for increasing disk bandwidth. A data stream is segmented by time-slicing and the segments are written in round-robin order to an array of N disks. When reading the data, N slices can be transferred in parallel, achieving a near-linear speedup of disk bandwidth [9]. Video on Demand (VOD) systems have been built using disk striping to provide bandwidth guarantees for many concurrent users [47, 6, 84]. However, to share bandwidth between multiple playback streams requires interleaved service, just as for a non-striped disk.

Figure 2.10(a) illustrates some problems associated with disk striping. The data for stream A is striped over only 3 out of 8 disks. If the stream is read at 3 times the bandwidth of a single disk, disks 1-3 will be fully utilized while disks 4-8 will be available for other users. A request for stream B must wait until A is finished if any of its data is located on the same disks. Figure 2.10(b) shows how the same requests can be serviced with less delay using staggered striping [6]. For each consecutive time slice, staggered striping increments the indices of the disks used to stripe data so that the full stream is

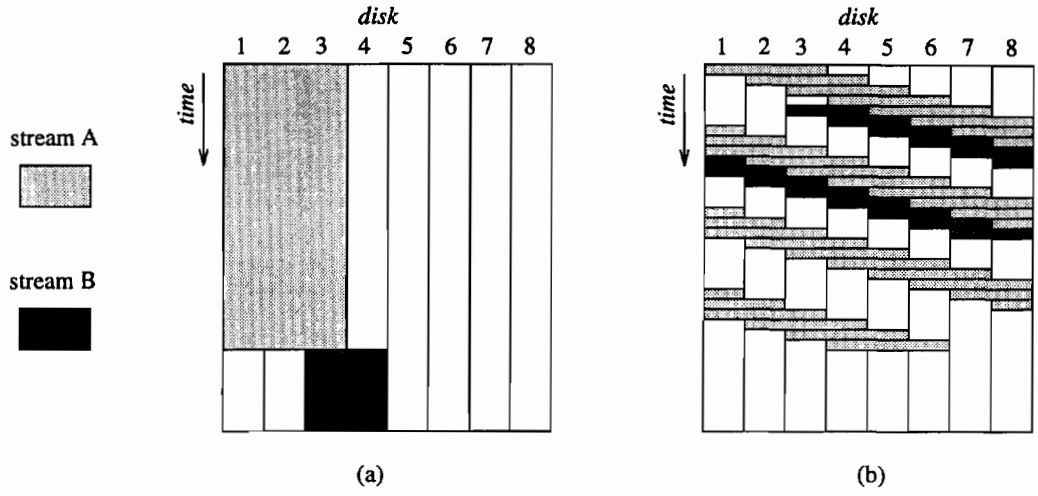

Figure 2.10: Simple striping versus staggered striping.

distributed across all disks. This data layout allows the same maximum bandwidth for each stream without tying up any one disk for the entire playout duration.

The maximum bandwidth for a stream is determined by the number of disks that are accessed in parallel for each time slice. Reading a stream at a lower rate leaves the disks underutilized since the bandwidth requirements of another stream is unlikely to match the particular pattern of idle disk time slices. This inflexibility can be solved by assigning stream segments to disks randomly instead of by striping. A random assignment of stream segments to disks can balance the load among disks nearly as well as a striped layout, but makes it more difficult to provide bandwidth guarantees [54].

2.2.4 Process Optimizations

Software transport, decoding and processing of video data is often the bottleneck in multimedia systems. It is often possible to reduce the processing time by *lossless* and *lossy* process optimizations. An optimization is *lossless* if it preserves information in the data stream, and *lossy* if some information is lost. Lossless optimizations include hand-tuning of machine code and elimination of unnecessary data copying [45, 20]. Specializations are optimizations that depend on special knowledge of an application. For example, if it is

known that a video presentation window cannot be moved or obscured then the presentation can bypass the window system and write directly to the display frame buffer. Lossless optimizations do not degrade the presentation quality.

Lossy optimizations, in contrast, do affect presentation quality. For example, frame dropping is a common technique that reduces the amount of CPU time used at the expense of the perceived frame rate. Ideally, frames are dropped at a regular rate to minimize the perceived degradation of quality and as early as possible in the pipeline to minimize handling costs [13]. Frame dropping is one example of subsampling a data stream in time. Other examples include subsampling an audio stream and spatial subsampling of an image [18]. Another lossy optimization is the use of a less expensive and lower-quality dithering algorithm. For example, an error-diffusion dither generally yields the best image quality for pixel-depth reductions, but a simple truncation of pixel values to the required depth is much faster [81].

2.2.5 Data Compression

Data compression is used to reduce the storage and transport costs of multimedia data, but these savings come at the expense of increased processing requirements for encoding and decoding. Typically, it is the decoding requirements that are of concern for stored data, since encoding can be performed offline. The benefits of the compressed data representation must outweigh the costs of decompression when the data is needed. This condition holds when storage is scarce and when available disk or network bandwidth is inadequate for the uncompressed data stream. The bandwidth requirements of an uncompressed digitized NTSC video stream are conservatively estimated at 80 Mbps, which currently exceeds the capacity of most file systems, network links, and even display interfaces.

Lossless compression techniques, such as run-length encoding, differential encoding, and entropy encoding, remove redundant information from a data stream without loss of information. Lossless techniques may only achieve a 2:1 compression ratio with continuous media data, but since continuous media data can tolerate some loss of information, *lossy* compression algorithms have been created that can achieve much higher compression ratios by throwing away redundant and perceptually less important information. Lossy

compression techniques include truncation, subsampling, motion compensation and the discrete cosine transform (DCT). The Motion Picture Experts Group (MPEG) MPEG-1 compression standard uses motion compensation and the DCT in combination with lossless compression techniques to achieve compression ratios on the order of 70:1 [49, 23, 74].

As with lossy code optimizations, lossy compression techniques are designed to minimize the perceived degradation of quality. MPEG-1 compression has been optimized to yield VHS quality video at CD-ROM data rates. However, the amount of compression that is possible without unacceptable loss of image quality depends on the complexity of images and motion in the original video. The quantization of DCT coefficients limits the amount of high-frequency information that can be encoded and produces visible artifacts around sharp edges where such information is needed. The motion compensation algorithm cannot find good matches for every block when the original video contains complex action as in close shots of a basketball game. With poor motion compensation, the difference encoding has a large amount of high frequency information and again, artifacts appear in the decoded video images. The MPEG standard for encoding allows control of the resolution, the amount of quantization, the amount of frame difference encoding, and the search algorithm for motion compensation. As these parameters are used to increase compression, the loss of quality becomes greater. we have found that it is possible to produce useful video with 30 frames per second at a compression ratio of 500:1, but that the loss of resolution and other artifacts are annoying.

Compression also makes a data stream more vulnerable to the effects of packet losses in network transmission. For example, an MPEG video stream is typically encoded with bi-directionally predicted or "B" frames that require both a previous and a subsequent frame to be decoded first as a reference. If a packet loss causes an error in the decoding of either of these two frames, then the error will be propagated to (or prevent the decoding of) all the intervening B frames.

Scalable video resolution may become a common requirement in future applications. MPEG video streams may be filtered in real-time to remove high-frequency coefficients, producing a lower-bandwidth and lower-resolution video stream at the expense of some additional processing at the server [21]. Stanford University and Sun Microsystems have

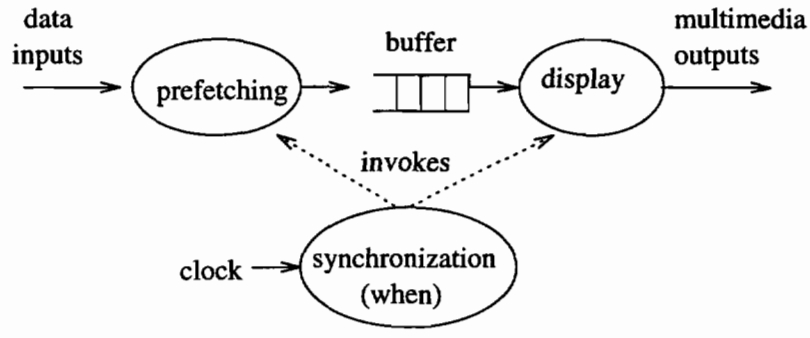

Figure 2.11: Abstract model for prefetching.

designed a VOD system that supports multi-resolution access to encoded video [14]. The MPEG-2 video encoding allows both HDTV and NTSC resolution images to be decoded from a single stream.

2.2.6 Prefetching

Prefetching is a common technique for hiding storage latency when the access pattern is known in advance. A prefetching process reads data from secondary storage into main memory before the data is actually needed, as shown in Figure 2.11. When the application tries to access the data, it is found in main memory and storage access delays are avoided. In an earlier work we have described the problem of constrained-latency storage access (CLSA) and identified prefetching as the generic class for solutions [70].

Prefetching reduces presentation delay and reduces jitter (variation in delay) by allowing the display process to perform a shorter computation at the scheduled output time. Prefetching may also allow higher overall throughput, since computation may be overlapped with concurrent disk access. However, prefetching does incur some computational overhead for scheduling concurrent processes.

Figure 2.11 suggests that prefetching and display processes can be viewed as a pipeline. Many continuous media players are organized as a pipeline of storage access, network transport, decoding and display processes [48, 4, 66, 13, 39]. Let us generalize the idea of prefetching to describe any decoupling of a computation into concurrent producer and consumer processes. By this definition, an interrupt-driven process that reads from a

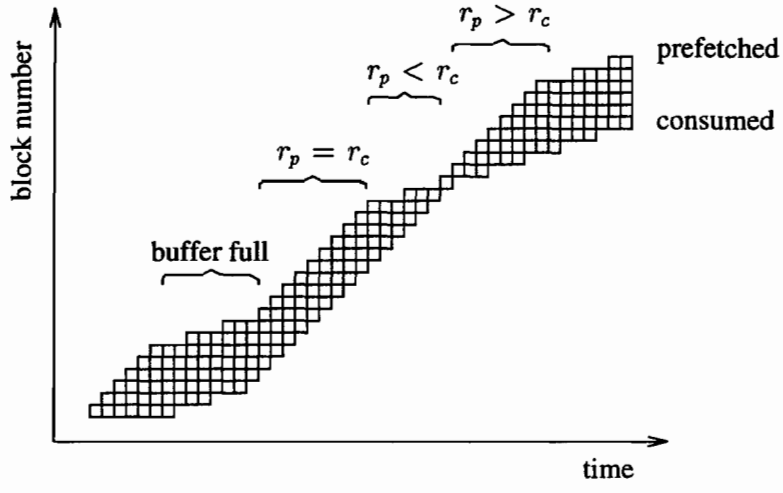

Figure 2.12: Greedy prefetching.

live-video camera and sends the data to a remote display process can be described as prefetching data for the network process. The network process, in turn, is prefetching for the display process.

Prefetching flow control is needed to avoid overwriting data in the buffer or starving the consumer. A wide range of prefetching techniques exist with differing methods for flow control between producer and consumer. Examples are described below, with a citation of where the technique is used and a discussion of the advantages and disadvantages.

Greedy Prefetching

We use the term *greedy prefetching* to refer to a process that outputs a stream of data to a queue as long as the queue is not full. A process that writes to a UNIX pipe is viewed as a greedy prefetching process because it does not wait for a read on the pipe unless the queue for the pipe is full [65]. The AudioFile system uses pipe semantics to connect an audio playback application to a device server using a fixed sized queue [39].

Greedy prefetching uses a simple back-pressure technique to synchronize a fast producer with slower consumer. So long as the queue is non-empty, the consumer is insulated from delay and jitter in the prefetching process. But what if the consumer is faster over

some small interval of time? Suppose that data are passed from the producer to the consumer in 1 block units. Let r_c be the rate that blocks are consumed by application demand and r_p be the rate that blocks are produced by prefetching. The rate r_c may vary with time. The rate r_p will be zero when the queue is full and will be limited by scheduling delays and by the latency for each fetch when there is free space in the queue. Figure 2.12 illustrates greedy prefetching with a queue of size 6. When $r_c < r_p$, the queue will fill up or remain full as shown. When $r_c < r_p$, the queue will empty out. To avoid starving the consumer, the greedy algorithm must allocate and fill enough buffers for the queue to be able to satisfy demand during the worst-case interval in which the consumption exceeds prefetching. For every interval (t_1, t_2) this condition can be expressed as follows:

$$n > \int_{t_1}^{t_2} (r_c - r_p) dt$$

Note that if c greatly exceeds p over some interval, this condition may require a very large queue.

The system designer must understand the application and the storage performance well enough to specify how many buffers are needed to allow the prefetch process to get a headstart on the consumer.

Consider the case where a prefetch process reads continuous media data from disk and a display process consumes this data at a constant rate. If data is read a sector at a time from disk, a minimum of two sector-sized storage buffers are needed to allow the consumer to read data from one as the greedy prefetching algorithm copies data into the other. Let the constant demand rate be r_c bytes/second, the disk transfer rate be r_t bytes/second, the size of a disk sector be s_s bytes and the smallest unit of introduced delay (e.g. rotational delay) be d_{min} seconds. Let the data be clustered in segments of i contiguous sectors on disk and the maximum bound on seek time between consecutive segments be d_{max} seconds. Gemmell and Christodoulakis show that any sustainable prefetching algorithm must use a segment size of at least

$$i \geq \left\lceil \frac{r_c d_{max}}{1 - r_c/r_t} \right\rceil \frac{1}{s_s}$$

disk sectors and must allocate at least

$$n \geq \left\lceil \frac{r_c}{s_s} \left(d_{max} + d_{min} + \frac{s_s}{r_t} \right) \right\rceil$$

sector-sized buffers [25].

One of the advantages of greedy prefetching is that producers are automatically blocked when consumers block so that no data is lost. However, this feature can also be a disadvantage when an exception causes the delay of a consumer process. With greedy prefetching, when the consumer stalls, the entire pipeline is held up. Any attempt to resynchronize the presentation by skipping data will be delayed by the full latency of the pipeline.

Rate-Based Prefetching

Rate-based prefetching separates prefetch scheduling from the queue space availability. Ideally, the prefetcher produces data at the same rate that it is consumed so that the queue is never empty or full. In practice, the consumer must either adapt to the prefetch rate or use feedback to adjust the rate of the prefetch process. The ACME continuous media I/O server supports *connection-driven* rate control, where a media output process adapts its rate of consumption to keep pace with a real-time file access process [4]. Rate-based feedback techniques allow distributed prefetching processes to be synchronized with consumer processes [64, 66, 13, 14].

A rate-based approach is appropriate when the prefetched data becomes obsolete at a predictable rate, regardless of whether the consumer process has read it. Obsolete data may be overwritten by the prefetching process without waiting for the consumer. Rate-based protocols can achieve flow control with less overhead than a greedy prefetching approach [64]. For minimal latency communications, unreliable messaging protocols such as UDP may be used since there is not time to retransmit lost messages [60]. Variations in prefetch and transport delays will produce jitter in the arrival of data in the queue. The average amount of data in the queue should be sufficient to avoid starvation when packets are delayed. The average amount of free space in the queue should be sufficient to accommodate packets that arrive early without loss.

Scheduled Prefetching

Scheduled prefetching uses an explicit *prefetch schedule* of times to initiate the retrieval of each object in a presentation. Many multimedia applications call for an aperiodic presentation of media objects. Rate-base prefetching is inappropriate for such presentations and a greedy prefetching approach wastes buffer space. If the prefetch latency can be predicted for each object, then a prefetch schedule can be derived from the presentation schedule by subtracting predicted latency from display times. By prefetching so that data is available "just-in-time", the data can be displayed immediately with no buffering requirements.

On personal computers and some workstation environments, storage latency is repeatable and may be empirically determined by rehearsing a presentation [19, 53]. Even if some buffering is desirable to hide prefetch jitter, worst-case latency estimates may be used to determine a schedule that prefetches data as late as possible.

If prefetch latency is unpredictable, a worst-case bound may still provide a better prefetch schedule than the greedy approach. In a multi-user environment, a reasonable worst-case bound may require resource reservations as discussed below.

Speculative Prefetching

Prefetching strategies cannot satisfy unpredictable application access patterns unless all the candidate data objects can be prefetched simultaneously. Ghandeharizadeh, et al., describe an interesting approach in which the start of several possible data streams are prefetched before the user has selected which will be needed [26]. Each stream supplies the data for an outgoing path from the current location in a hypermedia graph. The storage system prefetches sufficient data for each path to satisfy presentation demands while the storage system seeks to the remainder of the data for the path that was selected.

2.2.7 Resource Reservations

Multimedia applications have proliferated in the personal computer world in part because most PC platforms provide a single-user environment. Without competition for resources from other users, the performance of multimedia applications can be predictable. Even

with multi-tasking, the scheduling of multimedia digital video and audio has been successfully achieved on single-user systems by elevating the priority of the media-handling tasks [48].

In a multi-user environment, reservations have been used to guarantee the availability of resources for a real-time application. Real-time file systems have been designed that guarantee a lower bound on bandwidth for sequential access to a file [46, 61, 5]. The Real-Time Mach operating system allows virtual memory pages to be "pinned-down" to avoid page faults [79]. Processor bandwidth may be reserved for periodic real-time tasks [51, 61]. Network bandwidth reservation protocols have been described and implemented [22, 2, 86].

A reservation protocol describes the parameters used to make a reservation. For example, the Continuous Media File System (CMFS) defines a real-time session by the maximum read size and read frequency guaranteed for sequential file access [5]. Each real-time file in the CMFS is created with a maximum rate parameter that constrains the size and frequency of seek times that may be incurred in all subsequent sessions that require sequential accesses to the file. Before a request for a real-time session can be granted, the reservation protocol runs an admission test to determine if its maximum throughput and minimum buffer requirements can be met concurrently with the previously guaranteed sessions. The CMFS disk scheduling algorithm reads (writes) enough data for each session during a cycle to make sure that none run out of data between cycles. Since the algorithm is conservative in estimating the amount of data needed to avoid starvation, the FIFO queues for each session will eventually fill up (empty out), reducing the scheduling policy to a round-robin schedule of greedy prefetching (write out). Non real-time file accesses may be handled during slack time before the next cycle starts.

Although processor, network and disk resources might seem very different, many of the reservation protocols are very similar. The Continuous Media (CM) Resource Model provides a general characterization of workload requirements for a resource [2]. A real-time file session is characterized by the unit of data access, guaranteed rate of delivery, and the *workahead* or number of units that may be delivered ahead of schedule to allow servicing bursts. Processing reservations are characterized by the duration of a processing task, rate of periodic task scheduling, and scheduling workahead. Requests for real-time

network connections specify message size, rate, and workahead. Processing and network resources are also characterized by the logical delay between the time a work unit arrives at the resource and the time that it is completed.

The reservation protocols cited above provide session guarantees beginning when the request is granted and continuing as long as the requester keeps the session open. Scripted presentations that include access to many separate files may find it expensive to open sessions on all files before beginning the presentation. Conversely, if the presentation is begun before all sessions have been guaranteed, then the failure of a later session request may make it impossible to finish the presentation. It may be necessary to extend existing reservation protocols to allow reservations that begin at a specific future time and have finite duration.

The problem of scheduling a set of tasks with time and resource constraints is known to be NP-complete [38]. While effective heuristic algorithms exist for this problem [91], they are sensitive to the uncertainty in task completion times. Worst case latency estimates can be so large as to make schedulability analysis impracticable.

The Spring Kernel provides dynamic scheduling of new real-time tasks in parallel with the execution of previously guaranteed tasks [72]. The principle feature of its scheduling approach is a functional partitioning of CPU and other resources between a planning scheduler and the dispatching and execution of guaranteed tasks. At any time, the scheduler has knowledge of the currently executing set of guaranteed tasks, their resource requirements and worst case execution times. When a new task arrives, the scheduler uses a heuristic algorithm to find a new schedule that avoids resource conflicts between tasks. If a feasible schedule is found, the new task is added to the guaranteed set and the old schedule replaced with the new. In complex scripted multimedia presentations, the number of storage access tasks to be scheduled can be very large and may swamp the capabilities of the algorithm used in the Spring Kernel. The complexity of their heuristic algorithm for scheduling a set of n tasks in a system having r resources is $O(rn^2)$ [91].

Blake and Schwan [8] report on another dynamic scheduler that uses a bin-packing approach to provide real-time guarantees (if possible) for dynamically occurring tasks.

As in the Spring Kernel, resources may be allocated exclusively by the scheduling process to avoid contention. Scheduling of periodic processes as a group and processes with precedence constraints is also supported. The bin-packing approach seems more appropriate for the advance reservation needs of a scripted application but is even more likely to be overwhelmed by large scheduling problems. Blake and Schwan claim only that the scheduling overhead is reasonable for moderate system loads of ten different deadline bins per processor.

Distributed systems may have concurrent applications with conflicting resource reservation requests. Since a reservation is a form of lock acquisition, the results of distributed transaction theory are applicable to distributed resource reservation protocols. In addition, since the resource requirements for a presentation may be interdependent, it may be impossible to choose the optimal reservation parameters until a minimally acceptable set of reservations has been granted. The SRP protocol for distributed transport and processing of continuous media consists of two phases: a resource acquisition phase to ensure that a feasible set of reservations can be granted followed by a relaxation phase to minimize end-to-end delay [2]. If the reservation protocol were to request too small an end-to-end delay value in the first phase, the request might not be granted due to an inability to schedule high-frequency (low-delay) service at some bottleneck resource.

2.3 Summary

Real-time scheduling is important for video presentations where a pause of 1/15th of a second is noticeable. But how much timing error can a presentation tolerate? The multimedia systems surveyed do not specify constraints on the accuracy of timing or even output values. Instead they use ad hoc techniques to approximately reproduce content with available resources and scheduling mechanisms. The number of techniques identified in this survey comprise a large space of variables for a multimedia system designer. The choice of techniques depends not only on resource availability but on the tradeoffs between different aspects of presentation quality. For example, a higher video frame rate may be achieved with lossy compression by reducing spatial resolution. An ad hoc choice

		technique	benefit	cost
computation	data location	continguous	disk bandwidth	storage specialization
		multiplexing	disk bandwidth	specialized composition
		striping	disk bandwidth	storage specialization
	format	lossless compression	storage, bandwidth	codec overhead
		lossy compression	storage, bandwidth	codec overhead, information loss
	processing	lossless optimizations	CPU bandwidth	code specialization
		runtime subsampling	CPU bandwidth	resolution, jitter
		cheap dithering	CPU bandwidth	image fidelity
synchronization	output	clock-driven	ordered, not early	clock overhead
		strict clock-driven	ordered, not early, no delay accumulation	runtime subsampling
		duration-driven	partially ordered, not early, minimum duration guaranteed	synchronization
	prefetching	greedy	jitter, bandwidth	buffer space, communication overhead
		rate-based	jitter, communication overhead	buffer space, rate synchronization
		scheduled	jitter, communication overhead, buffer space	schedule calculation
	reservations	dedicated resources	bandwidth	underutilization
		priority scheduling	CPU bandwidth	priority analysis
		bandwidth reservations	bandwidth	bandwidth analysis
		interval reservations	bandwidth	schedulability analysis

Figure 2.13: Summary of design technique benefits and costs.

of presentation techniques is not likely to perform well with a variety of content across different hardware and resource configurations.

Figure 2.13 shows a summary of the techniques described in this chapter indicating both the benefits and costs for each. The benefits for computation techniques are primarily in efficient resource use. The costs are harder to characterize. Data location techniques require specializations for a particular access pattern that may be inappropriate for other uses. Compression techniques require compression and decompression processing and may result in an irreversible loss of signal quality. Processing optimizations may depend on specialized assumptions as with the data location techniques; or information loss as with the lossy compression techniques. The benefits of synchronization techniques appear in the presentation timing. The output synchronization techniques provide specific guarantees for event ordering while prefetching and reservation techniques reduce jitter due to resource contention. Output synchronization techniques have only the cost of interrupt handling overhead, except that the strict clock-driven technique may also result in missed output events. The prefetching techniques all require buffering to hold the prefetched data until it is needed. In addition, each has some overhead associated with the determination of when to prefetch. Reservation techniques require some overhead for an admission test and may result in an underutilization of the resource if the admission test is too conservative.

To determine which techniques will produce acceptable presentation quality, it is necessary to represent quality requirements and to predict the quality of alternative presentation plans. The next chapter describes a general architecture for presentation planning and Chapter 4 describes the formal semantics of a presentation QOS specification. The systems surveyed in this chapter lack this formal basis for making QOS management decisions.

Chapter 3

Reference Architecture

3.1 Reference Architecture Description

This chapter describes a high-level architecture for planning presentations that satisfy formal QOS specifications. The terminology introduced here is used in subsequent chapters.

A QOS specification consists of a *content descriptor*, a *view descriptor*, and a *quality descriptor*. A *content descriptor* defines the logical structure and output values of a multimedia presentation. Digital audio and video data have default content descriptors associated with them that specify the sample size and rate for normal playback. Complex content descriptors may be composed from simpler content descriptors with an *editor* as illustrated in Figure 3.1. A *player* is used to browse and play-back content. The player generates a *view descriptor* to specify the ideal mapping of logical content onto physical devices and real-time. The parameters of a view descriptor include window size and playback rate. The player also generates a *quality descriptor* to limit the amount of error that can be allowed in a presentation. Some of the quality parameters include spatial and temporal resolution, delay, and jitter. The player may derive the view and quality descriptors from application context, user inputs, or both.

Content, view, and quality descriptors specify orthogonal aspects of a presentation, so that any instance of one may be combined with any instances of the other two to yield a valid QOS specification. A QOS specification is a predicate on the real-time state of device outputs that may or may not be satisfied by a particular playback execution. Chapter 4 provides a formal and complete definition of content, view, and quality.

Figure 3.2 illustrates how a *presentation manager* selects a *presentation plan* from

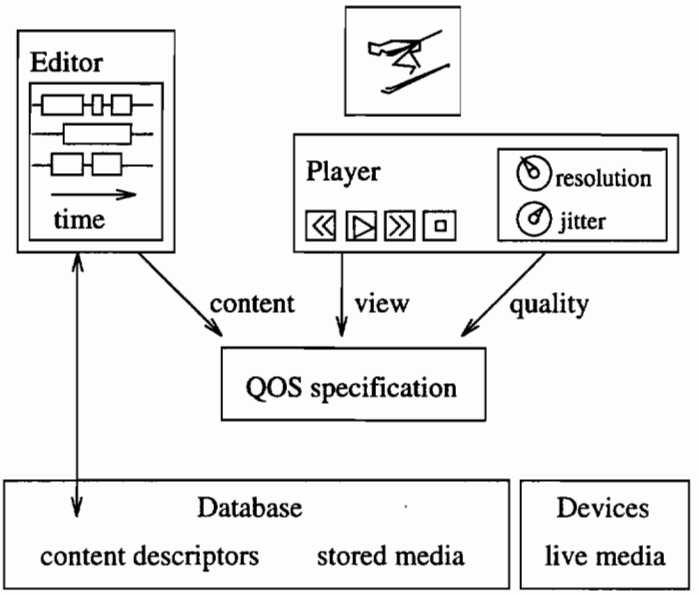

Figure 3.1: Editing and viewing multimedia presentations.

among a set of plans that compute device outputs from available data sources. Each presentation plan defines the resources required for each component and also the output quality that can be guaranteed if resource requirements are met. For example, with a given reservation of disk and CPU bandwidth, a prefetch task that reads compressed frames from an MPEG file can guarantee a lower-bound on the rate of frames read from disk. The presentation manager chooses a presentation plan whose resource requirements can be met with available resources and whose QOS guarantees are sufficient to satisfy the QOS specification.

Reservations are made in transactions with *resource managers*. The resource managers handle reservation requests from multiple applications and may deny a request if sufficient resources are unavailable. Some examples of resource reservation protocols are discussed in Chapter 2.

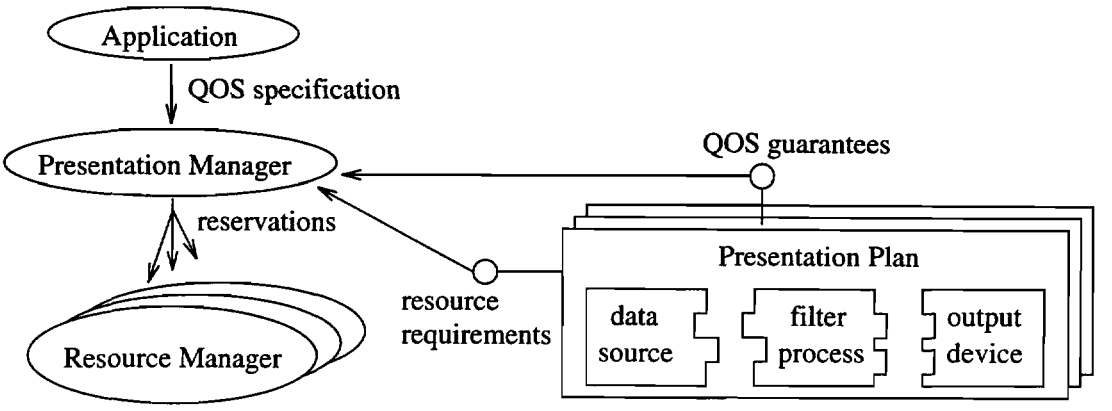

Figure 3.2: Translating QOS specifications into an acceptable presentation plan.

3.2 Application Examples

This section provides two examples of applications that require sophisticated multimedia database functionality with support for both high-quality presentations and highly interactive browsing. One is television news production and the other is a video database for professional sports. These applications were identified through our relationships with local industry. The key characteristics of these applications are the variation in QOS requirements for viewing and the need to share resources. Unlike video-on-demand applications that deliver uniform quality for each user, the demands of television news editors vary considerably depending on the task. A studio can support more concurrent users with a given set of resources by allocating resources according to actual QOS requirements than by allocating the resources equally. In the professional sports setting, a video player can support concurrent views more effectively if the QOS requirements for each view are explicitly specified.

3.2.1 Electronic News Gathering

The first example is Electronic News Gathering (ENG), which is expected to replace older analog video production technology in the broadcast news industry. Advances in computer and network technology have the potential to revolutionize the TV production industry by providing integrated support for the functions that are supported separately in analog

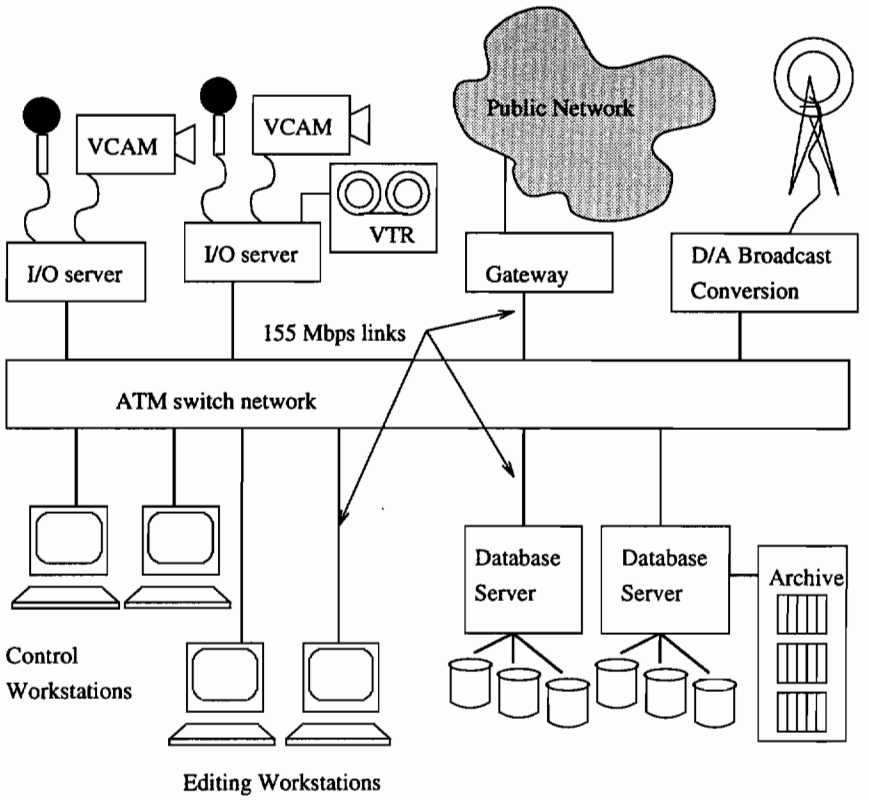

Figure 3.3: Architecture for a network-based digital television studio.

production studios. A digital television studio provides the functionality of analog studios, but with a single high-bandwidth data network replacing many separate dedicated data and control channels, and specialized software replacing expensive hardware for special effects and control.

The architecture of the studio, as sketched in Figure 3.3, consists of multiple database servers connected to editing and control workstations via an ATM network. Media input and output connections are made through workstation I/O devices, possibly on dedicated I/O servers. Software on each machine provides bandwidth guarantees for real-time interprocess communication.

The digital TV studio allows many users to work concurrently with random accesses to the database of continuous media. In contrast, analog video production relies on a master copy of each video tape that can only be accessed serially by a single user. The primary

functions of the studio consist of loading input media, editing content and specifying view and quality parameters for live output. The format and access characteristics of the input data are summarized below.

An NTSC color video signal can be digitized using 8-bit samples at three times the color subcarrier frequency to yield an 80Mbps datastream. This figure is a reasonable estimate of the bandwidth requirements for a production studio, since compressing the data by an order of magnitude will introduce visible artifacts with today's compression technology. When data is loaded into the database from live sources, it must be captured and written at this rate.

A single day's news-feeds might constitute 10 hours of video and require up to 360GB of storage. The studio can be expected to have many terabytes of archival video on site. A single user may be interested in only an hour of video data and thus might be able to work effectively with a 35 GB partition of the database.

Once in the database, video data is immutable. This constraint simplifies sharing of the data, since the same segment of a video may be included by reference in independent content descriptions without making independent copies. Playback views can require the simultaneous presentation of multiple video segments with one or more audio tracks. The first time a user requests such a presentation, the database will need to retrieve all the data streams concurrently. The aggregate bandwidth will be the sum of the bandwidths required for each individual stream.

Input capture must occur concurrently with editing and production in the studio. In particular, live feeds must be captured, even as they are passed through for broadcast, so that important events that occur during other stories or commercial interruptions are not lost and can be played back from storage.

Video tape recorders include time codes with every frame that tell the time at which the frame was captured. The information from these time-codes must be preserved when the data is loaded into the database, though it may be more useful to use them for indexing rather than leaving them embedded in the data stream. These time codes can be used to automatically resynchronize video with its associated soundtrack. Similarly, when multiple cameras are recording the same proceedings, the timecodes provide a way

to accurately cut between cameras on playback without losing audio synchronization. In addition to the time codes, video data will have annotations such as title, author and location of shoot. These annotations can be used to query the database for appropriate video segments.

The persons who edit the news need to retrieve useful video footage, load new data, interactively view the data, and compose selected segments into new video content descriptions. Interactive viewing includes manual fast-forward and reverse control to find visual and audio cues. A precise, or high-quality, playback rate is not as important during this interactive search as it is during normal-speed presentation. Interactive viewing does require that full frames be retrieved without reading the entire data stream serially from storage, since the latter might need several orders of magnitude more bandwidth than normal-speed playback.

Content composition operators include sequential cuts from one data stream to another, parallel compositions such as lip-synched audio or voice-over narration, spatial layout of multiple regions of a display, and combination of inputs, including transitions such as wipes and fades.

For reviewing the compositions, presentation must be real-time with broadcast-quality signal reproduction and synchronization of media elements. The products of the editing process are content descriptors. These descriptors specify the media selections and compositions to be performed for playback. In contrast to the raw input media, these descriptors are viewed as mutable data that are updated in place. After creation of a content descriptor, committed versions may be viewed as immutable in order to facilitate sharing among editors. Eventually, a complex descriptor may be simplified by copying the data referenced into a contiguous space: a process sometimes referred to as "flattening". However, flattening results in the loss of the original context of the component media segments.

The digital television studio is representative of large multimedia systems with networked computing resources and multi-user execution environments. Real-time constraints are harder to meet in such systems because of the greater number of factors that contribute to delay, including network communications and contention for resources between competing users. The studio requirements for concurrent multi-user access to stored video

data argues for a global namespace and shared storage resources. Because the video data consumes large amounts of bandwidth and storage, it is not possible to replicate the entire database for each user. At the same time, it will almost certainly be necessary to do some caching of data when a user may need to search interactively through the data many times without interference from other users.

Database technology to search for and retrieve data is needed in the digital television studio. Searching requirements include support for content-based queries that specify attributes such as objects, people, shapes and textures. Browsing requirements include support for logical views of multimedia data and optimized real-time presentation. Today's database systems are only beginning to address these needs.

ENG provides a good example of the need for formal QOS specifications. During searching and editing functions, it is desirable to support many concurrent streams at various play rates and resolutions. For example, suppose that a visual search of NTSC video at ten times normal speed can be effectively performed with 1/4 the resolution (1/16 the bandwidth), but cannot afford to drop more than 9 frames in a row because of the risk that an important visual cue would be missed. The result is that the presentation has low resolution requirements, but must still display 30 frames per second. A QOS specification appropriate to the playback task tells the system where resource use can be reduced without an unacceptable degradation of presentation quality. Such resource optimizations allow more users to share resources without conflict. The specification language must be rich enough to express requirements for resolution, sample rate, image quality and potentially many other aspects of quality that can vary in a presentation.

3.2.2 Digital Video Support for Professional Sports

Professional sports teams already have extensive analog videotaping and video production capabilities. Multimedia computing promises to extend these current capabilities with automated media annotation, ad-hoc query facilities, and random access review and editing capabilities.

Our local National Basketball Association (NBA) franchise, the Portland Trailblazers,

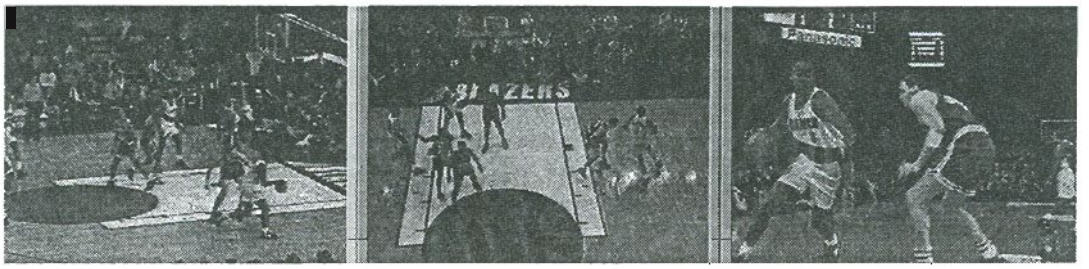

Figure 3.4: Synchronized views of sporting action.

regularly videotapes each home game with three or more separate cameras and microphones. The cameras offer different viewpoints onto the action and the microphones provide sound tracks from either end of the floor and from the game commentary. These sources provide approximately 6 hours of video per game.

The analog videotaping provides accurate time-codes in each video stream to allow re-synchronization of the separate recordings. Further annotation can be derived from manual statistics gathering, which records player names and actions such as blocked shots, assists, field goals, and free throws.

These video recording and annotation functions can be incorporated into a digital multimedia system with better support for interactive browsing. As with the ENG example in the last section, the video recordings may be stored as immutable data and indexed through time and annotations. The statistics annotations, which must be recorded in near-real-time, allow ad hoc queries for time intervals that include selected actions and players. A query might also specify one or more camera views to be displayed with accurate synchronization as illustrated in Figure 3.4.

A professional sports team needs to study and evaluate the performance of both its own players and its opponents. Quick access to replays involving a particular player can help to identify and confirm patterns of behavior. Once an interesting play has been located, interactive control of the temporal display point and slow motion replays are useful for seeing actions that are easily overlooked at normal play speed. The multiple viewpoints shown in Figure 3.4 reveal details that would be hidden in any one camera view.

As with the ENG application discussed in the last section, digital video support for

professional sports requires visual searching and concurrent playback of multiple streams from large video databases. During a fast forward search, it may not be possible to provide full spatial resolution without frame dropping. Similarly, it may not be possible to provide multiple concurrent video displays with full resolution and maximum frame rate.

A QOS specification can express the relative importance of synchronization, resolution, frame rate, and image quality. For example, to view a basketball game, the user might desire one large video display window and several small windows to monitor the other camera angles. If the user selects one of the small windows, its content is subsequently displayed on the large window. This change affects both view and quality parameters. The large window would require the highest quality in all aspects since this would be the main focus of attention. The smaller windows have significantly lower quality requirements in every aspect and a QOS specification is needed to instruct the system how to achieve this unequal allocation of resources.

3.3 Summary

The previous examples illustrate the need for intelligent control of presentation QOS, both within a single application and between competing users. The key features of these examples are the variation in QOS requirements for concurrent presentations and the need to handle resource overloads. Chapter 4 describes a general method for defining presentation QOS and provides a detailed semantics for QOS specifications using content, view, and quality descriptors. The content descriptors support the major composition operations required by our application examples. Playback requirements for rate control and image scaling can be expressed through the view descriptor. Requirements for image quality, resolution, frame rate, and other parameters can be expressed through the quality descriptor.

Chapter 4

Specification of Presentation Quality

This chapter presents a formal semantics for presentation QOS. The method for defining these semantics consists of 3 steps: defining an ideal presentation, choosing an model for describing error in an actual presentation, and representing constraints on that error. The first step is illustrated with a new language for describing complex *content* and for describing a presentation *view* of that content. The new language permits a simple definition of an ideal presentation, but we believe similar definitions can be based on the multimedia authoring languages described in Chapter 2. The relation between an ideal presentation and the measurable outputs is of an actual presentation are described with an error model. The second step identifies the properties that an error model for presentations must have and compares several alternatives. We give a formal definition for an error model that subsumes many of the QOS parameters suggested by other researchers. Finally, we describe alternative forms for QOS constraints based on the error model. Section 4.4 gives a formal semantics for a conservative example of QOS constraints.

The purpose of these formal specifications is to enable system designers to reason about the correctness of QOS management algorithms. The prototype described in Chapter 5 demonstrates that the definitions of content, view, and quality given in this chapter can be practically applied. The method used to develop these definitions can be applied to define QOS semantics for other authoring models and less conservative constraints.

4.1 Z Notation

We use a subset of the Z (pronounced "Zed") specification language as defined in The Z Reference Manual [69] and augmented with the standard arithmetic and calculus operators and relations defined over the set of real numbers R. The definitions of syntax and notation given here should be used as a reference while reading the rest of the chapter.

The example below shows a global declaration of a schema type S. A Z schema type consists of a signature of typed variables together with constraints on those variables. A declaration $v : S$ says that v has schema type S and the components $v.x$ and $v.y$ must obey the constraints for x and y respectively in schema S.

$$
\begin{array}{|l}
\hline
S \\
x, y : R \\
\hline
y \geq x \\
\hline
\end{array}
$$

Other global functions and constants can be declared with an axiomatic description. The following example declares that *length* is a function from schema type S to the set of real numbers and that, for all variables v of type S, *length* v is the difference between the y and x components of v.

$$
\begin{array}{|l}
\hline
length : S \longrightarrow R \\
\hline
\forall\, v : S \bullet length\ v = v.y - v.x \\
\hline
\end{array}
$$

Free type definitions declare type constructors and arguments that generate a new type. For example, a *Tree* is either a *leaf* or a *branch* with an integer value and two subtrees. The type constructor *branch* is a function from a 3-tuple of an integer and two *Trees* to a *Tree*.

$$Tree ::= leaf \mid branch\langle\!\langle \mathbb{Z} \times Tree \times Tree \rangle\!\rangle$$

The Z language includes common notation from set theory and first-order logic. The brief definitions here are for additional notation that may be unfamiliar.

$S : \mathbf{P}\, X$	S is declared as a subset of X
$X \leftrightarrow Y$	binary relation on X and Y
$X \longrightarrow Y$	total function from X to Y
$X \nrightarrow Y$	partial function from X to Y
dom f	domain of the function f
ran f	range of the function f
min S	the minimum of a nonempty set of numbers S
max S	the maximum of a nonempty set of numbers S
seq X	a finite sequence with elements of type X
$\langle\rangle$	the empty sequence
head s	head of sequence
tail s	tail of sequence
$\forall x : T \bullet P$	for all x with type T, P is true
$\exists x : T \bullet P$	there exists some x with type T such that P is true
(**let** $x == E1 \bullet E2$)	let x be an abbreviation for $E1$ in $E2$
$\{ x : T \mid P \}$	the set of all x such that x has type T and P is true
$\{ x : T \mid P \bullet E \}$	the set of all E such that x has type T and P is true

Where evaluation order for these expressions is unclear we have used parentheses to provide an unambiguous interpretation. We frequently use the last two set expressions above to specify a set of tuples, in which case the declaration to the left of the vertical bar is a list of tuple elements and their types.

4.2 Content Specification

To provide a concrete example of presentation QOS semantics, this section defines a data model for specifying the content of multimedia presentations. The model supports composition of audio and video data to create complex presentations. Other media such as text and still images may be included by modeling them as video stills with non-zero duration. This data model can be extended to support user interaction by making content specifications depend on the timed sequence of user inputs, but we have kept it simple

so as not to distract from the main task of defining presentation quality. Because our subsequent definition of quality is independent of the content description model, it will apply to more complicated models.

Our content specifications define a set of logical output channels and the acceptable real-number values for those outputs that may vary continuously with time. An important feature of this model is that the audio and video specifications may have infinite resolution. For example, the visualization of a continuous function whose values can be computed rather than read from storage is limited by the computational resources and the display device, but not by the content specification. Our content specifications provide physical data independence since they do not describe the representation of source data.

4.2.1 Content Descriptors

The specification begins with a declaration of two types: real numbers and integers. Digital inputs and outputs will be declared as integers, but nearly all other quantities will be modeled as real numbers. Real numbers are used for the specification of logical values to avoid placing an artificial limit on the content resolution. The reals are declared as a basic type and integers are declared as a subset of reals:

$$[R, \mathbb{Z}]$$

$$\mathbb{Z} \subset R$$

The *Interval* schema gives a start position and an interval extent. These are used for both clipping intervals and linear transformations, as described later.

Interval

$start : R$

$extent : R$

To make it easier to treat outputs uniformly, a single schema describes output dimensions. This schema must contain the maximal set of dimensions for all output types. For specification of audio outputs, the x and y intervals are unimportant as long as they have

positive extent. The *Space* schema specifies intervals for t, x, and y coordinate dimension and a z interval for the output range. For example, the dimensions of a video source are described with a *Space* that stores the start time and duration in the t interval, the image dimensions in the x and y intervals, and the range of signal values in z.

Space

$t : Interval$

$x : Interval$

$y : Interval$

$z : Interval$

A *Content* descriptor is a recursive construct built from basic audio and video sources. Each *audio*, *video* descriptor defines a single logical output. More complex content may be specified using *clip*, *transform*, *cat*, *synch*, and *select* descriptors. The *LOutput* type is used in the *select* descriptor to reference a particular logical output. To distinguish logical outputs, each *LOutput* is identified by its media type, *mAudio* or *mVideo*, and by an integer.

$$LOutput ::= mAudio\ \langle\langle \mathbb{Z} \rangle\rangle \mid mVideo\ \langle\langle \mathbb{Z} \rangle\rangle$$

$$MTypes == \{mAudio, mVideo\}$$

$$\begin{aligned} Content ::= {} & audio\langle\langle Space \times (R \longrightarrow R) \rangle\rangle \\ \mid {} & video\langle\langle Space \times (R \longrightarrow R \longrightarrow R \longrightarrow R) \rangle\rangle \\ \mid {} & clip\langle\langle Space \times Content \rangle\rangle \\ \mid {} & transform\langle\langle Space \times Content \rangle\rangle \\ \mid {} & cat\langle\langle \text{seq } Content \rangle\rangle \\ \mid {} & synch\langle\langle \text{seq } Content \rangle\rangle \\ \mid {} & select\langle\langle LOutput \times Content \rangle\rangle \end{aligned}$$

The *audio* descriptor takes a pair with a *Space* descriptor and a function from a real time coordinate to a real z value. The domain and range for the function are specified with the *Space* descriptor. As described in the following sections, the resolution of a

presentation is limited only by an actual implementation on digital outputs. For example, the sine function could define an audio source with no implied limit on the resolution of the signal. The *video* descriptor also takes a pair with a *Space* descriptor and a function, but the video source function requires additional real coordinates for x and y. Again the domain and range for the function are specified by the *Space* descriptor. For simplicity, this definition supports only monochrome video, but the same approach can be generalized to specify a tuple of values at each point for color.

Figure 4.1 provides an example of a content descriptor. The leaves of the tree consist of two *video* descriptors and the one *audio* descriptor. The first video descriptor references an external data source, *camA*, and declares that image values range from 0 to 256 and are defined over t values from 0 to 115, and (x, y) pairs from $(0, 0)$ to $(320, 240)$. The second video source named *camB* is defined over t values from 0 to 53, and (x, y) pairs from $(0, 0)$ to $(640, 480)$. The audio source *micA* has values ranging from 0 to 256 that are defined for time values from 0 to 100. The first video is scaled by a factor of 2 in x and y to match the dimensions of the second video and is offset by -100 so that the clip can begin at logical time zero. The second video and the audio are both offset for synchronization with the first video. The video presentation is assembled by concatenating a clip of seconds 0-5 from the first transformed video with seconds 5-8 from the second, followed by the clip of seconds 8-15 from the first again. The result is then synchronized with a clip of seconds 0-15 from the transformed audio.

The *transform*, *clip*, *cat*, *synch*, and *select* descriptors support stretching and shrinking, cut, paste, synchronization, and selection of logical outputs. The formal meaning of each descriptor is given in the next subsection. These descriptors are very similar to the algebraic video operators described in Chapter2 [85]. Our *cat* is similar to their `concatenation` operator. Our *transform* and *synch* descriptors are similar, but more general than their `stretch`, ||, and ⟆ operators. They support additional features, such as transition effects from one video segment to the next, but do not provide a formal presentation semantics. Our definition of a small set of very general content descriptors makes it easier to describe a formal semantics, while still supporting the composition of useful and complex multimedia presentations.

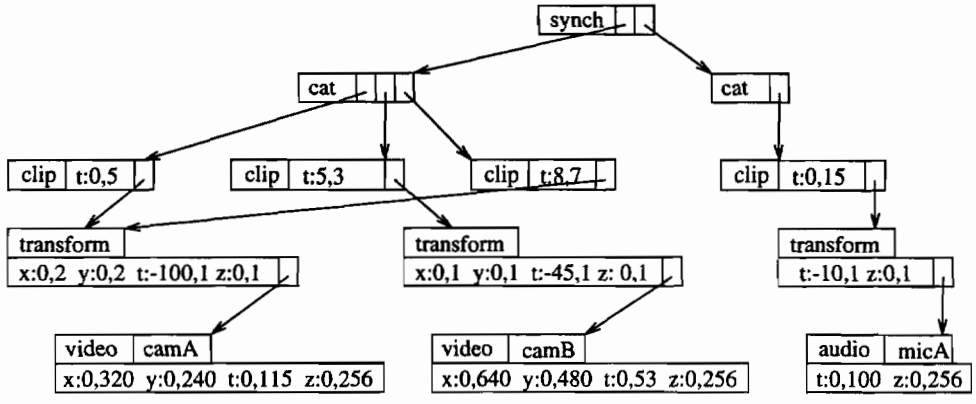

Figure 4.1: Content descriptor example.

4.2.2 Semantics

The meaning of a content descriptor is defined by a set of allowed logical output values for every point of the logical output space. Let the *Interval* function I return the set of real numbers in an interval. Our definition of an interval includes the start point but not the end point. Then a point (x, y, t) is in the logical space s if $(x \in I\ s.x) \wedge (y \in I\ s.y) \wedge (t \in I\ s.t)$.

$$I : Interval \longrightarrow \mathbf{P}\, R$$

$$I\ v = \{\, r : R \mid (v.start \leq r) \wedge (r < v.start + v.extent) \,\}$$

The *Interval* type can also describe linear transformations. For any *Interval* i, $tr\ i$ is the linear transformation that maps the unit interval onto i and $utr\ i$ maps i onto the unit interval. For example, if $i.start = 3$ and $i.extent = 2$ then $tr\ i\ 0 = 3$ and $tr\ i\ 1.1 = 5.2$.

$$tr, utr : Interval \longrightarrow R \longrightarrow R$$

$$tr\ i\ x = x * i.extent + i.start$$

$$utr\ i\ x = (x - i.start)/i.extent$$

Content descriptors constrain logical output values *only* during explicit intervals. For example, the content descriptor in Figure 4.1 allows any output values before logical time 0 and after logical time 15. The functions *start*, *end*, and *duration* are used to reference

the logical time interval over which output values are constrained by a content descriptor. The logical start of a content descriptor is the minimum time t at which *some* output value is *not* acceptable! The logical end is the minimum time t such that no output value is constrained for times greater than or equal to t.

Content descriptors also constrain only a finite number of logical outputs. In Figure 4.1 only two logical outputs are constrained and we refer to these two logical outputs as $mAudio1$ and $mVideo1$. All other *LOutput* descriptors refer to unconstrained logical outputs. The function *num* takes a logical output type and a content descriptor and returns the integral number of logical outputs of that type that are constrained by the specification. The *restrict* function is used to guarantee that a number is within an interval. The function *restrict* takes an *Interval* and a number and repeatedly subtracts or adds the interval extent to the number until it can return a value within the interval.

$$
\begin{array}{|l}
start, end, duration : Content \longrightarrow R \\
num : MType \longrightarrow Content \longrightarrow \mathbb{Z} \\
restrict : Interval \longrightarrow R \longrightarrow R \\
\hline
start\ c = min\ \{\ t : R\ \mid \\
\quad \neg\ (\forall\, l : LOutput;\ x, y, z : R \bullet (l, x, y, t, z) \in logical\ c)\ \} \\
end\ c = min\ \{\ t : R\ \mid \forall\, t' : R \bullet (t \leq t') \Rightarrow \\
\quad (\forall\, l : LOutput;\ x, y, z : R \bullet (l, x, y, t', z) \in logical\ c)\ \} \\
duration\ c = end\ c - start\ c \\
num\ m\ c = max\ \{\ n : \mathbb{Z} \mid\ \neg\ (\forall\, x, y, t, z : R \bullet (m\ n, x, y, t, z) \in logical\ c)\ \} \\
restrict\ i\ x = ((x - i.start) modulo\ i.extent) + i.start
\end{array}
$$

The meaning of each of the content descriptors is captured by the following definition of a function for logical content. For a given content descriptor, the *logical* function returns a relation between a point in the logical output space and the *acceptable* output values for that point. The expression $(l, x, y, t, z) \in logical\ c$ means the content descriptor c allows the logical output l, at point (x, y) and time t to have value z. Note that specifications

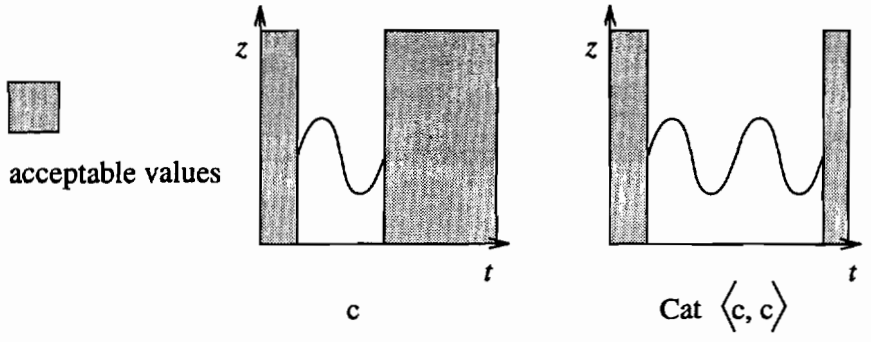

Figure 4.2: Content semantics.

reduce the set of allowable values as illustrated in Figure 4.2. Where nothing is specified, all values are acceptable.

$$LValue == LOutput \times R \times R \times R \times R$$

$$logical : Content \longrightarrow \mathbf{P}\ LValue$$

$$logical(audio(s,f)) = \{\, l : LOutput;\ x, y, t, z : R \mid (l = mAudio\ 1) \land (t \in I\ s.t) \Rightarrow z = restrict\ (I\ s.z)\ (f\ t)\,\}$$

$$logical(video(s,f)) = \{\, l : LOutput;\ x, y, t, z : R \mid (l = mVideo\ 1) \land (x \in I\ s.x) \land (y \in I\ s.y) \land (t \in I\ s.t) \Rightarrow z = restrict\ (I\ s.z)\ (f\ t\ y\ x)\,\}$$

$$logical(clip(s,c)) = \{\, l : LOutput;\ x, y, t, z : R \mid (x \in I\ s.x) \land (y \in I\ s.y) \land (t \in I\ s.t) \Rightarrow (\exists\, z' : R \bullet (l, x, y, t, z') \in logical\ c \land z = restrict\ (I\ s.z)\ z')\,\}$$

$$logical(transform(s,c)) = \{\, l : LOutput;\ x, y, t, z : R \mid (l, x, y, t, z) \in logical\ c \bullet (l, tr\ s.x\ x, tr\ s.y\ y, tr\ s.t\ t, tr\ s.z\ z)\,\}$$

$$logical(cat\langle\rangle) = \{\, l : LOutput;\ x, y, t, z : R \mid true\,\}$$

$$logical(cat\ q) = logical(head\ q) \cap \{\, l : LOutput;\ x, y, t, z : R \mid (l, x, y, t, z) \in logical(cat(tail\ q)) \bullet (l, x, y, t + end(head\ q) - start(cat(tail\ q)), z)\,\}$$

$$logical(synch(\langle\rangle)) = \{\, l : LOutput;\ x, y, t, z : R \mid true\,\}$$

$$logical(synch\ q) = logical(synch(head\ q)) \cap \{\, m : MType;\ n : \mathbb{Z};\ x, y, t, z : R \mid (m\ n, x, y, t, z) \in logical(synch(tail\ q)) \bullet (m(n + num\ m\ (head\ q)), x, y, t, z)\,\}$$

$$logical(select(m\ n, c)) = \{\, m' : MType;\ n' : \mathbb{Z};\ x, y, t, z : R \mid (m' = m) \land (n' = 1) \Rightarrow ((m\ n, x, y, t, z) \in logical\ c) \bullet (m'\ n', x, y, t, z)\,\}$$

The first predicate for $logical(audio(s,f))$ says that if l is the logical output $mAudio\ 1$ and t is within the interval $s.t$, then the only acceptable value for z is the function $f(t)$. Otherwise, any values are acceptable for z. The predicate for $logical(video(s,f))$ expresses a similar constraint for the logical output $mVideo\ 1$.

A $clip(s,c)$ descriptor specifies that for all logical outputs, points with x, y, and t coordinates in the *Space* s are constrained to have the values specified by c restricted to

the interval $s.z$. All points not in s are effectively "clipped" out and may have any value.

A *transform*(s, c) descriptor specifies a linear transformation of points in the content specified by c. For example, if *start* $c = 0$, *duration* $c = 60$, $s.t.start = 10$, and $s.t.extent = 2$, then *start*(*transform*$(s, c)) = 10$ and *duration*(*transform*$(s, c)) = 120$. The transformation descriptor *transform*(s, c) with all start fields in s equal to zero and all extent fields in s equal to one is the identity transformation and has no effect.

A temporal sequence of content can be specified with a *cat*(q) descriptor. The content for a member of the sequence q is logically shifted in time to start just as the previous content in the sequence ends. For example, a concatenation of two video descriptors results in a new content descriptor whose duration is the sum of the parts. In general, a content descriptor *cat* $\langle c_1, c_2, ..., c_n \rangle$ defines a sequence of transition times $(t_1, t_2, ..., t_{n+1})$, where $t_i = start\ c_1 + \sum_{j=1}^{i-1} duration\ c_j$. During each interval $[t_i, t_{i+1})$ the logical outputs are defined by the content descriptor c_i, offset to start at time t_i. Note that if a descriptor c_1 constrains only the logical output *mVideo* 1, but a descriptor c_2 constrains two logical outputs *mVideo* 1 and *mVideo* 2, then the descriptor *cat* $\langle c_1, c_2 \rangle$ constrains only *mVideo* 1 during the first interval and both *mVideo* 1 and *mVideo* 2 during the second interval.

The *synch*(q) descriptor specifies that a set of content descriptors all reference the same time scale, but that their logical outputs are disjoint. For example, each *video* descriptor constrains the single logical output *mVideo* 1. If c_1 and c_2 are two *video* descriptors, then *synch* $\langle c_1, c_2 \rangle$ constrains the two logical outputs *mVideo* 1, and *mVideo* 2. The behavior of *mVideo* 1 is the same as specified by c_1, while the behavior of *mVideo* 2 is the same as c_2 specified for *mVideo* 1. The start time for the *synch* content descriptor is the earlier of the start times of the original videos, while the end time is the later of the original two end times. The logical audio and the logical video outputs defined by a *synch* descriptor are independently renumbered according to their occurrence in the sequence of content descriptors.

The *select*(l, c) descriptor offers a way to reference only the content of a single logical output within a complex descriptor. Where the *synch* descriptor aggregates multiple logical outputs into a single specification, *select*(l, c) specifies only a single logical output with the same content as c specifies for logical output l. For any logical output type

m and integer n, the logical output defined by *select*$(m\ n, c)$ is $(m\ 1)$. This maintains the invariant that for all content descriptors c, constrained logical outputs of type m are numbered from 1 to *num* m c. If a content descriptor does not constrain a logical output l then *select*(l, c) is the null specification; all values are permissible on all outputs.

Figure 4.1 shows an example of a content descriptor in *normalform*. In normalform, every descriptor is a directed acyclic graph with a *synch* descriptor at the root. The *synch* descriptor specifies a sequence of *cat* descriptors. Each *cat* descriptor specifies a single logical output with a sequence of *clip* descriptors. Each *clip* specifies a portion of a *transform* descriptor and each *transform* descriptor defines the logical dimensions of a basic media source. A basic media source must be either an *audio* or *video* descriptor. Every content descriptor that forms a finite, acyclic graph can be converted automatically to a normalform descriptor that specifies the same logical content. The algorithm relies on the fact that *audio* and *video* descriptors can be trivially represented in normalform and each of the other content constructors can be eliminated if their children are in normalform.

This definition of content satisfies the goal of a data model for complex presentations except that there is no way to relate the logical content to actual presentation outputs. The logical outputs of a content specification have both temporal and spatial proportions, but they have no physical size or real duration. The next section describes how the content is mapped to physical coordinates by a *View* specification.

4.3 View Specification

A *View* specification allocates physical devices for logical outputs and maps logical time to a real-time clock. While the physical devices may present an upper bound on spatial and temporal resolution, the view does not specify presentation quality. We choose to define the presentation output to be the values of device registers written by the presentation process. We could instead measure the analog output of audio and video devices, but the digital-to-analog conversion is typically inflexible and presents no opportunities for resource optimization.

Figure 4.3 shows a view descriptor that allocates a small window on a monochrome

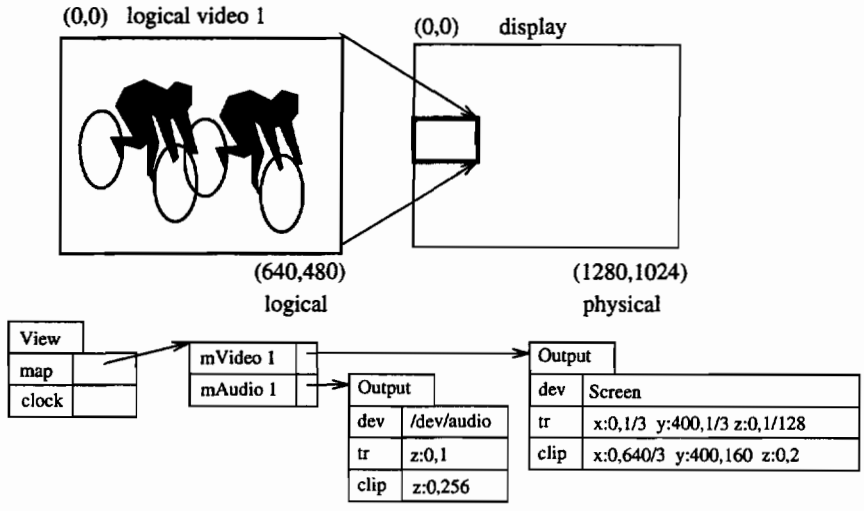

Figure 4.3: Example of a view descriptor.

(black and white) display for a bicycling video presentation. Although the output device clearly limits the quality of the presentation, the view does not specify how the content is to be represented on the display. It is the presentation plan, as described in Chapter 3, that determines how to sample the source and how to represent gray scale information. The combination of content and view descriptors serve as a device independent and physical-data independent specification of a perfect quality presentation. The idea of an ideal presentation is formally defined in this section. The next section defines less-than-perfect quality based on the difference between this ideal presentation and actual presentation outputs.

4.3.1 View Descriptors

Since the details of physical I/O devices are unimportant for specifying an ideal presentation, the following declaration simply assumes that there is a set of audio output devices *AudioDev* and video output devices *VideoDev*. A *Device* is either one of the audio devices or one of the video devices. The only requirement for the implementation of a device is that it supports writing output values at the coordinates specified in a *View*.

[*AudioDev*, *VideoDev*]

$Device == AudioDev \cup VideoDev$

The logical dimensions in a content specification are generally not the same as the physical dimensions of the view. The *Output* schema declares a field *tr* that defines the transformation from logical to view output dimensions and a field *clip* that defines clipping bounds for view outputs. In Figure 4.3, the *Output* descriptor for *mVideo* 1 scales the 640x480 logical image by a factor of 1/3 and then shifts the corner of the image down to display coordinate (0,400). The z values are scaled from the logical range (0,256) to the display range of (0,2). The clipping bounds for video match the full range of the transformed content. The audio content is not transformed in this example. The *Output* schema also allows each output to have an independent mapping from logical time to real-time. However, in the implementation described in Chapter 5, an identity mapping for time is used to preserve synchronization between logical outputs.

Output

$dev : Device$

$tr, clip : Space$

A *View* specifies a partial function *map* that assigns a subset of the logical outputs to physical *Output* descriptors. Logical outputs that are not in the domain of the map function are ignored. The *tr* field is used to transform logical time in a content specification to a real-time clock. The *clip* field specifies the real-time start and duration of the presentation. Just as the details of I/O devices are unimportant, the designation of a real-time clock is left to an implementation.

View

$map : LOutput \pfun Output$

$tr : Interval$

$clip : Interval$

This definition of a *View* does not prevent us from mapping a logical audio output to a video device or a logical video to an audio device. While a good user interface for

view specification would prevent a user from creating such mappings by mistake, these cross-type mappings do not present a problem for view semantics. We simply assert that at each point in time, an audio signal defines a constant image intensity over its clipping region in x and y, while a video image defines a range of values that the audio signal may have.

4.3.2 Semantics

A content specification together with a view specification defines an *ideal* presentation, where the output devices are assumed to have infinite resolution. This assumption is necessary for a device-independent definition of quality. A presentation is modeled as a set of *DValue* tuples (d, x, y, t, z) that give the z value for a particular *Device* d and coordinates x, y, and t.

$$DValue == Device \times R \times R \times R \times R$$

The function *ideal* c v returns the relation between devices and the values specified by a *Content* descriptor c and a *View* descriptor v. As with the previous definition of logical content, an ideal presentation allows any device values except where constrained by the content and view descriptors. The relation *ideal* c v contains all *DValue* tuples (d, x, y, t, z), where the view maps a logical output l to a device d and x, y, and t are within the clipping bounds for d, only if the corresponding logical value is allowed by the content descriptor c. The corresponding logical point is expressed by substituting l for p and "un-transforming" x, y, t, and z back to logical space. For example, let c be the content descriptor shown in Figure 4.1, let v be the view descriptor from Figure 4.3, and let v map *start* c onto the real-time s. Then the tuple $(Screen, 0, 400, s, (camA\ 0\ 0\ 0)/128)$ is in *ideal* c v because the x and y coordinates are in the clipping rectangle for the video *Output* descriptor and the view maps this tuple to the *LValue* $(mVideo\ 1, 0, 0, start\ c, camA\ 0\ 0\ 0)$, which is in the set *logical* c.

$$ideal : Content \longrightarrow View \longrightarrow \mathbf{P}\, DValue$$

$$\begin{aligned}
&ideal\ c\ v = \{\, d : Device; x, y, t, z : R \mid \\
&\quad (\exists\, l : LOutput; p : Output \bullet (l \in \mathrm{dom}\ v.map) \wedge (v.map\ l = p) \wedge (p.dev = d) \\
&\qquad \wedge (t \in I\ v.clip) \wedge (x \in I\ p.clip.x) \wedge (y \in I\ p.clip.y) \Rightarrow \\
&\qquad\quad (l, utr\ p.tr.x\ x, utr\ p.tr.y\ y, utr\ v.tr\ t, utr\ p.tr.z\ z) \in logical\ c)\,\}
\end{aligned}$$

4.3.3 Actual Presentation

The z values in an ideal presentation can be compared directly with measurable outputs in an actual presentation. To make this comparison easier, we also model an actual presentation as a set of *DValue* tuples.

The implementation of a presentation plan uniquely determines the value for every device at every point and time. The schema *Presentation* models a discrete-valued presentation with integer functions for audio and video outputs. The *audio* function takes an *AudioDev* and an integer clock value and return an integer z value. The *video* function takes a *VideoDev* and integer values for the clock, x, and y coordinates, and returns the integer value at that pixel. These definitions assume that only one output value can be observed for each clock tick and for each video pixel. The z function provides a device independent way to refer to presentation device values. The *Point* schema is introduced here to simplify notation in the next section.

Point

$$x, y, t : R$$

Presentation

$$audio : AudioDev \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z}$$

$$video : VideoDev \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z}$$

$$z : Device \longrightarrow Point \longrightarrow R$$

$$d \in AudioDev \Rightarrow z\ d\ (x, y, t) = audio\ d\ \lfloor t \rfloor$$

$$d \in VideoDev \Rightarrow z\ d\ (x, y, t) = video\ d\ \lfloor x \rfloor\ \lfloor y \rfloor\ \lfloor t \rfloor$$

The next section defines a mapping from presentation device coordinates and values to the ideal values specified for some *Content* and *View*. This mapping will serve as the basis for a definition of presentation quality.

4.4 Quality Specification

Our definition of presentation quality is motivated by a few observations:

1. For a given task, the utility of a presentation can be measured empirically.
2. By definition, an ideal presentation delivers the highest utility.
3. Utility decreases as presentation error increases.
4. Utility is non-negative if users can recognize and ignore bad presentations.

The utility of a presentation is a task-dependent empirical performance measure, such as the probability of correctly identifying a face. We define presentation *quality* to be the ratio of the utility of the actual presentation to the utility of an ideal presentation. Quality is unity when the actual presentation is without error and monotonically approaches zero as presentation error increases. Although the precise dependence of presentation quality on error must be determined empirically for each task, we suspect that many of these relations can be modeled with a single parameterized function.

This section provides a formal definition of presentation error and a function for estimating presentation quality based on presentation error. The definition of presentation error depends only on the observable presentation outputs and not on the presentation mechanism. This ensures that the QOS specifications described later are both device independent and physical data independent. In particular, the definition of quality is not based on the data throughput required for a presentation, but instead can be used to derive throughput requirements as shown in Chapter 5.

The declaration for an *ErrorInterpretation* in Section 4.4.1 is the most important part of our QOS specification because it provides an *error model* for describing presentation error. Let *Error*[*Names*] be a schema with named error component functions. Each error component function takes a device and a point and returns a real number.

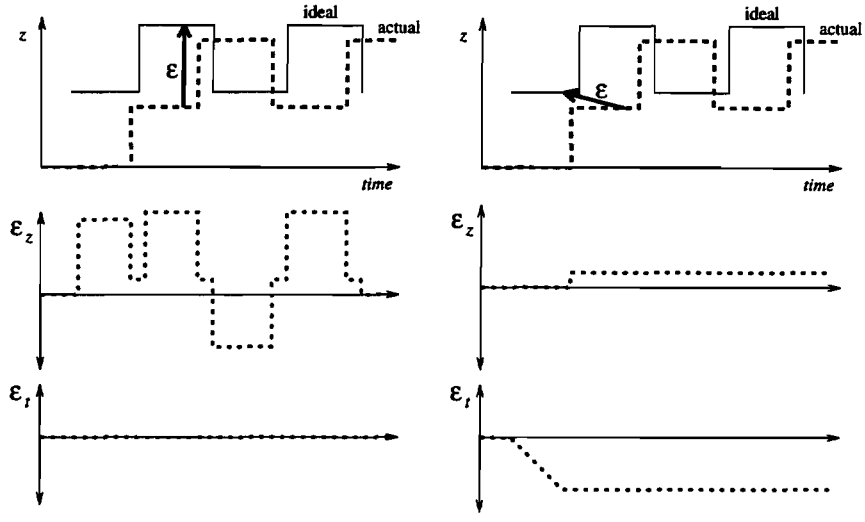

Figure 4.4: Multiple interpretations of error.

$$\begin{array}{|l} \textit{Error}[\textit{component}_1 ... \textit{component}_n] \\ \hline \textit{component}_1, ..., \textit{component}_n : \textit{Device} \longrightarrow \textit{Point} \longrightarrow R \\ \hline \end{array}$$

This declaration uses a generic construct to represent a different schema for each set of component names. We call an instance of some schema $Error[Names]$ an *error interpretation.* An *error model* defines a set of error interpretations that account for the differences between an actual presentation and an ideal presentation.

$$ErrorModel[Names] == (Presentation \times \mathbb{P}\, DValue) \longrightarrow \mathbb{P}\, Error[Names]$$

It is important to recognize that an error model can allow many different error interpretations for a given actual and ideal presentation. Consider the two examples in Figure 4.4. The ideal z value for an audio output is shown as a function of time. The error between the ideal presentation and an actual presentation can be represented as a vector function of time, $\vec{\varepsilon}$ with two error components for the z and t dimensions. On the left we show $\vec{\varepsilon}$ for a particular time near the beginning of the presentation. The vector function $\vec{\varepsilon}$ has a large variable ε_z component, but the ε_t component is always zero. On the right side of Figure 4.4, the error between the same ideal and actual presentations can be described by a constant ε_t component and a smaller constant ε_z component. Both interpretations

satisfy an error model that requires $\vec{\varepsilon}$ to map points in the actual presentation to points in the ideal.

The example error vector function has only two error components to describe error as a function of time on a single audio output. To describe error in a multimedia presentation, we need to define such a vector for every output device as a function of both spatial and temporal coordinates. We would also like to be able to describe spatial errors in x and y in addition to timing and z errors. Let M_B be the *basic error model* that defines a set of error interpretations with x, y, t, and z components that map values in a *Presentation* P to ideal values in a set S:

$$
\begin{array}{|l}
M_B : \mathit{ErrorModel}[x, y, t, z] \\
\hline
M_B(P, S) = \{\varepsilon : \mathit{Error}[x, y, t, z] \mid \\
\quad (\forall\, d : \mathit{Device};\ p : \mathit{Point} \bullet \\
\quad (d, p.x + \varepsilon.x\ d\ p, p.y + \varepsilon.y\ d\ p, p.t + \varepsilon.t\ d\ p, P.z\ d\ p + \varepsilon.z\ d\ p) \in S)\}
\end{array}
$$

M_B allows interpretations that express arbitrary displacements in any presentation dimension, with the exception that it does not allow presentation values on one output device to be mapped to ideal values on another device. A good presentation will output z values that are close to the ideal at approximately the correct time and at approximately the correct x and y coordinates. Thus, a good presentation P for some ideal presentation S may be characterized by a small number $\delta > 0$ such that, there exists an error interpretation $\varepsilon \in M_B(P, S)$ in which the magnitude of the vector $(\varepsilon.x\ d\ p, \varepsilon.y\ d\ p, \varepsilon.t\ d\ p, \varepsilon.z\ d\ p)$ is less than δ for all *Device* d and *Point* p. We call the set of all such presentations a *neighborhood* of the ideal presentation S and denote this set with $\aleph(\delta, S)$. Conversely, a poor presentation will make large temporal, spatial, or z value errors that preclude the existence of a small error interpretation. We can specify arbitrary accuracy in a multimedia presentation with *Content* descriptor c and *View* descriptor v by choosing a sufficiently small positive value for δ and requiring that the presentation be in the neighborhood $\aleph(\delta, \mathit{ideal}\ c\ v)$.

A neighborhood constrains all error components equally, ignoring any differences in importance between timing error and spatial error. A more general approach uses weights

for each error component to compute a normalized error vector. We define a *weighted-neighborhood* $\aleph_\omega(M, \omega, \delta, S)$ to be the set of presentations with error interpretations that, when normalized with positive weights from ω, are everywhere less than δ for an error model M and ideal presentation S:

$$Positive == \{r : R \mid r > 0\}$$

$$\begin{array}{|l} Weight[component_1 ... component_n] \\ \hline component_1, ..., component_n : Device \longrightarrow Point \longrightarrow Positive \\ \hline \end{array}$$

$$\begin{array}{|l} \aleph_\omega : (ErrorModel[Names] \times Weight[Names] \times Positive \times \mathbf{P}\, DValue) \\ \quad \longrightarrow \mathbf{P}\, Presentation \\ \hline \aleph_\omega\ (M, \omega, \delta, S) = \{P : Presentation \mid \\ \quad (\exists \varepsilon : Error[Names] \bullet \varepsilon \in M\ P\ S \\ \qquad \wedge \forall\, d : Device; p : Point \bullet (\sum_{i \in Names} (\frac{\varepsilon.i\ d\ p}{\omega.i\ d\ p})^2)^{1/2} < \delta\} \end{array}$$

If each component of an error weight function ω always returns a value of one, then $\aleph_\omega\ (M_B, \omega, \delta, S) = \aleph\ (\delta, S)$. For other error models and error weights, a weighted-neighborhood can describe a different set of presentations. However, we would like all error models to share the following useful properties of M_B. We say an error model M is *sound* if every weighted-neighborhood using M contains some positive neighborhood of the ideal and is also contained by some finite neighborhood of the ideal. This property assures us that our specifications will accept some non-trivial subset of good presentations and disallow presentations with unbounded error. The formal definition of soundness for error models is:

- An error model $M : ErrorModel[Names]$ is *sound* if:

$$\forall \omega : Weight[Names]; \delta : Positive; S : \mathbf{P}\, DValue \bullet$$
$$(\exists\, \delta', \delta'' : Positive \bullet \aleph(\delta', S) \subseteq \aleph_\omega(M, \omega, \delta, S) \subseteq \aleph(\delta'', S))$$

We say an error model M is *complete* if every finite neighborhood of the ideal is contained by some weighted-neighborhood using M and every set of non-ideal presentations is excluded by some weighted-neighborhood using M. This property assures us that we can specify tolerance for any finite error and specify intolerance for arbitrary levels of error, although not necessarily with the same specification ofa weighted neighborhood. The formal definition of completeness for error models is:

- An error model $M : \mathit{ErrorModel}[\mathit{Names}]$ is *complete* if:

$$\begin{aligned}
&\forall \omega : \mathit{Weight}[\mathit{Names}]; \delta : \mathit{Positive}; S : \mathbb{P}\, \mathit{DValue} \bullet \\
&\quad (\exists \delta' : \mathit{Positive} \bullet \aleph(\delta, S) \subseteq \aleph_\omega(M, \omega, \delta', S)) \\
&\quad \wedge\ (\exists \delta'' : \mathit{Positive} \bullet \aleph_\omega(M, \omega, \delta'', S) \subseteq \aleph(\delta, S))
\end{aligned}$$

The definition for a complete error model observes that every presentation P that contains *DValue* tuples not in an ideal presentation S must have a $\delta > 0$, such that $\aleph(\delta, S)$ excludes P. M_B can be shown to be sound by choosing δ' equal to the reciprocal of the maximum weight and δ'' equal to the reciprocal of the minimum weight. M_B can be shown to be complete by choosing all weights equal to one. But M_B is a poor basis for specifying presentation QOS. Ideally, a presentation QOS specification should accept all presentations that the user would accept and reject only those that the user would reject. A *conservative* specification is one that never accepts a presentation that the user would reject. M_B supports only very conservative specifications that reject many good presentations. For example, if an application can tolerate up to a 5 second delay in the start of a video presentation, but then will tolerate no more than 1/10 second of jitter in the timing accuracy, then a 1 second delay in the start with negligible timing error afterward would be acceptable. Yet, the conservative specification would reject this presentation, because M_B is incapable of distinguishing between delay and jitter. We say that an error model M_J is more *expressive* than an error model M_K if, for any conservative specification $\aleph_\omega(M_K, \omega, \delta, S)$, there exists a conservative specification $\aleph_\omega(M_J, \omega', \delta', S)$ that accepts a strict superset of the presentations in $\aleph_\omega(M_K, \omega, \delta, S)$. The next subsection describes a more expressive error model that is used to define our QOS semantics.

4.4.1 Reference Error Model

The following *ErrorInterpretation* schema defines many new error components to achieve a better match between conservative QOS specifications and human perception. The declarations for *shift*, *jitter*, and the other functions define a set of error component names for an error model. We call this error model, the *reference error model*, abbreviated as M_R. The *shift* error component allows a presentation to be behind (or ahead of) schedule. Rather than require the time shift to be constant, many applications allow it to vary through a presentation. The rate at which the time shift changes can be constrained by a *rate* error component. The *rate* error is zero while the *shift* error is constant, but increases in magnitude when the presentation speeds up or slows down. M_R also includes a *jitter* error component, which allows small "hiccups" in the timing error that would be prohibited by constraints on *rate* error if they were attributed to shift. For example, if each frame of a video has an ideal time for its display, then the video display constitutes a logical clock that advances in discrete jumps. Rather than accounting for these discontinuities in the *rate* error component, the small jumps in time may be interpreted as *jitter*.

How much of the timing error is due to *jitter* and how much to *shift* is a matter of interpretation. There is no information in the presentation outputs to distinguish timing error from z error and no information to distinguish *jitter* from *shift*. Instead, the "best" interpretation of error depends on which error components have the least effect on presentation quality.

A *synch* error component for each pair of devices is defined as the difference in the shift error at each device. The *synch* component allows the specification of a high tolerance for shift errors while at the same time specifying a low tolerance for synchronization error between outputs.

The *shift*, *rate*, and *jitter* error components are defined similarly for x and y dimensions since spatial presentations can suffer from displacement, scaling and small distortions analogous to the temporal error components.

Even after accounting for temporal and spatial errors, the difference between an actual presentation value and the corresponding ideal value at an infinitesimal point is not

particularly meaningful. The problem is that humans don't perceive independent values at infinitesimal points, but instead integrate over small display areas and time intervals. This fact is routinely exploited by graphics algorithms that use dithering. For example, a black and white display can represent a 50% gray tone by a pattern with every other pixel turned on. Dithering trades off spatial resolution for more accurate tone or color values. Resolution in the x dimension can be thought of as the width of the narrowest vertical stripe that can be reproduced by a presentation. Resolution in y and in time has a similar intuitive definition. Then the interesting measure of z error is the difference in average value over the neighborhood of a point defined by x, y, and t resolution. This definition of z error allows the error model to interpret objective value errors as a combination of perceived resolution loss and perceived value errors.

The following declaration of the *avg* function simplifies the definition of the reference error model in the *ErrorInterpretation* schema. The *avg* function is needed to express the relation between error in z and resolution error in x, y, and t. The expression *avg res p f* returns the average value of the function f over the cube with size *res p* centered on point p. Because audio outputs do not vary in x or y, the average *avg res p f* is independent of the x and y resolution components of an error interpretation.

$$avg : (Point \longrightarrow Point) \longrightarrow Point \longrightarrow (Point \longrightarrow R) \longrightarrow R$$

$$(avg\ res\ p\ f =$$
$$(\textbf{let}\ r == res\ p;\ r_1 == p - (r/2);\ r_2 == p + (r/2) \bullet$$
$$\frac{1}{r.x * r.y * r.t} \int_{r_1.x}^{r_2.x} \int_{r_1.y}^{r_2.y} \int_{r_1.t}^{r_2.t} f)$$

An *ErrorInterpretation* can now be declared as a set of error component functions that satisfy M_R for a particular trio of *Content*, *View*, and *Presentation*. The *jitter* and *shift* functions return a vector of x, y, and t components so that they each define three independent error component functions. The *rate* function is defined as the differential of *shift*, i.e. the gradient of each component of *shift*. The result of the *rate* function is a 3×3 matrix containing x, y, and t components for each gradient vector. The *res* function also returns a vector of x, y, and t components that define the size of the cube around each point for computing average *zError*. The *synch* function returns the difference in

the time component of *shift* error between any two output devices. Error components for devices and points that are not constrained by the content and view will always allow a zero interpretation of error. The number of devices that are constrained by the content and view descriptors define a finite set of error components that might not have a zero error interpretation. To simplify notation, the "+" operator is used to add functions with the obvious meaning that $(f + g)\ x = (f\ x) + (g\ x)$.

Matrix

$$x, y, t : Point$$

ErrorInterpretation

$$c : Content$$
$$v : View$$
$$P : Presentation$$
$$jitter : Device \longrightarrow Point \longrightarrow Point$$
$$shift : Device \longrightarrow Point \longrightarrow Point$$
$$rate : Device \longrightarrow Point \longrightarrow Matrix$$
$$res : Device \longrightarrow Point \longrightarrow Point$$
$$zError : Device \longrightarrow Point \longrightarrow R$$
$$synch : Device \longrightarrow Device \longrightarrow Point \longrightarrow R$$

$$\forall\, d : Device; p : Point \bullet$$
$$\exists f : Device \longrightarrow Point \longrightarrow R \bullet$$
$$(\textbf{let}\ \varepsilon == jitter\ d\ p + shift\ d\ p \bullet$$
$$(d, p.x + \varepsilon.x, p.y + \varepsilon.y, p.t + \varepsilon.t, P.z\ d\ p + f\ d\ p) \in ideal\ c\ v)$$
$$\wedge\ rate\ d = differential\ (shift\ d)$$
$$\wedge\ avg\ (res\ d)\ p\ (f\ d) = avg\ (res\ d)\ p\ (zError\ d)$$
$$\wedge\ \forall\, d' : Device \bullet synch\ d\ d'\ p = (shift\ d\ p).t - (shift\ d'\ p).t$$

To show that the reference error model M_R is sound, we need to find a subset $\aleph(\delta', S)$ and a superset $\aleph(\delta'', S)$ for any weighted neighborhood $\aleph_\omega(M_R, \omega, \delta, S)$. Let δ' be an

unspecified positive real number. Then, by the definition of a neighborhood, for any presentation $P \in \aleph(\delta', S)$, there exists an error interpretation $\varepsilon_B \in M_B(P, S)$, such that:

$$\forall\, d : Device;\ p : Point \bullet (\textstyle\sum_{i \in \{x,y,t,z\}} (\varepsilon_B.i\ d\ p)^2)^{1/2} < \delta'$$

This implies that each of the error components, $\varepsilon_B.x$, $\varepsilon_B.y$, $\varepsilon_B.t$, and $\varepsilon_B.z$, is everywhere less than δ'. Let ε_R be the interpretation in $M_R(P, S)$ such that:

$$\varepsilon_R.jitter = (\varepsilon_B.x, \varepsilon_B.y, \varepsilon_B.t)$$
$$\varepsilon_R.zError = \varepsilon_B.z$$

and all other components of ε_R are zero. If ω_{min} is the minimum weight from ω for all devices and points, then the magnitude of the error vector defined by ε_R is everywhere less than $((\delta'/\omega_{min})^2 * 4)^{1/2}$. If we choose $\delta' = (\delta * \omega_{min})/2$ then we have identified a neighborhood that is a subset of $\aleph_\omega(M_R, \omega, \delta, S)$. Now consider any presentation $P \in \aleph_\omega(M_R, \omega, \delta, S)$. By the definition of a weighted neighborhood, there exists an error interpretation $\varepsilon_R \in M_R(P, S)$ such that:

$$\forall\, d : Device;\ p : Point \bullet (\textstyle\sum_{i \in Names_R} (\frac{\varepsilon_R.i\ d\ p}{\omega.i\ d\ p})^2)^{1/2} < \delta\}$$

where $Names_R$ is the set of error component names for the reference error model M_R. Let ω_{max} be the largest weight from ω for all devices and points. Then the magnitude of every component of ε_R is everywhere less than $\delta * \omega_{max}$. Let ε_B be an error interpretation in $M_B(P, S)$ such that:

$$\forall\, d : Device;\ p : Point \bullet (\varepsilon_B.x, \varepsilon_B.y, \varepsilon_B.t) = \varepsilon_R.shift + \varepsilon_R.jitter$$

Then the error component $\varepsilon_R.zError$ is the average of $\varepsilon_B.z$ over a region defined by $\varepsilon_R.res$. There must be an upper bound $\varepsilon_{max} > \varepsilon_B.z$ for all devices and points. If not, we could prove a contradiction, since the average of $\varepsilon_B.z$ is finite, an ideal specification always allows finite values, and since a *Presentation* is, by definition, constant over unit regions. Taking $\delta'' = ((2 * \delta * \omega_{max})^2 * 3 + \varepsilon_{max}^2)^{1/2}$, it follows that the neighborhood $\aleph(\delta'', S)$ is a superset of $\aleph_\omega(M_R, \omega, \delta, S)$.

In the proof of soundness, we showed that for any set $\aleph_\omega(M_R, \omega, \delta, S)$ we can find a subset $\aleph(\delta', S)$ and a superset $\aleph(\delta'', S)$. The proof of completeness uses the same reasoning

to find a superset $\aleph_\omega(M_R, \omega, \delta', S)$ and a subset $\aleph_\omega(M_R, \omega, \delta'', S)$ for any $\aleph(\delta, S)$ and weight function ω.

The reference error model M_R is more expressive than M_B, since M_R is equivalent to M_B if *shift* and *res* error components are interpreted as zero everywhere, but non-zero interpretations for these components allow smaller interpretations of *jitter* and *zError*.

4.4.2 Quality Descriptors

Quality is a schema that declares a value *min* to represent the minimum acceptable level of quality and a function *estimate* for estimating quality from an error interpretation. The *estimate* function uses values from weight functions $jitter_\omega$, $shift_\omega$, $rate_\omega$, res_ω, $zError_\omega$, and $synch_\omega$ to model the importance of each error component. A small weight indicates that quality is very sensitive to the corresponding error component. Conservative estimates of presentation quality can be made by specifying sufficiently small return values for all weight functions. The *estimate* models quality as an exponential decay function of the error vector magnitude. This model has the following properties:

- quality is one when all error components are zero.
- quality decreases monotonically with an increase in any error component.
- quality approaches zero as error components approach infinity.

Quality

$$min : R$$

$$jitter_\omega : Output \longrightarrow Point$$

$$shift_\omega : Output \longrightarrow Point$$

$$rate_\omega : Output \longrightarrow Matrix$$

$$res_\omega : Output \longrightarrow Point$$

$$zError_\omega : Output \longrightarrow R$$

$$synch_\omega : Output \longrightarrow Output \longrightarrow R$$

$$estimate : ErrorInterpretation \longrightarrow Output \longrightarrow Point \longrightarrow R$$

$$(0 \leq min) \wedge (min < 1)$$

$$estimate\ \varepsilon\ o\ p =$$

$$(LET\ \varepsilon_j == \varepsilon.jitter\ o.dev\ p; \varepsilon_s == \varepsilon.shift\ o.dev\ p; \varepsilon_d == \varepsilon.rate\ o.dev\ p;$$

$$\varepsilon_r == \varepsilon.res\ o.dev\ p; \varepsilon_z == \varepsilon.zError\ o.dev\ p;$$

$$\omega_j == jitter_\omega\ o; \omega_s == shift_\omega\ o; \omega_d == rate_\omega\ o;$$

$$\omega_r == res_\omega\ o; \omega_z == zError_\omega\ o;$$

$$\varepsilon_{norm} == ((\tfrac{\varepsilon_j.x}{\omega_j.x})^2 + (\tfrac{\varepsilon_j.y}{\omega_j.y})^2 + (\tfrac{\varepsilon_j.t}{\omega_j.t})^2 + (\tfrac{\varepsilon_s.x}{\omega_s.x})^2 + (\tfrac{\varepsilon_s.y}{\omega_s.y})^2 + (\tfrac{\varepsilon_s.t}{\omega_s.t})^2 +$$
$$(\tfrac{\varepsilon_d.x.x}{\omega_d.x.x})^2 + (\tfrac{\varepsilon_d.x.y}{\omega_d.x.y})^2 + (\tfrac{\varepsilon_d.x.t}{\omega_d.x.t})^2 + (\tfrac{\varepsilon_d.y.x}{\omega_d.y.x})^2 + (\tfrac{\varepsilon_d.y.y}{\omega_d.y.y})^2 + (\tfrac{\varepsilon_d.y.t}{\omega_d.y.t})^2 +$$
$$(\tfrac{\varepsilon_d.t.x}{\omega_d.t.x})^2 + (\tfrac{\varepsilon_d.t.y}{\omega_d.t.y})^2 + (\tfrac{\varepsilon_d.t.t}{\omega_d.t.t})^2 + (\tfrac{\varepsilon_r.x}{\omega_r.x})^2 + (\tfrac{\varepsilon_r.y}{\omega_r.y})^2 + (\tfrac{\varepsilon_r.t}{\omega_r.t})^2 + (\tfrac{\varepsilon_z}{\omega_z})^2$$
$$+ \textstyle\sum_{o' \in \mathrm{ran}\,\varepsilon.v.map} (\tfrac{i.synch\ o.dev\ o'.dev\ p}{synch_\omega\ o\ o'})^2)^{1/2} \bullet$$

$$e^{-\varepsilon_{norm}})$$

4.4.3 Semantics

The meaning of the quality schema in conjunction with content and view descriptors is given by the following schema for a *QOS* specification:

QOS

c : *Content*

v : *View*

q : *Quality*

P : *Presentation*

$\exists \varepsilon$: *ErrorInterpretation* $\bullet$ $\varepsilon.c = c \wedge \varepsilon.v = v \wedge \varepsilon.P = P \wedge$

$(\forall o \in \text{ran}\, \varepsilon.v.map; p$: *Point* $\bullet$

$q.min \leq q.estimate\ \varepsilon\ o.dev\ p)$

This schema consists of *Content*, *View*, and *Quality* descriptors that constrain a presentation P. The *QOS* specification is satisfied only if an *ErrorInterpretation* exists for c v and P such that, at every point on every output, the quality of the presentation is greater than or equal to $q.min$.

This definition for QOS specifications is very strict in that quality must exceed the minimum *everywhere* during a presentation. It would be nice to extend the specification semantics to allow a presentation to occasionally drop below this minimum quality, but this extension is left for future work.

For a given presentation and its specification, the reference error model allows an infinite number of interpretations, each with a different affect on the calculation of presentation quality. What matters is that an interpretation exists that has acceptable errors. This claim assumes that humans are good at recognizing the intended presentation content and that they will recognize an interpretation with acceptable error if it exists.

To calibrate the quality estimation function, the functions in a *Quality* descriptor can be defined from empirical studies of user perception. These values returned by the functions $jitter_\omega$, $shift_\omega$, $rate_\omega$, res_ω, $zError_\omega$, and $synch_\omega$ are called *critical error values.* For every error component in the error model, there is a corresponding critical error value in the *Quality* descriptor. When an error component equals the corresponding critical error value the quality is at most e^{-1} or approximately 0.37. For a simple user model, these critical error values can be chosen to correspond to poor quality.

4.5 Summary

The QOS semantics described in this chapter demonstrate the following important results:

- Content, view, and quality are orthogonal.
- The specification of an ideal presentation allows a formal definition of presentation error.
- Differences between actual and ideal presentations can be accurately described by many different error models.
- QOS specifications can have device independence and physical data independence by requiring only the existence of a satisfactory presentation-level error interpretation.
- A quality estimation function can be used to specify satisfactory presentations.

The definition of *Content* and *View* descriptors given in Sections 4.2 and 4.3 provide a minimal language for multimedia authoring with simple semantics for an ideal presentation. Other languages provide a richer authoring environment, but fail to define an ideal presentation. MAEstro, OCPNs, and the MHEG standard all describe an operational semantics where the timing of one presentation event may depend on the run-time behavior of another presentation process [19, 41, 52]. These languages can be extended to define an ideal presentation by specifying ideal run-time behavior for all presentation actions. With such extensions, formal QOS specifications can be defined by following the framework described in this chapter.

The definition of a *Quality* descriptor in Section 4.4 provides an expressive error model and quality estimation function for constraining presentation error. The reference error model borrows familiar concepts such as "jitter" from the literature on QOS specification. However, ours is the first formal definition of these error components in terms of a mapping from an actual to an ideal presentation. The reference error model allows specifications with a high tolerance for one component of error, such as temporal shift, and a low tolerance for another, such as temporal jitter. In the next chapter we demonstrate that this error model is expressive enough for practical applications. However, more expressive

error models may be desirable to express tolerance for other presentation artifacts. For example, another error model could distinguish errors in image brightness and contrast from image noise.

Chapter 5

A QOS-Driven Multimedia Player

5.1 Purpose and Scope of the Prototype

Chapter 3 described an architecture for specifying multimedia presentations and for planning and scheduling resources to satisfy the specifications. The SQUINT multimedia player provides an implementation of the specification and planning portions of this architecture. As the name suggests, SQUINT supports controls for image resolution and other components of presentation quality. SQUINT is also an acronym for Smalltalk QOS User INTerface, because it makes heavy use of the Smalltalk programming environment [57]. This section gives a brief overview of the design goals for the player. A detailed description of the design and implementation of the player are given in Sections 5.2, 5.3, and 5.4.

The main purpose of the player is to show how content, view, and quality descriptors can be generated and used for resource scheduling. SQUINT demonstrates the orthogonality of content, view and quality by allowing any content to be displayed with any view and any quality specification. The trio of content, view, and quality descriptors form a QOS specification that is used to request worst-case error guarantees from a presentation manager. Of course, the computing platform and software components limit the best quality that can be achieved. When SQUINT detects a conflict between platform capabilities and QOS requirements, a description of the conflict location is generated.

Figure 5.1 illustrates the control panel for the player. The **content** button brings up a menu of content descriptors. The descriptors are created outside of SQUINT as described in the next section. Selecting from the content menu opens a display window for each video track in the presentation and displays the name of the selected content at the top

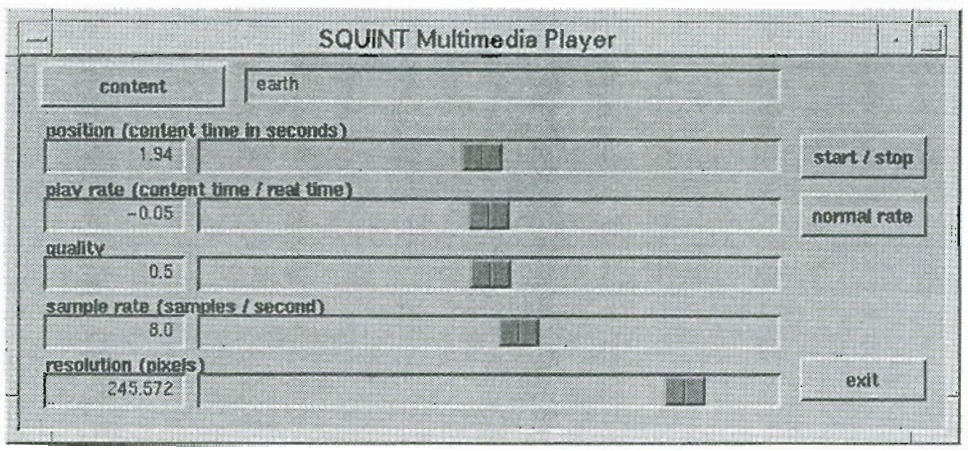

Figure 5.1: The SQUINT multimedia player.

of the control panel. Audio tracks are played using the HP Audio Application Program Interface [30]. The **position** slider shows the current logical time in seconds, where the left and right ends of the slider correspond to the presentation start and finish respectively. The **play rate** slider controls the rate at which logical time advances. For example, setting the slider at 2.0 makes the playback occur twice as fast as normal. The **normal rate** button sets the rate to 1. The **start/stop** button also controls the rate by toggling between the current **play rate** setting and zero. In addition to the buttons and sliders on the control panel, the video windows have the standard Motif window decorations that support positioning and resizing. As discussed in Section 5.2, the player controls generate view and quality descriptors for the presentation.

The SQUINT interface also supports experimentation with user QOS controls. The control panel allows the user to set the minimum acceptable quality measure and also the calibration values for image resolution and jitter. These controls are described fully in Section 5.2.3. The prototype interprets the QOS specification derived from user interface controls as an accurate specification of application requirements. SQUINT can be easily modified to obtain calibration values from other sources, such as a table indexed by playback mode.

Formal specification of QOS requirements can be used to optimize resource usage in a multimedia system. SQUINT provides a simple example of QOS-driven resource

optimization. The variables for stored video access include spatial resolution and frame skipping. Each of several test video sources has been encoded and stored in multiple files at full, 1/2, 1/4, and 1/8 resolution. All files encode every frame at the original frame rate, but the presentation has the option to skip frames for a lower presentation frame rate. If presentation QOS requirements permit lower resolution or lower frame rate, then SQUINT will reduce the file access bandwidth to conserve both CPU and disk resources. The calibration values for the quality estimation function described in Chapter 4 allow SQUINT to balance bandwidth tradeoffs intelligently between frame rate and spatial resolution.

We call a presentation plan *viable* if it would satisfy QOS requirements when resources are plentiful. That is, the plan is guaranteed to have an acceptable error interpretation if there are no scheduling delays. We call a presentation plan *acceptable* if it is viable and scheduling requirements can be met. Given a QOS specification, the determination of an acceptable presentation plan is referred to as the *mapping problem.* A general solution for the mapping problem is intractable and depends on real-time resource scheduling. In particular, SQUINT does not provide a priori guarantees for presentation timing since timing guarantees require real-time scheduling not only for SQUINT, but also for X Window, audio server, and file system processes. Instead, SQUINT maps QOS requirements onto a *viable* presentation plan. The player then monitors jitter at run-time and invokes an error notification handler when jitter guarantees are violated. SQUINT can be extended to explore the mapping problem with scheduling guarantees, distributed resources and more dimensions for variable quality and resource usage.

SQUINT currently supports arbitrary compositions of synthetic video and of stored uncompressed monochrome video data. Synthetic videos are defined by a continuous real function of x, y, and time. Limited support is available for presentations of MPEG-1 encoded video and standard audio formats. SQUINT can display any number of video tracks in separate windows and can play a single audio track using the default audio server. Because SQUINT does not support mixing outputs, multiple audio tracks cannot be played on a single audio device.

```
| vSrc aSrc composition |

vSrc := DigitalVideo
    file: 'dog' width: 256 height: 240 depth: 4 sampleRate: 10.0.

aSrc := DigitalAudio
    file: 'bark' depth: 8 sampleRate: 8000.

composition := Synch new
    specs: (OrderedCollection
        with: (Video new source: vSrc; space: vSrc space)
        with: (Audio new source: aSrc; space: aSrc space)).
```

Figure 5.2: Smalltalk syntax for creating content descriptors.

5.2 QOS Request Generation

5.2.1 Creating and Selecting Content

The process for creating and editing content descriptors is external to the SQUINT player. In the Smalltalk programming environment, content descriptor objects can be created from any text window by evaluating code expressions as shown in Figure 5.2. In this example, three temporary variables are declared with the names `vSrc`, `aSrc`, and `composition`. The first two variables are assigned the results of `Media` object constructor expressions that describe stored data. The last expressions assigns `composition` to be a new `Synch` content descriptor. This content descriptor contains a collection of references to `Video` and `Audio` content descriptors that reference the digital video and audio media.

Content is described by objects that closely model the Z content descriptors in the last chapter. Figures 5.3 and 5.4 introduce a subset of the OMT (Object Modeling Technique) notation used in subsequent figures [24]. Figure 5.5 shows the SQUINT Content class hierarchy and a subset of the protocol associated with each class.

The class `Content` maintains a dictionary of content descriptors that may be referenced by name. The Smalltalk environment provides a dictionary inspector that supports adding new descriptors and editing existing ones. SQUINT opens a pop-up menu of names in

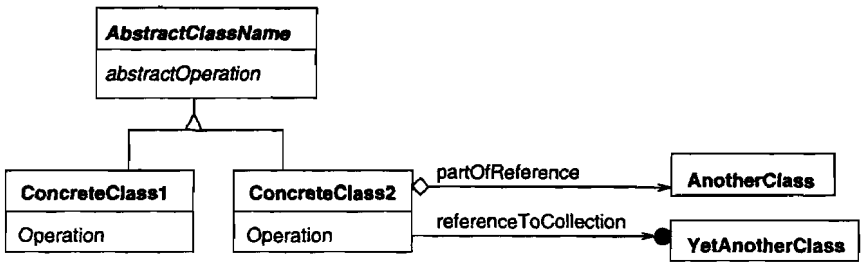

Figure 5.3: OMT notation for Class relationships.

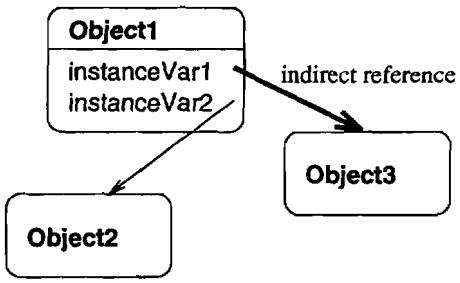

Figure 5.4: OMT notation for object snapshots.

the `Content` dictionary when the button labeled **content** is pressed. The content selected by name from the pop-up menu is displayed in the top text field of the SQUINT control panel, and a default view for the content is opened as described below.

5.2.2 View Controls

The class `ViewSpec` shown in Figure 5.6 describes a view descriptor that maps from logical to physical outputs and from logical time to a real-time clock. The values in this view specification can be modified via the SQUINT control panel.

A default view descriptor is created when SQUINT's content selection is changed. The `ViewSpec` class method `#defaultFor:` takes a content descriptor as an argument and creates a view descriptor with one window for each video track and at most one output for an audio track. For each window, the view has an `Output` descriptor that specifies a spatial transformation `tr` and clipping bounds `clip`. For the default view, the window size and clipping bounds are taken from the logical dimensions of the first clip

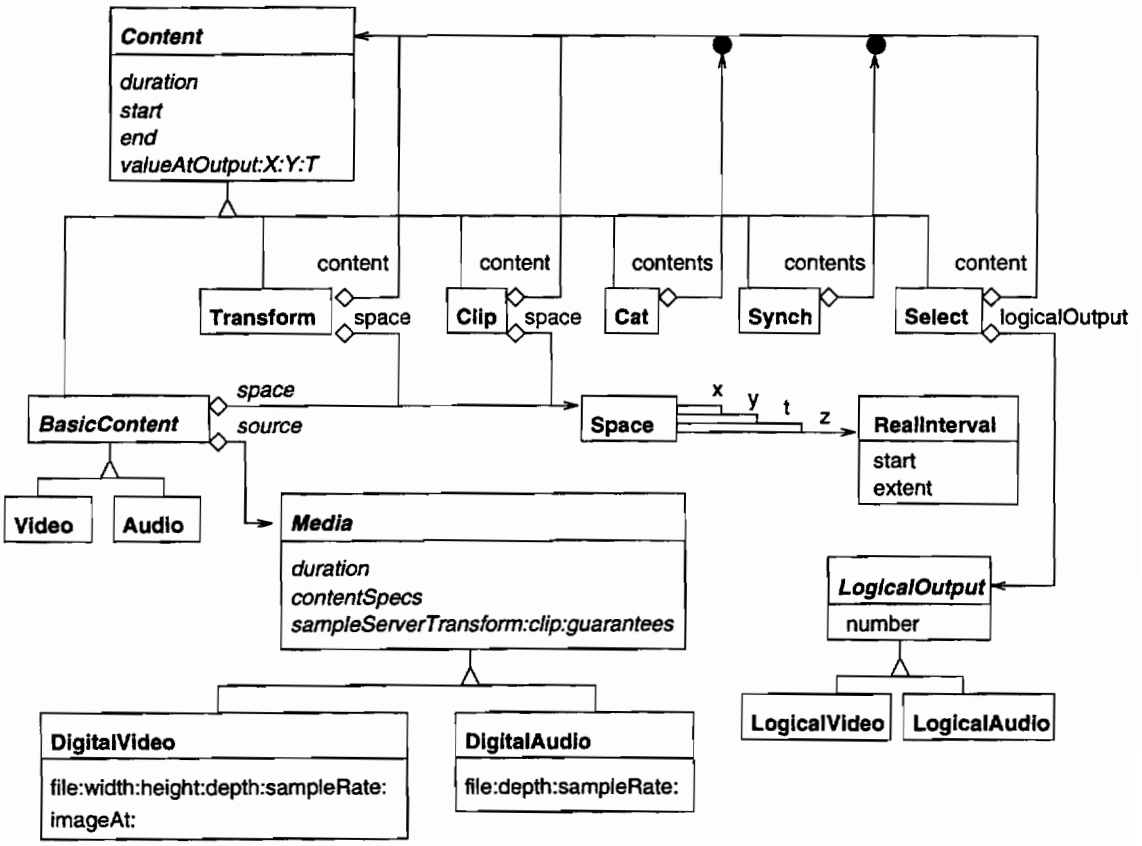

Figure 5.5: SQUINT content classes.

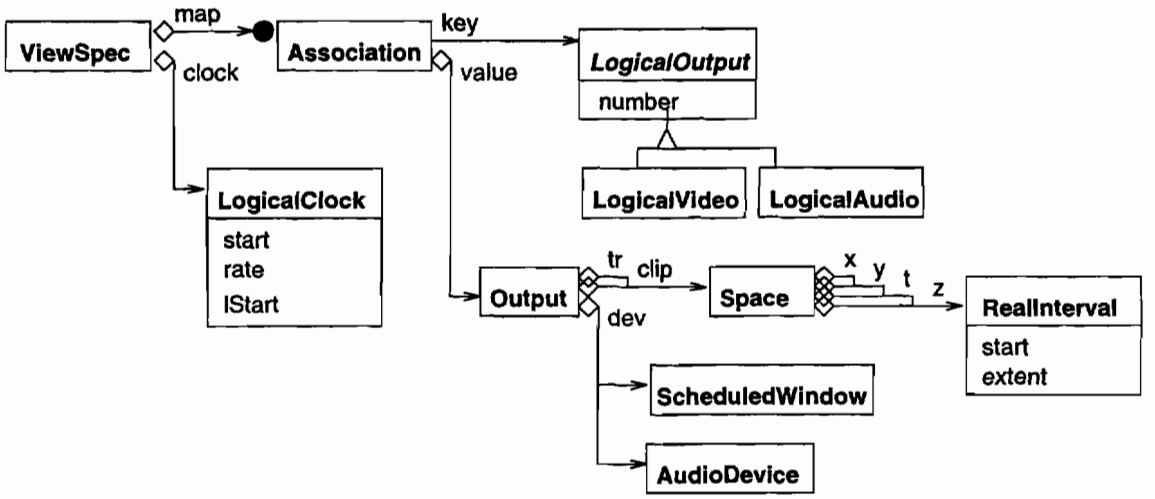

Figure 5.6: ViewSpec class.

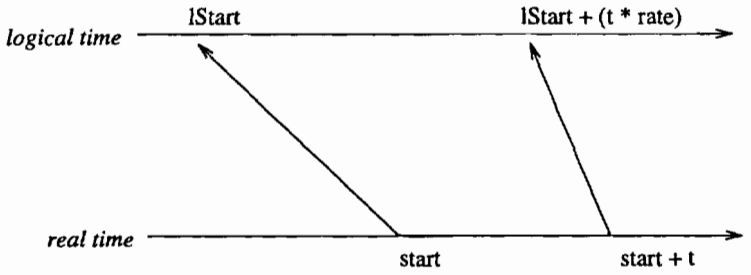

Figure 5.7: Mapping from real-time to logical-time.

in each video track. The output transform is the identity transform so that the logical coordinates are equivalent to window coordinates.

The time mapping for a view descriptor is defined by a `LogicalClock` instance. The default view sets the logical start time `lStart` to the start of the presentation and sets the `rate` to zero so that the presentation is stopped. Figure 5.7 illustrates the mapping from real-time to logical-time defined by the parameters `lStart`, `start`, and `rate`. The **position** slider shows the current logical time with the left and right end points of the slider corresponding to the content start and end, respectively.

The user can drag the **position** and **play rate** sliders at any time to redefine the view

specification's time mapping. The **start/stop** button toggles the logical clock's **rate** between zero and the current setting of the **play rate** slider. Adjusting the **position** slider manually causes the presentation to jump to the new logical time. Adjusting the **play rate** slider causes the presentation to speed up or slow down. Each change in the view specification triggers a recomputation of the presentation plan as described in the next section.

SQUINT also supports interactive control of the window dimensions in a view. Window resize events generated by the window manager are caught by SQUINT and are used to update the spatial output mapping in the view descriptor. These updates to the view also cause a recalculation of the presentation plan.

The generation of a default view for a newly selected content descriptor is a policy decision. SQUINT could just as easily leave the view descriptor unchanged when selecting new content. In that case, the video windows would simply display the video tracks of the new selection with the same display transformations, clipping and clock rate that the previous content had been playing. Another policy choice is what to do at the end of a presentation. There are three obvious choices: display null content, freeze the display with the last output value, or loop-back to restart at the beginning. SQUINT implements only a loop-back policy, but could be extended to offer a choice of policies.

5.2.3 Quality Calibration and Constraint

SQUINT uses the error model and quality estimation function described in the last chapter. An example of a `Quality` descriptor is shown in Figure 5.8. A `Quality` descriptor has an instance variable `min` indicating the minimum acceptable value for the quality estimation function. Recall from Chapter 4 that we define presentation quality to be the ratio of actual to ideal presentation utility. The value of `min` is set by the **quality** slider on the SQUINT control panel. Calibration values for the quality estimation function are stored in a structure of nested `Dictionary` objects and accessed through the `Quality` descriptor's `calibration` instance variable. Individual calibration values are retrieved by using error component attributes as arguments to the `Dictionary` lookup method `#at:`. For example, `((calibration at: Video) at: #jitter) at: #t` returns the calibration value for

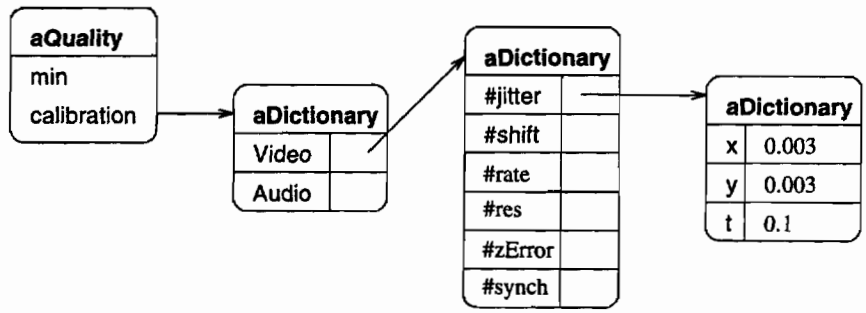

Figure 5.8: Quality class.

the time component of jitter on a video output. SQUINT sets this value to the reciprocal of the control panel's **sample rate** slider value. Values for `((calibration at: Video) at: #res) at: #x` and `((calibration at: Video) at: #res) at: #y` are set to the reciprocal of the **resolution** slider value. The reciprocal is used because the number of resolvable pixels in the image decreases as resolution error increases.

The remaining calibration values are the constants shown in Table 5.1. The units for temporal components of *jitter*, *shift*, *res*, and *synch* are in seconds. All values for spatial error in x and y are given relative to the size of the output window. The values for *zError* are also relative to the output z range. The *rate* error components represent the rate of change of the *shift* error in the appropriate units. For example, the rate of change of the x component of *shift* with respect to time is given in units of "window-widths per second". Values for the rate of change of the x component of *shift* are given in the first *rate* column and values for the y and t components are given in the next two columns. The calibration value for each error component was determined subjectively from a presentation of basketball video by increasing the component in question, while all other error components were negligible. The calibration value represents the point at which the error was judged to be "very annoying". The quality estimation function can be modified for other tasks by changing these values.

The same calibration values are used for all outputs with the same device type. This feature allows a quality descriptor to apply to views with any number of outputs, but also makes it impossible for SQUINT to specify that quality is more important in one video

		jitter	shift	rate x	rate y	rate t	res	zError	synch
Video	x	0.003	0.1	0.1	0.1	0.1	0.006	0.1	0.25
	y	0.003	0.1	0.1	0.1	0.1	0.006		
	t	0.1	15	0.01	0.01	0.5	0.1		
Audio	t	0.0002	15			0.2	0.0003	0.01	

Table 5.1: Calibration values for quality estimation function.

window than in another. To support different calibration values for each output, SQUINT could be extended to instantiate specialized calibration values as they are specified for each output.

The quality estimation function is hard-coded into the planning algorithm described in Section 5.3.

5.2.4 Representation of QOS Requirements

The content, view, and quality descriptors comprise the state of a `Player` instance. SQUINT treats the player as an abstract description of a presentation with two dependent objects. The first dependent object is the `ApplicationWindow`, which displays the current state of the control panel. The second dependent object is the `PresentationManager` that is responsible for displaying video frames and audio samples. Figure 5.9 illustrates a `Player` and its dependents. User changes to the player state via buttons, sliders, and typing, cause update messages to be sent to the dependents. Also, a real-time process advances the logical presentation time and sends update messages to the dependents. The dependents have access to the player's current state.

5.3 Presentation Planning

The planning algorithm uses a heuristic to choose the lowest quality presentation plan that satisfies the QOS specification. By mapping QOS requirements to a set of acceptable presentation plans, SQUINT is able to choose the plan with the least resource requirements. For example, if the image resolution required is 256x192 and the stored images

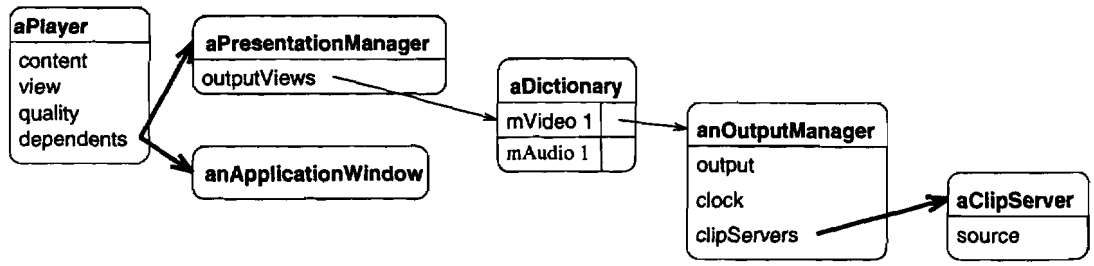

Figure 5.9: Player with dependent objects.

can be retrieved at either 320x240 or 640x480, SQUINT will choose the lower resolution with the assumption that it requires less bandwidth for storage access and transport.

5.3.1 Components of a Presentation Plan

A few new terms are needed to describe a presentation plan. The term *track* refers to a sequence of content displayed on a single output device. A *clip* in a presentation refers to content from a single source.

Figure 5.10 shows a tree of objects that make up a presentation plan. At the root of the tree, a `PresentationManager` is responsible for guaranteeing the QOS requirements of the player. It guarantees synchronization between outputs by requiring each of its children to synchronize with a common clock. The remainder of the QOS responsibilities are delegated to the `TrackManager` objects beneath it. Each `TrackManager` is responsible for the timing of samples written to a single output. Responsibility for the quality of video frames and audio samples is delegated to the `ClipServer` objects.

A presentation plan is created or modified each time the values of the content, view, or quality descriptors are changed. A `PresentationManager` is created only once for each instance of the SQUINT player. The `PresentationManager` creates a `TrackManager` for each output in the view specification. Each `TrackManager` is given a content descriptor for the track, a reference to the output descriptor, a real-time clock, and a request for minimum QOS guarantees. If any of the guarantees cannot be met then a guarantee-fail handler is invoked. In the current version of SQUINT, the guarantee-fail handler simply prints a diagnostic message and the presentation plan continues to execute in a best-effort

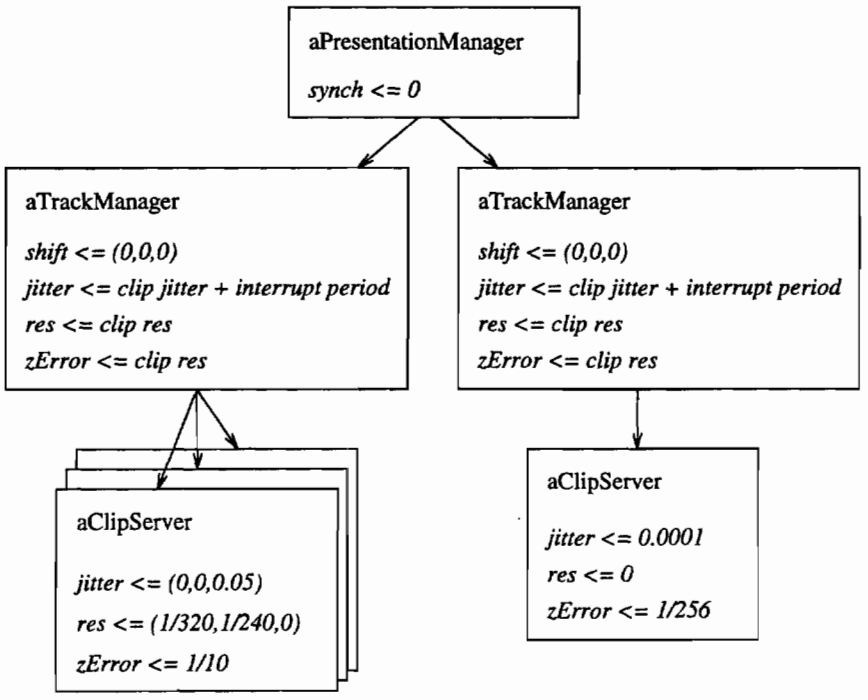

Figure 5.10: QOS guarantees.

mode.

The QOS guarantees are expressed as an upper bound for the magnitude of each error component. A `TrackManager` passes on the request for QOS guarantees to each `ClipServer` that it needs to supply data for the presentation. If all `ClipServer` objects grant the QOS request, then the `TrackManager` computes worst-case bounds for temporal *jitter* based on the `ClipServer` jitter guarantee and on the `TrackManager`'s own timing accuracy. Since SQUINT cannot predict process execution times or the scheduling behavior of the underlying operating system, the `TrackManager` objects make the optimistic assumption that execution times and scheduling delays are negligible. Violations of this assumption are detected at run-time and the guarantee-fail handler is invoked. To make timing guarantees in the admission test, the operating system and other resource managers would need to provide resource reservation protocols as discussed in Chapter 2.

A specialized `ClipServer` is created for each media source to retrieve, convert, and display data at the specified quality and in the format required by the output device.

Each `ClipServer` is created with a request for worst-case limits on all error. These guarantees may be determined from annotations on the source data and properties of the data transport and processing algorithms. The current `ClipServer` implementations account for error in the data sources, and for the introduction of error in spatial scaling, and in output timing. For example, a high-quality video might provide a resolution of 640×480 pixels, a signal-to-noise ratio of 100, and 30 fps. SQUINT creates a half-resolution version of a video by reducing each 2×2 block of pixels to one pixel with the average value of the block. A new guarantee for worst-case *zError* can be computed for each version of a video. Spatial scaling error is introduced when the ideal presentation requires fractional scaling, since SQUINT always rounds to integer scale factors. The maximum presentation frame rate is limited both by the recording rate of the source and by the bandwidth of the presentation platform. A `ClipServer` will introduce jitter through skipping source frames and through the imprecision of scheduling display events.

The definition of a `ClipServer` is intended to allow many types of specialization to suit a particular QOS request. For example, MPEG video decoding can be greatly simplified by the elimination of inter-frame dependencies. An MPEG video stream consists of a repeating pattern of **I**, **P**, and **B** frames, where **I** frames may be decoded independently, **P** frames require the previous **I** frame for decoding, and **B** frames require both the previous and next **I** or **P** frame for decoding. Suppose that an MPEG video file is encoded at 30 fps with the pattern **IBPB** so that the frequency of **I** frames is more than 7 fps. A playback request with a `T jitter` limit of 1/7 second could return a `ClipServer` that is specialized to read and decompress only **I** frames. In general, a `ClipServer` can be implemented by a pipeline of processes distributed across a network. Each stage of the pipeline modifies the guarantees of the previous stage with data format changes and new timing guarantees.

5.3.2 Admission Test

An *admission test* is a necessary part of any system that offers QOS guarantees. In SQUINT, the admission test involves two activities: calculating error limits and requesting guarantees from plan components. A set of error limits are called *satisfactory* if they satisfy the QOS requirements and *feasible* if they can be guaranteed by the components of some

presentation plan. SQUINT first calculates a set of satisfactory error limits, then attempts to build a presentation plan that will guarantee those limits. If any component cannot provide the requested guarantees, the admission test fails and a guarantee-fail handler is invoked.

An optimal solution would require testing every satisfactory set of error limits to see if it is feasible. Having found a set of plans that guarantee a satisfactory presentation, it would still be desirable to choose among them, the plan with the fewest resource requirements. This optimization problem reduces to the problem of scheduling a set of tasks with time and resource constraints, which is known to be intractable [91]. SQUINT instead employs a few simple heuristics to identify error limits that are likely to be feasible and to create a presentation plan that uses near-minimal resources.

The first heuristic is that an error interpretation based on the intended correspondence between the actual and ideal presentations is likely to provide a near-optimal estimate of presentation quality. Let the term *sample* refer to a discrete output value. We define the intended correspondence for each sample written to the output devices as follows. Suppose that every sample written by a `ClipServer` is annotated with the parent `TrackManager`'s *Output* descriptor, a timestamp indicating the ideal output time and duration for the sample value, and the *res* error and *zError* guarantees for the `ClipServer`. Suppose also that the annotations remain associated with the output location until the value is overwritten. The *SampleAnnotations* schema represents this information and the *sample* function returns the annotations for any *Device* and *Point*.

SampleAnnotations

$$output : Output$$
$$timestamp : Interval$$
$$res : Point$$
$$zError : R$$

$$sample : Device \longrightarrow Point \longrightarrow SampleAnnotations$$

The following definition for a minimum difference d_{min} helps to define the smallest

interpretation of temporal jitter when samples have a non-zero ideal display duration. The difference $d_{min}\ i\ t$ between an interval i and a time t is zero if t is contained in i. If t is before the interval than the minimum difference is the interval start minus t. If t follows the interval than the minimum distance is the interval end minus t.

$$
\begin{array}{|l}
d_{min} : Interval \longrightarrow R \longrightarrow R \\
\hline
\forall\, i : Interval; t : R \bullet \\
\quad (t \in I\ i \Rightarrow d_{min}\ i\ t = 0) \\
\quad \wedge\ (t < i.start \Rightarrow d_{min}\ i\ t = i.start - t) \\
\quad \wedge\ (t \geq i.start + i.extent \Rightarrow d_{min}\ i\ t = i.start + i.extent - t)
\end{array}
$$

The error interpretation used by the admission test is called ε_S and is defined as follows:

$$
\begin{array}{|l}
\varepsilon_S : ErrorInterpretation \\
\hline
\varepsilon_S.shift\ d\ p = (0,0,0) \\
\varepsilon_S.rate\ d\ p = ((0,0,0),(0,0,0),(0,0,0)) \\
\varepsilon_S.synch\ d\ d' = 0 \\
\forall\, d : Device; p : Point \bullet \\
\quad (LET\ s == sample\ d\ p; \\
\qquad constrained == (\exists\, z : R \bullet \neg\ ((d, p.x, p.y, p.t, z) \in ideal\ \varepsilon_S.c\ \varepsilon_S.v)) \bullet \\
\quad constrained \Rightarrow (\varepsilon_S.jitter\ d\ p = (0,0,d_{min}\ s.timestamp\ p.t) \\
\qquad \wedge\ \varepsilon_S.res\ d\ p = (s.res.x, s.res.y, 0)) \\
\quad \wedge \neg\ constrained \Rightarrow (\varepsilon_S.jitter\ d\ p = (0,0,0) \wedge \varepsilon_S.res\ d\ p = (0,0,0) \\
\qquad \wedge\ \varepsilon_S.zError\ d\ p = 0))
\end{array}
$$

Since SQUINT does not allow timing error to accumulate, the t component of *shift* is always interpreted as zero. The *rate* and *synch* error components are also zero everywhere since they depend only on the interpretation of *shift*. We could use a simpler error model without these error components, but it is comforting to observe that an expressive error

model does not force a complicated interpretation of error. The x and y components of *shift* and *jitter* are zero because SQUINT accurately maps spatial coordinates. This statement is not quite true for views that call for non-integral scale factors for video rendering, but the definition of ε_S can be extended to describe these cases also. In the interest of brevity, this description of ε_S assumes that rendering requires only integral scale factors. For the remaining error components, ε_S distinguishes between points that are constrained in the ideal presentation and points that are allowed to have any value. For constrained points, the t component of *jitter* is defined as the difference between the current time and the timestamp associated with the value at each point. If a sample is intended to have a non-zero duration, then jitter is zero during the timestamp interval. The interpretation of *res* is taken directly from the resolution associated with each sample except that the t component of *res* is always zero. The interpretation of *zError* is defined by the schema for an *ErrorInterpretation*, but for any *Device* d and *Point* p, (*sample* d p).*zError* provides an upper bound on the magnitude of ε_S.*zError* d p. For points that are unconstrained, all error components are zero.

Recall from Chapter 4 that a presentation satisfies a QOS specification if, for some error interpretation, the value of the quality estimation function is everywhere greater than *min*. The admission test for SQUINT is more strict: the presentation plan must guarantee acceptable worst-case error bounds for the error interpretation ε_S. Of course, ε_S was chosen to be near-minimal in its worst-case behavior.

The second heuristic employed by SQUINT is that if any set of error limits for ε_S is feasible, then a set is likely to be feasible in which *variable error components* contribute equally to lowering the quality estimate. A set of error limit values for each output are expressed with the *ErrorLimits* schema.

ErrorLimits

jitter : *Output* $\longrightarrow$ *Point*

shift : *Output* $\longrightarrow$ *Point*

rate : *Output* $\longrightarrow$ *Matrix*

res : *Output* $\longrightarrow$ *Point*

zError : *Output* $\longrightarrow$ *R*

synch : *Output* $\longrightarrow$ *Output* $\longrightarrow$ *R*

The worst-case behavior of an error interpretation ε_S is bounded by a set of error limits l if, for any *Device* d and *Point* p, each error component in ε_S is less than or equal to the corresponding limit for the output in *sample* d p. For example, $(\varepsilon_S.jitter\ d\ p).t$ must be less than or equal to $(l.jitter\ (sample\ d\ p).output).t$.

From Chapter 4, the quality estimation function is $e^{-\varepsilon_{norm}}$, where ε_{norm} is the magnitude of a vector of all error components, each divided by the corresponding weights. A set of error limits is satisfactory if the quality estimate using limit values for the error components is greater than the value of *min* specified in the `Quality` descriptor. This requirement can be rewritten as

$$\sqrt{(\sum_{i \in Names_R} (\frac{l.i}{\omega.i})^2)} \leq -ln(q.min) \tag{5.1}$$

where the $l.i$ are the error limit values and the $\omega.i$ are the corresponding weights for each error component name i in M_R. Let l_S be the satisfactory set of *ErrorLimits* chosen by the admission test. For outputs not specified in the view all error limits are zero. For the error interpretation ε_S, the worst-case error for *shift*, *rate*, *synch*, the t component of *res*, and x and y components of *jitter* are all zero. Consequently, SQUINT sets the error limits for these components to zero. The worst-case value $l_S.zError\ o$ for an output o is determined by the media sources for that output. Only the x and y components of *res* error and the temporal component of *jitter* are variables of the presentation plan.

By applying the second heuristic, each of the terms for x and y *res* error and t *jitter* error for this track are equal to some value v. Then Equation 5.1 can be rewritten with

these substitutions and solved for the value of v:

$$\sqrt{(3 * v + (\frac{l_S.zError\ o}{q.zError\ o})^2)} \leq -ln(q.min) \tag{5.2}$$

and

$$v \leq \frac{1}{3}((ln(q.min))^2 - (\frac{l_S.zError\ o}{q.zError\ o})^2) \tag{5.3}$$

Since the weights for the x and y components of *res* error on the track are $(q.res\ o).x$ and $(q.res\ o).y$ respectively, and the weight for the t component of *jitter* is $(q.jitter\ o).t$, SQUINT chooses the following values to complete a set of satisfactory error limits for the track:

$$(l_S.res\ o).x = (q.res\ o).x * \sqrt{v}$$
$$(l_S.res\ o).y = (q.res\ o).y * \sqrt{v}$$
$$(l_S.jitter\ o).t = (q.jitter\ o).t * \sqrt{v}$$

These error limits are used to request guarantees when creating the `ClipServer` components of a presentation plan.

If these error limits are not feasible, it may still be possible to obtain guarantees by relaxing one or more limits and tightening the rest to compensate. Since there are three variable error components, the limits for two could be reduced to zero, allowing the third to be relaxed up to a factor of $\sqrt{3}$. However, this range is not likely to greatly improve the chances of finding a feasible set of error limits.

The third heuristic employed by SQUINT is that a plan with weaker error guarantees is likely to use fewer resources. In the prototype, resource requirements are strictly increasing with both resolution and sample rate. Consequently, SQUINT finds the plan with near-minimal resource requirements by selecting the viable plan components with the weakest error guarantees.

5.3.3 Proof of QOS Guarantees

SQUINT interprets a press of the **start/stop** button as a request to begin a presentation immediately. Unfortunately, the error interpretation ε_S forces SQUINT to interpret any

start-up delay as jitter. To allow time for presentation planning, we can add a new error component to the reference error model. Let *response* error be the time that it takes to create and initialize a new presentation plan. During this startup period, all other error components may be interpreted as zero.

Claim: Every presentation that passes the admission test satisfies the player's QOS requirements.

Proof: It has already been shown that the error interpretation ε_S satisfies the *QOS* predicate if its worst-case behavior is bounded by the set of error limits l_S. It remains to be shown that a successful admission test produces plan components that guarantee these error limits.

Part of this proof is trivial, since *shift*, *rate*, *synch*, and x and y components of *jitter* are zero by definition in ε_S. All plans produced by the admission test assume this error interpretation and therefore guarantee that these error components are everywhere equal to the error limit of zero specified in l_S. Also, any plan trivially guarantees zero error for every device d and point p where the ideal presentation is unconstrained. The non-trivial part of the proof is to show that $\varepsilon_S.zError$, the x and y components of $\varepsilon_S.res$, and the t component of $\varepsilon_S.jitter$ are less than the error limits for all device coordinates constrained in the ideal presentation.

Let d be any device and p be any point constrained by the ideal presentation. There is a finite set of outputs in the view (typically only one) that define a mapping of constraints onto d at p. Let O be that set. If the *response* error limit has not expired at time $p.t$ then all errors are ignored and the guarantees are considered satisfied. If $p.t$ is beyond the *response* error limit and some `TrackManager` has not updated its output then SQUINT invokes the guarantee-fail handler with a timing error. Recall that SQUINT detects timing errors only at run-time, so a timing-guarantee failure is considered a late failure of the admission test. If $p.t$ is beyond the *response* error limit and all `TrackManagers` have updated their outputs, then we need to prove that the output sample value at p satisfies the remaining QOS requirements.

Since all outputs have been written, the sample value of d at p is considered annotated with $s = sample\ d\ p$, where $s.output \in O$. Our hypothesis asserts that the acceptance

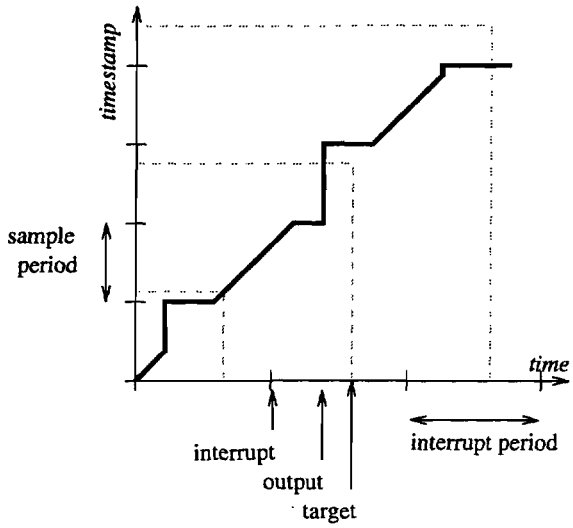

Figure 5.11: Determination of worst-case jitter between updates.

test was successful, therefore the `ClipServer` that wrote this sample guarantees that $\varepsilon_S.zError\ d\ p \leq l_S.zError\ o$ with $\varepsilon_S.res\ d\ p \leq l_S.res\ o$. Finally, the t component of $\varepsilon_S.jitter\ d\ p$ is $d_{min}\ s.timestamp\ p.t$. Every time a `TrackManager` writes an output value, it computes a worst-case analysis of temporal jitter since the last write. If the magnitude of $(\varepsilon_S.jitter\ d\ p).t$ could have exceeded $(l_S.jitter\ o).t$ during this interval than the guarantee-fail handler is invoked.

The protocol for creating a `TrackManager` guarantees only that the jitter limit will not be exceeded when there are no scheduling delays. Figure 5.11 illustrates the parameters that determine jitter. A periodic software interrupt is used by each `TrackManager` for updates. At each interrupt, the `TrackManager` requests a sample from the appropriate `ClipServer` for a target time midway between interrupts. Each `ClipServer` guarantees that it will provide a sample with a timestamp not more than one-half the source sample period from the target time.

Let t_s be the sample period, t_p the interrupt period, t_o the time that an output was generated after an interrupt at t_i, and t_d the start of the output sample timestamp. Let t_h be the maximum time to handle an interrupt. The following inequalities are true for the definitions above.

$$t_i + 0.5t_p - 0.5t_s \leq t_d \tag{5.4}$$

$$t_d \leq t_i + 0.5t_p + 0.5t_s$$

$$t_i \leq t_o$$

$$t_o \leq t_i + t_h$$

Timestamps for digital video in SQUINT have zero duration, so $\varepsilon_S.jitter$ is simply $t_d - t$. Since jitter decreases monotonically between outputs, the maximum value between outputs occurs at the start of the interval. Setting $t = t_o$ makes jitter $t_d - t_o$, which is always less than or equal to $0.5(t_p + t_s)$. The minimum value for jitter occurs just before the next output. Setting $t = t'_o$, where the prime indicates a value associated with the next interrupt, makes jitter $t'_d - t'_o$, which is always greater than or equal to $0.5(t_p - t_s) - t_h$.

SQUINT uses these equations to compute the largest acceptable values for t_p from the temporal jitter error limit. Taking $t_h = t_p$ and recalling that the jitter limit for the sample in question is $(l_S.jitter\ s.output).t$ gives:

$$0.5(t_p + t_s) \leq (l_S.jitter\ s.output).t \tag{5.5}$$

or

$$t_p \leq 2(l_S.jitter\ s.output).t - t_s \tag{5.6}$$

With this value for the interrupt period, each `TrackManager` guarantees that temporal jitter will not be exceeded as long as each interrupt is handled before the next occurs.

This completes the proof that SQUINT's admission test guarantees the QOS specified in the player.

How does SQUINT's admission test compare with other multimedia systems? Most of the systems surveyed in this thesis make guarantees based on bandwidth requirements for a particular media representation [56, 12]. They lack a means for expressing QOS requirements independent of the data type. Other systems that support scalable presentation quality do not provide complete guarantees [80, 21]. SQUINT's admission test supports

QOS specifications that are scalable and independent of device and data representations. These QOS specifications are translated into presentation plans that are guaranteed not to exceed the error constraints. SQUINT's admission test demonstrates that a formal approach to QOS management can be implemented efficiently, at least in simple cases.

5.4 Presentation Execution

In a sense, the SQUINT multimedia player is executing a presentation at all times. When the view's `rate` is zero, the presentation is considered stopped, but the presentation view must still display the video frames corresponding to the current logical time. A stopped presentation is simply a presentation in which logical time does not advance. When the `rate` parameter becomes non-zero, logical time begins to advance at the specified rate and the presentation view must be updated accordingly.

Any change to the player's QOS specification causes the `PresentationManager` to compute a new presentation plan and to begin executing it. The controls for QOS specification have been described in Sections 5.2, 5.2.2, and 5.2.3. When the position or rate are changed, the `Presentation view` does not create new `TrackManager` and `ClipServer` components, but instead merely broadcasts a message to the existing components that the logical clock's time mapping has changed. Each component of the plan reads the new mapping to determine what the current sample should be. Interactive changes to the minimum quality specification cause a re-computation of allowable error limits and these new limits are used to request new `ClipServers` with sufficient guarantees for resolution, image noise, and real-time sampling rate.

5.4.1 Resource Overload Detection and Handling

During execution, the `TrackManager` components compute the actual jitter for video displays at each output event. The timing error for an output event is just the difference $t_{actual} - t_{ideal}$. Presentation authoring specifies the value of t_{ideal} when an output event should occur. The value of t_{actual} can be determined approximately at runtime as illustrated in Figure 5.12. The measurement accuracy for t_{actual} is limited by the resolution of

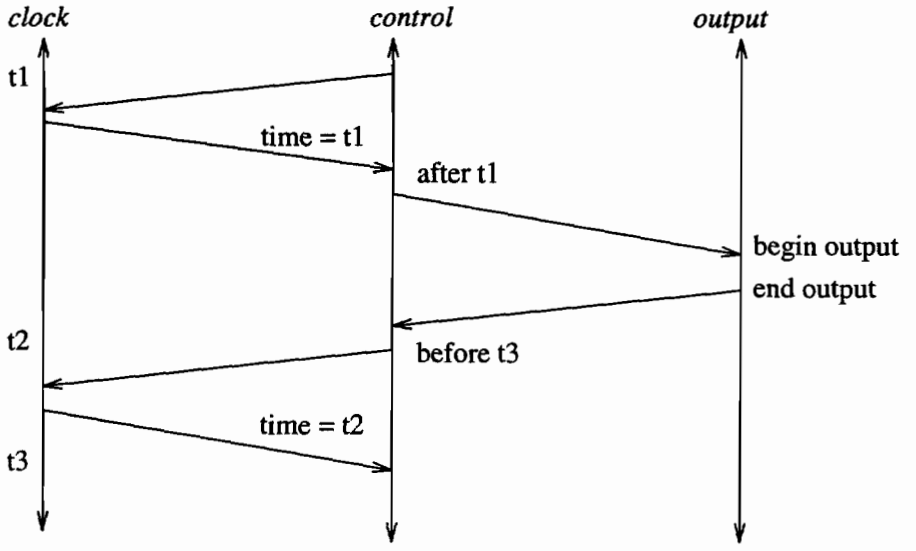

Figure 5.12: Computing bounds for timing error.

the clock, by the latency of clock reading operations, and by the duration of the output operation. Accepting this uncertainty, it is still possible to compute a bound on timing error from the range of possible values for t_{actual}.

Each `TrackManager` reads the clock before and after each frame is displayed and computes the worst-case jitter since the previous frame was displayed. A guarantee violation handler is invoked when jitter limits are exceeded. Currently, the handler simply displays a message saying that jitter limits were exceeded.

5.5 Discussion

SQUINT illuminates some of the complexity of using formal QOS specifications to drive resource management. In general, a multimedia player will support interactive views on content that may be distributed over a variety of storage devices. Our experience with SQUINT suggests that media objects should support a protocol for creating specialized clip servers with full QOS guarantees. These clip servers may be organized internally as pipelines or even trees for distributed media access. A track manager, or some process within a clip server may assume ultimate responsibility for the output timing. SQUINT

demonstrates that simple heuristics allow fast translation of QOS requirements into reasonable resource scheduling parameters.

The decision to synchronize all controllers with a single clock eliminates many of the synchronization concerns that have been raised by other researchers [41, 34, 58]. The use of a fixed schedule allows every component of the presentation plan to work independent of the others while still achieving good overall synchronization. The danger of this approach is that processing delays result in skipped segments of audio and video. The correct choice of whether to preserve a fixed schedule or to preserve information content is application dependent. Our QOS specification model allows expression of such a choice through the quality estimation function's calibration values.

The major goal for SQUINT was to test the practicality of using formal QOS specification in a multimedia player. The informal proof of correctness for the admission test demonstrates that meaningful execution guarantees can be derived from a formal QOS specification. While it seems impractical to provide a proof of correctness for every extension of a multimedia player, the existence of a formal semantics for the QOS specifications is useful for understanding the goals of a presentation. It seems clear that the methodology used to build and validate SQUINT can be vastly improved, increasing the benefits of the formal specification approach.

The problem of implementing a presentation plan without exposing performance is called the *mapping dilemma* and the use of QOS specifications in a request for multimedia services is an example of a meta-protocol [35]. Without a QOS specification, a multimedia system does not have enough information to control performance aspects of a presentation. The result is that inappropriate implementation choices provide unacceptable performance. A QOS specification describes performance requirements so that the implementation can be specialized for each request.

Chapter 6

Related Work

6.1 QOS Specification

Much of the literature on QOS specification focuses on Continuous Media (CM) data transport services. Anderson identifies the following seven parameters that can be used for reserving a continuous-media transport session: maximum message size, maximum message rate, input workahead limit, output workahead limit, maximum logical delay, minimum actual delay, minimum unbuffered actual delay [2]. These parameters are defined in terms of the CM-resource model and are used to derive end-to-end guarantees for real-time transport of an abstract stream of "messages". This work supplies an important analysis of techniques for bounding jitter in continuous media transport but does not address other user-level QOS issues such as scalable image quality. Other QOS parameters and algorithms have been described for reservation of file system [5, 47, 61, 84], CPU [80, 51], and network capacity [22, 90, 42, 3, 86, 89]. All of these resource reservation approaches characterize bandwidth as a QOS parameter. We do not include bandwidth in presentation QOS specifications because it depends on implementation choices for data encoding. In particular, the physical bandwidth for a media stream will vary at each stage of a pipeline where the data is compressed, decompressed, or filtered for transport and display requirements. By specifying only presentation output behavior and not implementation, our specifications are device and data independent.

The Multimedia System Services (MSS) architecture defines a set of "core QOS characteristics" consisting of the following parameters:

- guarantee level: Guaranteed, Best Effort, or No Guarantee

- reliable: True or False
- delay bounds: minimum and maximum delay
- jitter bounds: minimum and maximum delay variance
- bandwidth bounds: minimum and maximum bandwidth

The MSS architecture was developed by the Interactive Multimedia Association, an organization with representatives from Hewlett-Packard (HP), IBM and SunSoft [36]. As with earlier work, their QOS parameters do not address scalable image quality and are oriented toward resource reservations rather than user-level QOS.

The Multimedia Projects Group at Lancaster University is developing the Quality of Service Architecture (QOS-A) for multimedia communications [11]. This architecture explicitly recognizes the need for distinct QOS characterizations at each level of the protocol stack. The objectives of the QOS-A project are to define an OSI-compatible architecture for QOS management in an Open Distributed Processing (ODP) multimedia environment. Application-layer QOS parameters are encapsulated within a set of commonly used "channel types". New channel types are created by providing a QOS-mapper service to translate from the channel type name into a QOS specification for the next layer down. As an example, the channel type "StandardVideo" is mapped to the following parameters: bandwidth=25 Mbps, jitter = 10 ms, delay = 250 ms, traffic type = probabilistic, and error rate = 10^{-3}. The user is able to choose among these standard channel types, possibly even selecting a different channel type during a presentation. Our work extends this architecture by offering a continuum of channel types. The mapping from our QOS specifications to low-level QOS parameters can be accomplished by simple heuristics such as those employed in SQUINT, or by more complex algorithms that take into account the current resource availability to provide the best quality.

Our work is also distinguished from previous approaches by our methodology for choosing QOS parameters. Our framework breaks the QOS specification problem into two parts: definition of an error model and specification of acceptable quality in terms of the error

model. We have described a completeness criteria for the definition of error model components. This methodology is similar to the formal theory for Epsilon Serializability in transaction processing [62]. Using the relational data model, ESR proposes a general theory for defining a transaction error metric and using that metric to determine when locking requirements can be relaxed.

Whether QOS is specified through a set of ad hoc parameters or through a formal model as we have proposed, a correct specification of quality requirements depends on the purpose of a presentation and on human perception.

Higgins describes some of the factors that determine human perception of image quality [31]. Objective measures are given for tone reproduction, sharpness, and graininess. His definition of these quantities constitutes an error model for still images that is more expressive than the reference error model proposed in Chapter 4. In particular, tone reproduction conveys information about contrast errors and brightness shift, while our model can only express local differences in tone. While these measures allow more accurate user-models for still images, Higgins does not report estimates of image quality from a combination of these values.

Limb reports experiments in which the subjects rated a set of images by how annoying the perceived distortions were [40]. The subjective evaluations of distortion are correlated with root-mean-square-error and other objective measures. His success at creating a crude quantitative model for image fidelity suggests that useful empirical models can be determined for other multimedia presentation tasks.

Other researchers have reported empirical determinations of acceptable quality for various types of presentation error. Steinmetz documents the perceived level of annoyance as a function of synchronization error between audio and video [75]. His results argue for acceptable values of synchronization error between -80 ms and +80 ms when users are watching a moderate close-up of a person talking.

6.2 Presentation Planning

Most existing multimedia systems do not have the capability to select different QOS levels for the same content. However, future multimedia system will incorporate technology for scalable video quality and other tradeoffs between presentation quality and resource use [16, 17, 14, 21]. Other researchers are only now beginning to solve the problem of how to choose from among many possible presentation plans.

Nahrstedt and Smith have proposed the QOS Broker technique for presentation planning [56]. Application QOS requirements are input to a broker-buyer which translates them into requirements for local and remote resources. The broker-buyer first negotiates with the local operating system for local resources, rejecting or modifying the application requirements if sufficient resources are unavailable. Only when the local resources are reserved does the broker-buyer begin negotiations with a remote broker-seller for remote resources. The local bandwidth reservations determines the appropriate bandwidth to request from the remote broker. Finally, after both local and remote OS resources have been reserved, the broker-buyer requests appropriate communications channels from the network subsystem. Nahrstedt and Smith have implemented a prototype of the QOS Broker with a telerobotics application. The application QOS requirements are expressed in terms of sample size, sample rate, loss rate, and end-to-end delay. The translation of the application QOS parameters into network QOS requirements is relatively straightforward for fixed-sized samples.

The Circus multimedia environment from GTE Labs features a blackboard approach for orchestrating resource management [27]. Distributed elements that provide or require multimedia services communicate through a global blackboard where the Orchestrator attempts to configure optimal connections between them.

The AMOS Multimedia Playout Manager allows integration of multimedia data in a distributed database management system [78]. Physical storage and access for continuous media are supported by specialized services that can perform adaptive prefetching to make data available on demand in a client's local buffer. A goal of AMOS' adaptive playout management scheme is to consider user-specific sensitivity to presentation deficiencies.

Scalable media quality is currently supported by redundantly storing the same content with different compression factors. QOS goals are expressed in terms of sample rates and sample depth. This is a data-encoding-dependent approach to QOS specification that makes it difficult to guarantee the actual quality of a presentation. Our approach to QOS specification allows a complete specification of requirements and data independence.

Software feedback techniques have been used to dynamically adjust stream processing workloads to available system bandwidth [10, 13, 66, 80]. Our quality estimation function can be used with feedback techniques to optimize a presentation for the current resource availability. For example, a presentation manager can monitor each of the presentation error components at runtime. Network and processor bandwidth overloads are detected by missed deadlines for display events [13]. Many of the techniques described in Chapter 2 can reduce bandwidth requirements, including switching to a more highly compressed data source or skipping video frames. The response to overload detection should be an adaptation of the presentation plan to reduce bandwidth requirements. If each component of a presentation plan can predict the error in its outputs, then our quality estimation function can be used to drive the adaptation by indicating which new presentation plan is likely to deliver the best presentation quality. Useful predictions of presentation error are possible for clip servers based on source attributes and the assumption that reduced bandwidth requirements will nearly eliminate missed deadlines. However, the predictions may prove false if adaptations do not affect the bottleneck resources. The absence of overload detection may be used as a signal to increase bandwidth requirements in an attempt to improve presentation quality. Our quality estimation function can be used to drive this adaptation as well.

Chapter 7

Conclusions

This thesis has described a new framework for QOS specification in multimedia systems and provides a concrete example of useful QOS specifications with formal semantics. The primary contributions of our specification semantics are the orthogonal definitions of *content*, *view* and *quality* descriptors. These definitions support device independent and physical-data independent authoring, playback, and requests for presentation quality. The SQUINT multimedia player demonstrates that our QOS specifications can be used to satisfy a diverse mix of multimedia service requirements.

7.1 A Framework for Defining Formal QOS Semantics

Chapter 4 described a formal QOS specification semantics that can be used to provide presentation guarantees. The key precondition for optimal resource management in multimedia systems is to identify a metric for presentation quality. The methodology we used to define such a metric consists of three major steps. First, define an ideal presentation. Second, choose an error model that describe the difference between actual and ideal presentations. Third, define a quality estimation function in terms of the error model. Chapter 4 identifies completeness and soundness criteria to help in defining useful error models. This methodology distinguishes our work from other descriptions of presentation-level QOS parameters.

The content descriptors defined in Chapter 4 allow a physical-data independent specification of logical content and the view descriptors define a device independent mapping of

logical content onto an ideal presentation. The quality descriptor preserves this physical-data and device independence by specifying presentation output behavior rather than implementation. Physical-data and device independence increase the portability of a multimedia application by allowing it to use the same request for presentation functionality on any platform.

Our content descriptors were designed to abstract away or eliminate features that distract from the goal of QOS specification while still supporting complex and useful authoring tasks. We found that a very small set of operations could satisfy this goal. The result is a stripped down model of multimedia authoring that may provide a useful base for serendipitous investigations. Another deliberate property of the definitions is complete orthogonality of content and view descriptors. For example, the author-specified size and layout of video windows can be customized in a view to suit the requirements of a playback application. Content and view can be specified independently and reused: the same content appearing in many different views and the same view displaying many different content descriptors. This orthogonality extends to the quality descriptor as well. A single quality descriptor can be determined for a class of applications and reused with many different content and view descriptors.

The declaration of an *ErrorInterpretation* defines a particular error model for describing the relation between an actual and an ideal presentation. The error component names suggest familiar concepts, but the formal definition of these error components is new.. In particular, our model defines temporal jitter as all the timing error that is not shift (delay) error, but allows multiple interpretations of timing error and shift error for a given presentation. We found that a unique definition for jitter requires knowledge of a presentation's implementation. By abandoning an implementation-based definition of error, our QOS specifications gain device and physical-data independence. Such QOS specifications allow a player freedom to choose an optimal implementation according to current resource availability and cost.

7.2 An Architecture for Resource Optimization with QOS Guarantees

Chapter 5 describes an architecture for QOS-based resource optimization. The SQUINT multimedia player provides a concrete example of this architecture and demonstrates that simple heuristics allow fast translation of our QOS specifications into conservative resource scheduling parameters. The key components of the architecture are the *player* that defines the QOS specification, a *presentation manager* that reacts to changes in the QOS specification, *track managers* that interpret the specification for a given output, and specialized *clip servers* that each supply data from a single source. The track managers and clip servers support an admission test protocol for QOS guarantees.

The proof of correctness for the admission test demonstrates that meaningful execution guarantees can be derived from a presentation-level QOS specification. SQUINT does not provide a priori guarantees for resource scheduling, but it does guarantee a viable presentation plan. SQUINT reduces CPU and file system usage in response to relaxed QOS requirements. This feature allows better control of resource allocation in shared environments, such as the digital television studio described in Chapter 3.

7.3 Future Work

The survey of QOS management techniques in Chapter 2 should be extended to discuss transport protocols for distributed communications and their effect on presentation quality. Despite its incomplete scope, the survey identifies a large space of variables for the system designer, including data location, compression, prefetching and reservation techniques. SQUINT makes use of only two forms of compression for scaling quality and resource management. Multimedia players that use more of these techniques face increased planning complexity and will require more sophisticated heuristics. In particular, distributed resource reservation algorithms are needed for reliable access to remote data and network resources.

Empirical studies of user task performance are needed to improve the quality estimation function. SQUINT relies on user interface controls to define QOS requirements. It

would be nice for a player to infer QOS requirements automatically from the application mode, perhaps with some consideration of the content and view descriptors.

Bibliography

[1] ABBOTT, R., AND GARCIA-MOLINA, H. Scheduling real-time transactions. *SIGMOD Record 17*, 1 (March 1988), 71–81.

[2] ANDERSON, D. P. Metascheduling for continuous media. *ACM Transactions on Computer Systems 11*, 3 (August 1993), 226–252.

[3] ANDERSON, D. P., HERRTWICH, R. G., AND SCHAEFER, C. SRP: A resource reservation protocol for guaranteed-performance communication in the Internet. *Tech. Rep.* TR-90-006, International Computer Science Institute, February 1990.

[4] ANDERSON, D. P., AND HOMSY, G. A continuous media I/O server and its synchronization mechanism. *Computer 24*, 10 (October 1991), 51–57.

[5] ANDERSON, D. P., OSAWA, Y., AND GOVINDAN, R. A file system for continuous media. *ACM Transactions on Computer Systems 10*, 4 (November 1992), 311–337.

[6] BERSON, S., GHANDEHARIZADEH, S., MUNTZ, R., AND JU, X. Staggered striping in multimedia information systems. In *SIGMOD 94, Minneapolis, MN* (May 1994), pp. 79–90.

[7] BLAIR, G., CAMPBELL, A., COULSON, G., DAVIES, N., GARCIA, F., AND SHEPHERD, D. Summary of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video. *Operating Systems Review 28*, 2 (April 1994), 22–33.

[8] BLAKE, B. A., AND SCHWAN, K. Experimental evaluation of a real-time scheduler for a multiprocessor system. *IEEE Transactions on Software Engineering 17*, 1 (January 1991), 34–44.

[9] CABRERA, L.-F., AND LONG, D. D. E. Swift: A storage architecture for large objects. In *Proceedings of the 11th IEEE Symposium on Mass Storage Systems* (1991), pp. 123–129.

[10] CAMPBELL, A., COULSON, G., GARCIA, F., AND HUTCHISON, D. A continuous media transport and orchestration service. In *SIGCOMM '92, Computer Communcation Review* (August 1992), vol. 22, ACM Press, New York, pp. 99–110.

[11] CAMPBELL, A., COULSON, G., GARCIA, F., HUTCHISON, D., AND LEOPOLD, H. Integrated quality of service for multimedia communications. In *Proceedings IEEE INFOCOMM '93* (San Francisco, USA, April 1993), IEEE.

[12] CAMPBELL, A., COULSON, G., AND HUTCHISON, D. A quality of service architecture. *Computer Communication Review 24*, 2 (April 1994), 6–27.

[13] CEN, S., PU, C., STAEHLI, R., AND WALPOLE, J. A distributed real-time MPEG video audio player. In *NOSSDAV 95* (1995), vol. 1018 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 151–162.

[14] CHADDHA, N., WALL, G. A., AND SCHMIDT, B. An end to end software only scalable video delivery system. In *NOSSDAV 95* (April 1995), vol. 1018 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 139–150.

[15] CHRISTODOULAKIS, S., AND FALOUTSOS, C. Design and performance considerations for an optical-disk based, multimedia object server. *IEEE Computer 19*, 12 (Dec 1986).

[16] CKER CHIUEH, T., AND KATZ, R. H. Multi-resolution video representation for parallel disk arrays. In *ACM Multimedia 93* (August 1993), pp. 401–410.

[17] DELGROSSI, L., HALSTRICK, C., HEHMANN, D., HERRTWICH, R., KRONE, O., SANDVOSS, J., AND VOGT, C. Media scaling for audiovisual communication with the Heidelberg Transport System. In *ACM Multimedia 93* (August 1993), pp. 99–104.

[18] DELGROSSI, L., HALSTRICK, C., HEHMANN, D., HERRTWICH, R. G., KRONE, O., SANDVOSS, J., AND VOGT, C. Media scaling in a multimedia communication system. *Multimedia Systems 2*, 4 (October 1994), 172–180.

[19] DRAPEAU, G. D. Synchronization in the MAEstro multimedia authoring environment. In *ACM Multimedia 93* (August 1993), pp. 331–340.

[20] DRUSCHEL, P., ABBOTT, M. B., PAGELS, M. A., AND PETERSON, L. L. Network subsystem design. *IEEE Network 7*, 4 (July 1993), 8–17.

[21] ELEFTHERIADIS, A., AND ANASTASSIOU, D. Meeting arbitrary QoS constraints using dynamic rate shaping of coded digital video. In *NOSSDAV 95* (April 1995), vol. 1018 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 95–106.

[22] FERRARI, D. Real-time communication in an internetwork. *Journal of High Speed Networks 1*, 1 (1992), 79–103.

[23] GALL, D. L. MPEG: A video compression standard for multimedia applications. *Comm. ACM 34*, 4 (April 1991).

[24] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995.

[25] GEMMELL, J., AND CHRISTODOULAKIS, S. Principles of delay-sensitive multimedia data storage and retrieval. *ACM TOIS 10*, 1 (January 1992), 51–90.

[26] GHANDEHARIZADEH, S., RAMOS, L., ASAD, Z., AND QURESHI, W. Object placement in parallel hypermedia systems. In *Proceedings of the 17th International Conference on Very Large Data Bases* (September 1991), pp. 243–253.

[27] GUTFREUND, Y.-S., DIAZ-GONZALEZ, J., SASNETT, R., AND PHUAH, V. Circustalk: An orchestration service for distributed multimedia. In *ACM Multimedia 93, proceedings* (August 1993), pp. 351–358.

[28] HERRTWICH, R. The role of performance, scheduling and resource reservation in multimedia systems. In *Operating Systems of the 90s and Beyond* (1991), vol. 563 of *Lecture Notes In Computer Science*, Springer Verlag, pp. 279–284.

[29] HERRTWICH, R. G. Summary of the Second International Workshop on Network and Operating System Support for Digital Audio and Video. *Operating Systems Review 26*, 2 (April 1992), 32–59.

[30] HEWLETT-PACKARD CO. *Using the Audio Application Program Interface.* Palo Alto, CA, 1992.

[31] HIGGINS, G. Image quality criteria. *Journal of Applied Photographic Engineering 3*, 2 (1977), 53–60.

[32] HODGES, M. E., SASNETT, R. M., AND ACKERMAN, M. S. A construction set for multimedia applications. *IEEE Software* (January 1989), 37–43.

[33] HUTCHISON, D., COULSON, G., CAMPBELL, A., AND BLAIR, G. S. Quality of service management in distributed systems. *Tech. Rep.* MPG-94-02, Lancaster University, 1994.

[34] JEFFAY, K., STONE, D., TALLEY, T., AND SMITH, F. Adaptive, best-effort delivery of digital audio and video across packet-switched networks. In *Third International Workshop on Network and Operating System Support for Digital Audio and Video* (San Diego, California, November 1992), IEEE Computer Society, pp. 1–12.

[35] KICZALES, G., AND LAMPING, J. Operating systems: Why object-oriented? In *Proceedings Third International Workshop on Object Orientation in Operating Systems* (December 1993), pp. 25–30.

[36] KOEGEL-BUFORD, J. F. Middleware system services architecture. In *Multimedia Systems*. Addison-Wesley, New York, 1994, pp. 221–244.

[37] LEHOCZKY, J., SHA, L., AND DING, Y. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proc. IEEE 10th Real-Time Systems Symp.* (December 1989), pp. 166–171.

[38] LENSTRA, J., RINNOOY, A., AND BRUCKER, P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics 1* (1977).

[39] LEVERGOOD, T. M., PAYNE, A. C., GETTYS, J., TREESE, G. W., AND STEWART, L. C. Audiofile: A network-transparent system for distributed audio applications. In *USENIX Summer Conference* (June 1993).

[40] LIMB, J. O. Distortion criteria of the human viewer. *IEEE Trans. on Systems, Man, and Cybernetics SMC-9* (December 1979), 778–793.

[41] LITTLE, T., GHAFOOR, A., CHEN, C., CHANG, C., AND BERRA, P. Multimedia synchronization. *Data Engineering 14*, 3 (September 1991), 26–35.

[42] LITTLE, T. D., AND GHAFOOR, A. Network considerations for distributed multimedia object composition and communication. *IEEE Network 4* (November 1990), 32–49.

[43] LITTLE, T. D., AND GHAFOOR, A. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communication 8*, 3 (April 1990), 413–27.

[44] LIU, C., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard real time environment. *JACM 20*, 1 (1973), 46–61.

[45] LOCANTHI, B. N. Fast bitblt() with asm() and cpp. In *Proceedings of EUUG '87 Conference* (Dublin, Ireland, August 1987).

[46] LOUGHER, P., AND SHEPHERD, D. The design and implementation of a continuous media storage server. In *Third International Workshop on Network and Operating System Support for Digital Audio and Video* (San Diego, California, November 1992), IEEE Computer Society, pp. 63–74.

[47] LOUGHER, P., AND SHEPHERD, D. The design of a storage server for continuous media. *The Computer Journal 36*, 1 (February 1993), 32–42.

[48] LUTHER, A. C. *Digital Video in the PC Environment.* McGraw-Hill, New York, 1989.

[49] LUTHER, A. C. Digital video and image compression. In *Multimedia Systems.* Addison-Wesley, New York, 1994, pp. 143–174.

[50] MAIER, D., WALPOLE, J., AND STAEHLI, R. Storage system architectures for continuous media data. In *Foundations of Data Organization and Algorithms, FODO '93* (1993), vol. 730 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1–18.

[51] MERCER, C., SAVAGE, S., AND TOKUDA, H. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the International Conference on Multimedia Computing and Systems, Boston, MA* (May 1994), IEEE Computer Society Press, Los Alamitos, pp. 90–99.

[52] MEYER-BOUDNIK, T., AND EFFELSBERG, W. MHEG explained. *IEEE Multimedia 2*, 1 (1995), 26–38.

[53] MICROSOFT CORPORATION. *Microsoft Windows Multimedia Programmer's Workbook.* Microsoft Press, 1991.

[54] MILLER, E. L., AND KATZ, R. H. Rama: Easy access to a high-bandwidth massively parallel file system. In *Proceedings of the 1995 Winter USENIX Conference* (New Orleans, LA, January 1995), USENIX Association, pp. 59–70.

[55] MILLS, D. L. Precision synchronization of computer network clocks. *Computer Communication Review 24*, 2 (April 1994), 28–43.

[56] NAHRSTEDT, K., AND SMITH, J. M. The QOS broker. *IEEE Multimedia 2*, 1 (1995), 53–67.

[57] PARCPLACE SYSTEMS, INC. *VisualWorks.* 999 E. Arques Ave., Sunnyvale, CA 94086, 1992.

[58] PEREZ-LUQUE, M. J., AND LITTLE, T. A temporal reference framework for multimedia synchronization. *To appear in IEEE Journal on Selected Areas in Communications* (1995).

[59] PHILIPS SEMICONDUCTORS. *Desktop Video Handbook.* Palo Alto, CA, 1992.

[60] PISCITELLO, D. M., AND CHAPIN, A. L. *Open Systems Networking.* Addison-Wesley, 1993.

[61] RAMAKRISHNAN, K., VAITZBLIT, L., GRAY, C., VAHALIA, U., TING, D., TZELNIC, P., GLASER, S., AND DUSO, W. Operating system support for a video-on-demand file service. In *NOSSDAV '93* (November 1993), Lancaster University, pp. 225–236.

[62] RAMAMRITHAM, K., AND PU, C. A formal characterization of epsilon serializability. *IEEE Transactions on Knowledge and Data Engineering* (June 1995).

[63] RANGAN, P. V., BURKHARD, W. A., BOWDIDGE, R. W., VIN, H. M., LINDWALL, J. W., CHAN, K., AABERG, I. A., AND YAMAMOTO, L. M. A testbed for managing digital video and audio storage. In *Proceedings of the Summer 1991 USENIX Conference* (1991), pp. 199–208.

[64] RANGAN, P. V., AND RAMANATHAN, S. Rate-based feedback techniques for continuity and synchronization in multimedia retrieval over high-speed networks. *Tech. Rep.* CS92-230, UCSD, March 1992.

[65] ROCHKIND, M. J. *Advanced UNIX Programming.* Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[66] ROWE, L. A., PATEL, K. D., SMITH, B. C., AND LIU, K. MPEG video in software: Representation, transmission, and playback. In *High Speed Networking and Multimedia Computing, IS&T/SPIE, Symposium on Elec. Imaging Sci. & Tech., San Jose, CA* (February 1994).

[67] SHA, L., AND SATHAYE, S. S. A systematic approach to designing distributed real-time systems. *Computer 26*, 9 (September 1993), 68–78.

[68] SHETLER, T. Birth of the BLOB. *BYTE (USA) 15*, 2 (Feb 1990), 221–2, 224, 226.

[69] SPIVEY, J. *The Z Notation: a Reference Manual*, second ed. Prentice-Hall International, New York, 1992.

[70] STAEHLI, R. A., AND WALPOLE, J. Constrained-latency storage access. *Computer 26*, 3 (March 1993), 44–53.

[71] STANKOVIC, J. A., AND RAMAMRITHAM, K. The Spring Kernel: A new paradigm for real-time operating systems. *Operating Systems Review 23*, 3 (July 1989), 54–71.

[72] STANKOVIC, J. A., AND RAMAMRITHAM, K. The Spring Kernel: A new paradigm for real-time systems. *IEEE Software* (May 1991), 62–72.

[73] STANKOVIC, J. A., AND ZHAO, W. On real-time transactions. *SIGMOD Record 17*, 1 (March 1988), 4–18.

[74] STEINMETZ, R. Data compression in multimedia computing - standards and systems. *Multimedia Systems 1*, 5 (March 1994), 187–204.

[75] STEINMETZ, R., AND ENGLER, C. Human perception of media synchronization. *Tech. Rep.* 43.9310, IBM European Networking Center, 1993.

[76] SUN MICROSYSTEMS. *AUDIO_HDR(3).* Sun Software Technical Bulletin, 1989.

[77] THE VOYAGER COMPANY. *A Hard Day's Night.* 1992. CD-ROM.

[78] THIMM, H., AND KLAS, W. Delta-sets for optimized reactive adaptive playout management in distributed multimedia database systems. To appear in *12th International Conference on Data Engineering*, New Orleans, LA (February 1996).

[79] TOKUDA, H., NAKAJIMA, T., AND RAO, P. Real-time Mach: Towards a predictable real-time system. In *Mach Workshop* (1990), USENIX Association.

[80] TOKUDA, H., TOBE, Y., CHOU, S. T.-C., AND MOURA, J. M. Continuous media communication with dynamic QOS control using ARTS with an FDDI network. In *SIGCOMM '92, Computer Communcation Review* (1992), vol. 22, ACM Press, New York, pp. 88–98.

[81] ULICHNEY, R. *Digital Halftoning.* The MIT Press, 1987.

[82] VENKATRAMANI, C., AND CKER CHIUEH, T. A survey of near-line storage technologies: Devices and systems. *Tech. Rep.* Experimental Computer Systems Lab, TR #2, Dept. of Computer Science, SUNY Stony Brook, October 1993.

[83] VIN, H. M., GOYAL, P., AND GOYAL, A. A statistical admission control algorithm for multimedia servers. In *ACM Multimedia 94* (October 1994), pp. 33–40.

[84] VIN, H. M., AND RANGAN, P. V. Designing a multi-user HDTV storage server. *IEEE Journal on Selected Areas in Communication 11*, 1 (January 1993).

[85] WEISS, R., DUDA, A., AND GIFFORD, D. K. Composition and search with a video algebra. *IEEE Multimedia 2*, 1 (1995), 12–25.

[86] WERNIK, M., ABOUL-MAGD, O., AND GILBERT, H. Traffic management for B-ISDN services. *IEEE Network 6* (September 1992), 10–19.

[87] WHITE PINE SOFTWARE. *CU-SeeMe*, 1995. software.

[88] YU, C., SUN, W., BITTON, D., YANG, Q., BRUNO, R., AND TULLIS, J. Efficient placement of audio data on optical disks for real-time applications. *Comm. ACM 32*, 7 (July 1989).

[89] ZHANG, H., AND FERRARI, D. Improving utilization for deterministic service in multimedia communication. In *Proceedings of the International Conference on Multimedia Computing and Systems* (May 1994), pp. 295–305.

[90] ZHANG, L., DEERING, S., ESTRIN, D., SHENKER, S., AND ZAPPALA, D. RSVP: A new resource reservation protocol. *IEEE Network 7* (September 1993), 8–18.

[91] ZHAO, W., AND RAMAMRITHAM, K. Simple and integrated heuristic algorithms for scheduling tasks with time and resource constraints. *Journal of Systems and Software 7* (August 1987).

Appendix A

Glossary

clip A finite time segment from a single media source.

ClipServer SQUINT class for delivering a segment of a presentation from a single source.

complete error model An error model in which arbitrary accuracy can be specified by constraints on the magnitude of error model components.

content Specification of logical output values over time.

continuous media Common term in literature for digital audio and video, which approximate continuous real-time signals.

DBMS Database Management System.

distributed An activity spanning several computer systems.

error interpretation A set of functions that map an actual presentation onto an ideal presentation.

error model A definition of error component functions that may be used in an error interpretation.

error The difference between an ideal value and an actual value.

fps Frames per second.

hypermedia A network of media elements and navigable links between elements.

jitter error The high-frequency component of an error signal.

Kbps,Mbps,Gbps Data throughput units for thousands, millions, and billions of bits per second respectively.

logical output An abstraction for a physical output device such as a video display or an audio channel.

mapping dilemma Object implementation must map high-level functionality onto low-level mechanisms, but performance of this mapping decision cannot be hidden from clients.

mapping problem Finding a low-level presentation plan that satisfies a QOS specification.

MPEG-1 Motion Picture Experts Group standard for encoding a real-time stream of moving pictures.

presentation descriptor A set of parameters that specify a presentation.

presentation Real-time delivery of a composition that may include multiple media tracks.

QOS Quality of Service. Fidelity measure of service performance as compared to some ideal.

quality Specification of the allowable error between an ideal presentation and the actual outputs.

rate error The rate of change of shift error.

reference architecture Chapter 3 describes the elements for QOS playback from storage.

resolution The smallest reproducible pulse width.

resolution error The interval width for computing *zError*.

SQUINT Smalltalk QOS User Interface, the prototype multimedia player described in Chapter 5.

sample Data representing an output value at a single instant of time.

shift error The low-frequency component of an error signal.

sound error model An error model for which every specification allows presentations that are sufficiently close to the ideal and disallows presentations with unbounded error.

synch error The difference in shift error between two outputs.

track A composition of clips all to be presented on a single output device.

TrackManager SQUINT main class for presentation execution. Translates QOS requests into `ClipServer` requests.

PresentationManager SQUINT main class for presentation planning. Translates QOS requirements into subordinate `TrackManager` requests.

view A mapping from logical content to physical device coordinates and real time.

z **value** Either audio signal level or video image intensity.

zError The average difference between ideal z value and actual z value.

Biographical Note

Richard Staehli was born in Portland, OR, on October 29, 1957. He attended Washington High School in Portland for four years, graduating in 1975. During his sophomore year he moved with his family to Rome, Italy, where he continued his course work by mail. Richard attended the Honors College at the University of Oregon for one year as a pre-Journalism major. After two years working in a custom photography lab, he returned to school at The Evergreen State College in Olympia, WA, to study physics. Richard completed his B.S. there in 1982. From 1982 to 1989 he worked as a software engineer at Electro Scientific Industries (ESI) in Portland. At ESI, he developed a strong appreciation for the principles of modular programming and object-oriented software development. Although working primarily with the Pascal programming language, he learned Smalltalk through a 1985 class at the Oregon Graduate Institute (then called the Oregon Graduate Center) and brought object-oriented programming techniques to new software development at ESI. Seeking a stronger background in computer science, he joined the doctoral program at the Oregon Graduate Institute in the fall of 1989. His research interests include multimedia information systems, distributed operating systems, and object-oriented programming.

Selected publications

Richard Staehli, Jonathan Walpole, and David Maier. Device and physical data independence for multimedia presentations, To appear in *ACM Computing Surveys*, Symposium on Multimedia Systems, 1996.

Richard Staehli, Jonathan Walpole, and David Maier. Quality of service specifications for multimedia presentations. *Multimedia Systems*, 3(5/6):251,263, November 1995.

Shanwei Cen, Calton Pu, Richard Staehli, and Jonathan Walpole. A distributed real-time

MPEG video audio player. In *Proceedings of the 5th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, volume 1018 of *Lecture Notes in Computer Science*, pages 151-162. Springer-Verlag, 1995.

David Maier, Jonathan Walpole, and Richard Staehli. Storage system architectures for continuous media data. In *Foundations of Data Organization and Algorithms, FODO '93*, volume 730 of *Lecture Notes in Computer Science*, pages 1-18. Springer-Verlag, 1993.

Richard Staehli and Jonathan Walpole. Using script-based QOS specifications for resource scheduling. In *Proceedings of the 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 93-95, Lancaster, England, November 1993. Lancaster University.

Richard A. Staehli and Jonathan Walpole. Constrained-latency storage access. *Computer*, 26(3):44-53, March 1993.