Quality of Service Specification for Multimedia Presentations*

Richard Staehli, Jonathan Walpole and David Maier
{*staehli, walpole, maier*}*@cse.ogi.edu*

Department of Computer Science & Engineering
Oregon Graduate Institute of Science & Technology
20000 N.W. Walker Rd., PO Box 91000
Portland, OR 97291-1000

ABSTRACT

Multimedia presentations convey information not only by the output values and their sequence, but by the timing of those outputs. However, it is impossible to implement a presentation with perfect timing and it is often necessary to throw away information because of resource limitations. As with any reproduction of a signal, the utility of a time-based presentation depends on its fidelity to the ideal. This imprecise but intuitive definition of quality suggests that quality specification should be an important part of a multimedia system.

Recent operating systems and networking research has focussed on defining Quality of Service (QOS) parameters that define resource-level requirements for performance guarantees. Our work seeks to link user perceptions of quality with these resource-level QOS specifications. To make this link, we introduce a framework for the formal specification of a presentation in three orthogonal parts: content, view, and quality. The formal definition of presentation quality allows the widest possible latitude for optimizing system resources and may inspire new techniques for the storage and transport of presentation content. An architecture for translating user-level quality specifications into service guarantees with optimal use of resources is suggested.

Keywords: Quality of Service, Resource Reservations, Real-Time Specifications, Multimedia Authoring, Synchronization.

1 Introduction

Multimedia systems today support presentations with *continuous-media* [1, 17] such as video and audio, as well as synthetic compositions such as slide shows and computer-generated music. We call these *time-based* data types because they communicate part of their information content through presentation timing. While a query on a database of static data types results in a static view of (hopefully) correct data values, a query for playback of video data should result in a presentation with a dynamically changing view. The usefulness of such presentations depends in part on the accuracy of the timing. Because digital presentations can only approximate continuous values and timing, playback of continuous-media is a question of *quality* rather than correctness. For example, to reproduce NTSC video on a digital multimedia system a succession of frames from the video should be presented at approximately 30 frames per second and approximately synchronized with

*This research is supported by NSF Grants IRI-9223788 and IRI-9117008, and by funds from Tektronix, Inc. and the Oregon Advanced Computing Institute.

accompanying audio output. Frequently, the display device will not have the same resolution as the source data so that even the images will have to be approximations of the original content.

The previous example raises two questions: How accurate must a presentation be, and how can we ensure that a presentation achieves that accuracy? This paper attempts to answer the first question by giving a formal definition of presentation *quality* that measures both accuracy of timing and the accuracy of output values. This definition of presentation quality can then be used to specify user-level quality requirements. The question of how to ensure that quality requirements are met must be answered by a multimedia system. Section 2 suggests an architecture that derives guarantees for a QOS specification as part of an acceptance test.

QOS specifications for user requirements are still a novel concept. Network protocols have been proposed with transport-level QOS specifications that bound delay, minimum throughput and error rates for continuous media communications [11, 15, 17, 28, 29]. More recently, operating systems researchers have argued that *bandwidth reservations* are needed in a real-time operating system to support end-to-end QOS guarantees [3, 9, 13, 19, 21, 23, 30, 32]. Both the network and operating systems bandwidth reservations are typically derived from the type of the data being transmitted, with the assumption that multimedia presentations should deliver as much spatial and temporal resolution as possible. But with current capture, compression, and storage technology, multimedia data types can have resolution that exceeds both the output device capabilities and user requirements for playback quality. As the resolution of the data sources increases, users should be able to sacrifice quality in order to reduce the resource costs of playback.

Many existing multimedia systems make do without QOS-based resource reservations. For example, personal computer systems can successfully play compressed video and audio from CD-ROM, but are able to do so only because the application program has control of all system resources and because the data has been carefully crafted to suit the storage device's throughput [26]. Device independence is possible with adaptive algorithms that adjust the playback quality to the resources available [4, 14, 25, 31]. However, adaptive playback algorithms frequently degrade quality to an unacceptable level when resources overloads occur. A formal definition of quality is needed to specify which presentations are acceptable and what minimal reservations are required to avoid overloads.

This discussion leads to a number of goals for QOS specifications:

- **Model user perception of quality.** The value of a presentation depends on the user's perception of quality, while the cost of a presentation depends on resource usage. Just as modern compression algorithms are based on human perception [16, 33], a multimedia system can better optimize playback resources if it knows which optimizations have the least affect on quality.
- **Formal semantics.** Specifications should be unambiguous. A multimedia system should be able to prove that it can satisfy a given QOS specification through resource reservations.
- **Support for complex presentations.** Complex presentations can specify synchronization between media streams that originate at independent sources and at different times [18, 26].

This paper defines a framework and a language for specifcation of presentation QOS. The definitions are intended to be general enough to apply to any multimedia system. The framework considers user interactions for presentation control as interruptions that require re-computation of the presentation requirements. The next section defines our terminology in terms of an architectural model for multimedia presentations. Sections 3 and 4 elaborate on the specification of *content* and *view* respectively for a presentation. We then define *quality* in Section 5 as a function of a presentation's fidelity to the *content* and *view* specification. Section 6 suggests how a formal QOS specification can be used to optimize resource usage in a presentation. We close with a discussion of related work in Section 7 and our conclusions in Section 8.

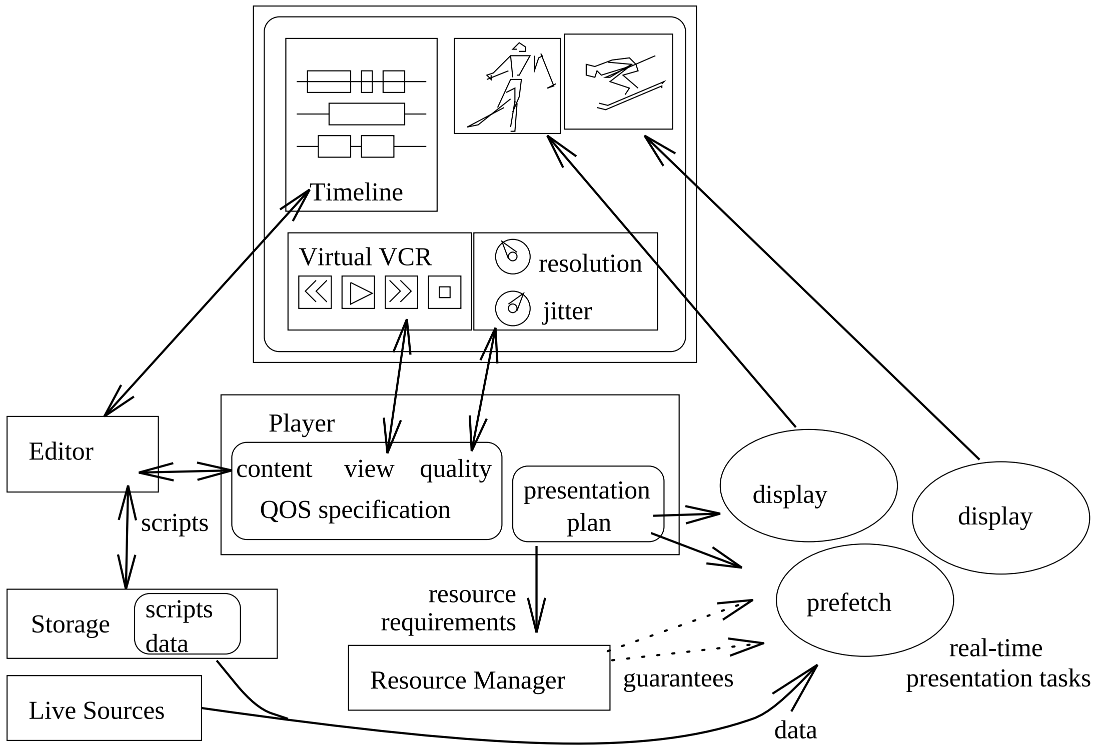

Figure 1: An architecture for editing and viewing multimedia presentations.

2 Architectural Model

In our architectural model, shown in Figure 1, multimedia data comes from live sources or from storage. A time-based media *editor* may be used to create complex multimedia *scripts* that specify the logical *content* of a presentation. Video and audio data have default scripts associated with them to specify the sample size, rate, and compression information needed for normal playback. For simplicity, we assume that scripts are not interactive. A *player* is used to browse and play-back scripts created by the editor. A user may control a player's *view* parameters, such as window size and playback rate, as well as *quality* parameters such as spatial and temporal resolution. The combination of content, view, and quality specifications constitute a QOS specification. When a user chooses to play a script, the player needs to find a *presentation plan* consisting of real-time tasks that satisfy the *QOS specification.* A presentation plan is feasible if guarantees can be obtained from a *Resource Manager* for the real-time presentation tasks that transport and transform the multimedia data from storage or other data sources to the system outputs.

2.1 Content, View and Quality

This architecture is similar to other research systems that provide QOS guarantees based on an admission test [25]. However, our definition of QOS is novel in that we make strong distinctions between content, view, and quality specifications. A *content* specification defines a set of logical image and audio output values as a function of time. A *view* specification maps content onto a set of physical display regions and audio output devices over a real-time interval. *Quality* is a measure of how well a real-time presentation matches the ideal presentation of some content on a view and a *quality specification* defines a minimum acceptable quality measure. We will refer to quality when we mean the measurement, and QOS when we mean the combination of content, view, and quality specifications.

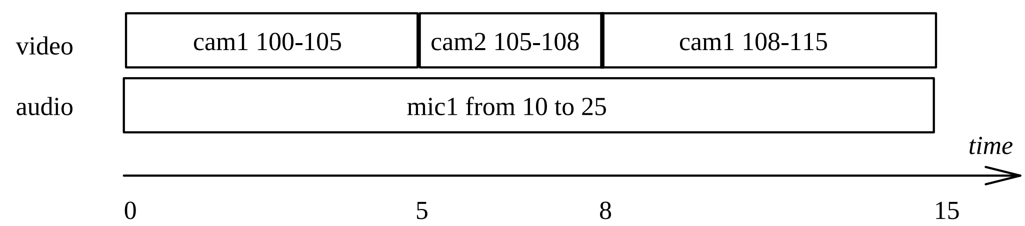

Figure 2: Timeline view of content specification for a presentation of video from a bicycle race.

By allowing independent control of content, view and quality, a multimedia system can offer a wider range of services that take advantage of the flexibility of computer platforms. To illustrate these services, consider the presentation of video from a bicycling race as described in Figure 2. The first video clip refers to 5 seconds of a digital video file. The video file is named *cam1* because it was captured with the first of two cameras recording the same bicycle race. The digital video for cam1 has a resolution of 320x240 pixels. A second video file named *cam2* shows another view of the cycling and has a higher resolution of 640x480 pixels. The video presentation cuts from *cam1* to *cam2* for 3 seconds, and then back to *cam1* for the last 7 seconds. The audio clip file *mic1* contains a digital audio soundtrack corresponding to the video clips. After selecting this content for presentation, a user should be able to choose view parameters and quality levels independently. For example, if the user chooses a view with a 640x480 pixel display window, but a quality specification that requires only 320x240 pixels of resolution, then the player may be able to avoid generating the full resolution images from cam2. The quality specification allows the user to indirectly control resource usage independent of the content and view selections. The player can optimize resource usage so long as the presentation exceeds the minimum quality specification. Users might also like to specify an upper bound on cost for resource usage, but measuring costs is beyond the scope of this paper.

3 Content Specification

To make the definitions of content, view, and quality as clear as possible, this paper describes a simple scripting language with minimal functionality. The *Timesynch* language defines data structures for *scripts* that specify non-interactive, time-based multimedia content. This section first defines the fundamental elements of a script, and then describes composition operators for constructing scripts of arbitrary complexity.

The content for a time-based multimedia presentation comprises a collection of logical displays or other output types whose values are defined over a period of time. For simplicity, we discuss only two output types: images and audio. Most state-of-the-art multimedia computing uses only combinations of these two output types to reproduce voice, stereo sound, text, graphics, still images, and video. Real numbers are used for the specification of logical coordinates and values to avoid placing an artificial limit on the resolution at which content can be reproduced in a presentation. In fact, many presentations are visualizations of continuous functions, in which case we believe it is inappropriate for the content specification to limit resolution. The resolution of a presentation is limited only by an actual implementation on digital outputs.

Figure 3 illustrates a recursive composition of script data structures to specify the same example presentation from Figure 2. We briefly describe this example before explaining the data structures in the remainder of this section. The root of the tree is a script that synchronizes the audio and video. Both children of the root are time-shift scripts, used to make both the audio and video scripts begin at logical time zero. The video script is a concatenation of the three video clips. Each clip references a sub-interval of a longer video script. The leaf-nodes in this figure are scripts that specify periodic updates to a logical device. The periodic script for audio is of type `Audio` with a logical output range of $[0, 1)$. Values are 8-bit samples read from a file named *mic1*, where each sample represents a real number in the range $[0, 256)$. The duration is 30 seconds and 8000 samples per second are to

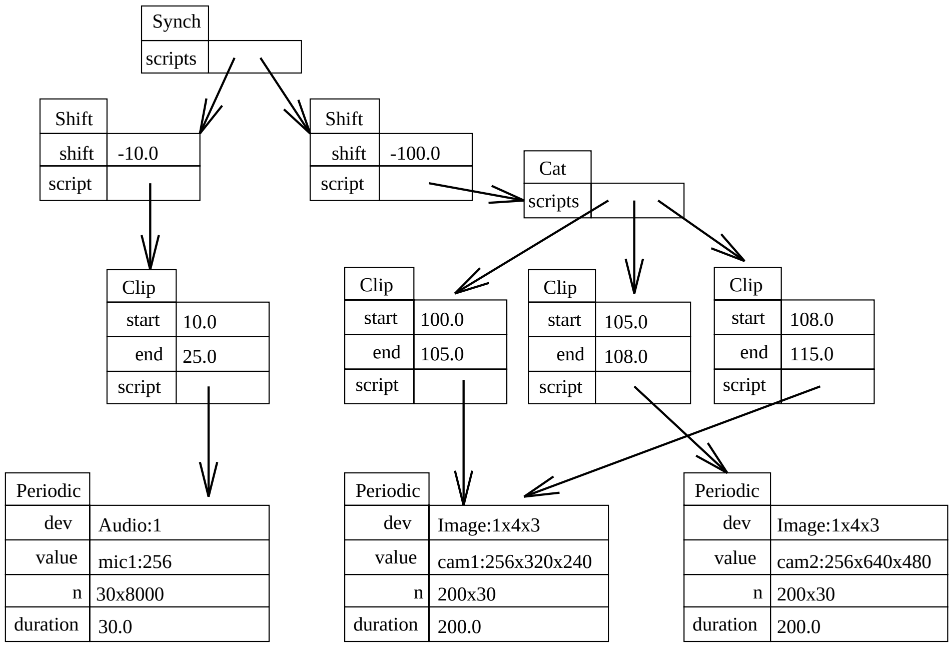

Figure 3: Script for example presentation. Values for the fields *dev* and *value* are suggested with a shorthand notation, where *Image:1x4x3* means that $dev.type = Image$, $dev.range.v = Range(0.0, 1.0)$, $dev.range.x = Range(0.0, 4.0)$, etcetera. The field n shows the number of samples as a product of the duration times the sample rate.

be output. The periodic video scripts are similar except that they specify a logical aspect ratio of 4x3 and the source files *cam1* and *cam2* have respectively 320x240 and 640x480 samples per frame.

A Timesynch script defines values for a set of logical output over time. All scripts share the ability to report their start time, duration, and the value specified for a logical output at any given time. Figure 4 shows the data types used to represent a script. These types have been implemented in the Smalltalk programming language and should be easily implemented in any other object-oriented language.

Before describing the representation for scripts, we need to explain the notation used in Figure 4. Named fields for each data type are shown within curly braces. The type of each field is indicated following a ":". We assume basic number types `Int` and `Real`, and the parameterized collection types `Set of` α and `List of` α. The notation `(Int)->ValueSource` denotes a function that takes an `Int` argument and returns a `ValueSource`. We will use a "." to reference a field in a structure so that, if r is a structure of type `RangeSpecs`, then $r.x.start$ refers to the *start* field within the x field of r. We will also write $TypeName(f_1, f_2, \ldots, f_n)$ to represent a structure of type $TypeName$ whose fields, in the order declared in Figure 4, have the values $f_1, f_2, \ldots, f_n$.

A `Script` is an abstract polymorphic type, with the subtypes shown in Figure 4 that each define a concrete representation. A `Basic` script specifies discrete media presentations with two fields: *dev* is a `LogicalOutput` for the presentation and *assignments* is a set of `Assignment` structures that define discrete presentation events, as illustrated in Figure 5. To make content specification independent of view specification, the `LogicalOutput` structure has only an abstract device type indicated by the field *type* and a logical range of values specified by the field *range*. The *range* field is a set of real numbers that specify intervals for x, y, and *value* coordinates. For example, a value v is in a range specified by a `Range` structure r if $r.start \le v < r.start + r.size$. A `LogicalOutput` *dev* with $dev.type$ = `Image` is for signals that vary in x and y, where x is in $dev.range.x$ and

```
Script = Basic | Periodic | Continuous
         | Shift | Scale | Synch | Clip | Cat

Basic = { dev:LogicalOutput assignments:Set of Assignment }
Periodic = { dev:LogicalOutput value:(Int)->ValueSource n:Int duration:Real }
Continuous = { dev:LogicalOutput value:(Real)->ValueSource duration:Real }
Shift = { shift:Real script:Script }
Scale = { scale:Real script:Script }
Synch = { scripts:List of Script }
Clip = { start:Real end:Real script:Script }
Cat = { scripts:List of Script }

LogicalOutput = { type:DevType range:RangeSpecs }
DevType = Image | Audio
RangeSpecs = { v:Range x:Range y:Range }
Range = { start:Real size:Real }
Assignment = { value:ValueSource time:Real }
ValueSource = { f:SourceFunction range:RangeSpecs }
SourceFunction = Real | (Real,Real)->Real
```

Figure 4: Data types for Timesynch scripts.

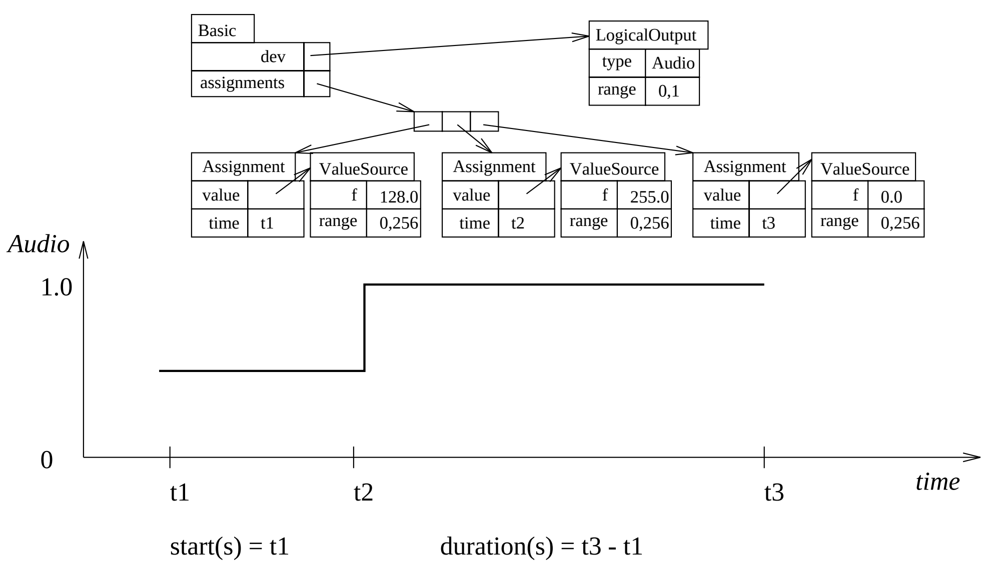

Figure 5: Semantic interpretation of a Basic script.

y is in $dev.range.y$. The values for a logical image output dev fall in the range $dev.range.v$. A `LogicalOutput` dev with $dev.type$ = `Audio` is for signals that have a single value at any given point in time and values that fall in the range $dev.range.v$. The fields $dev.range.x$ and $dev.range.y$ are ignored for a logical audio output.

Each `Assignment` structure has a *value* field that specifies a new value for the logical output and a *time* field that specifies a time for the assignment. The `ValueSource` structure has a field f that is a function that returns a real number for the assignment, and a *range* field that specifies the range for x, y, and value coordinates just as described for the `LogicalOutput` structure. For an assignment a, the `SourceFunction` $a.value.f$ must return a real number in the range $a.value.range.v$. For an assignment a to a logical image output, the `SourceFunction` $a.value.f(x, y)$ is defined for any x in $a.value.range.x$ and y in $a.value.range.y$.

A `Basic` script s specifies that the logical output $s.dev$ at time t is defined by an assignment a in $s.assignments$ if $a.time$ is the greatest assignment time in s that is less than or equal to t. Since the range of the value may differ from the range of the logical output, we define assignments using a scalar transformation function *trans* with type `(Real,Range,Range)->Real`. The *trans* function maps a value v in range r_1 to a new range r_2:

$$trans(v, r_1, r_2) = r_2.start + (v - r_1.start)r_2.size/r_1.size$$

If dev is a `LogicalOutput` and $dev.type$ = `Audio` then the value of dev specified by an `Assignment` a is:

$$trans(a.value.f, a.value.range.v, dev.range.v)$$

If dev is a `LogicalOutput` and $dev.type$ = `Video` then for all points (x_v, y_v) in the range specified in $a.value.range$, the value of dev at the corresponding point (x_{dev}, y_{dev}) is:

$$trans(a.value.f(x_v, y_v), a.value.range.v, dev.range.v)$$

where x_{dev} and y_{dev} are defined by:

$$x_{dev} = trans(x_v, a.value.range.x, dev.range.x)$$
$$y_{dev} = trans(y_v, a.value.range.y, dev.range.y)$$

Figure 6 shows an example of the transformation from the coordinate space of the source function, to the space of a logical image output. For simplicity, this definition for assignments to a logical image output supports only monochrome images, although the same approach can be generalized to specify multiple values at every point for color images.

Let the functions $start(s)$ and $duration(s)$ represent the start time and duration respectively for a script s. The start time for a `Basic` script is the minimum of all of its assignment times. Its duration is the difference between the greatest of all its assignment times and the start time. The value of a logical output is undefined before it has been assigned by the script and after the script's end. Note that the last assignment in a script serves only as an end marker and its value is always ignored. If a `Basic` script assigns multiple values to a logical output at exactly the same time, the specification is interpreted as a non-deterministic choice between them. This interpretation is just the limiting case of multiple assignments that are very close together, where only the value of the last assignment persists for any duration. Although non-deterministic choice in multimedia presentations is unusual, it does not present a problem for our definition of quality in Section 5.

Digital audio and video can be specified as a `Basic` script with periodic assignments to a logical output, but the use of a separate `Assignment` structure for every media sample is unnecessary. Instead, Timesynch provides a `Periodic` script structure that specifies four fields: *dev* is a `LogicalOutput`, *value* is a function that maps from a sample number to a `ValueSource` structure, n is the number of samples, and *duration* is the logical duration of the script. A `Periodic` script s has the same semantics as a `Basic` script with the following set of `Assignments`:

$$\bigcup_{i=0}^{s.n} \{\texttt{Assignment}(s.value(i), (i \cdot s.duration/s.n))\}$$

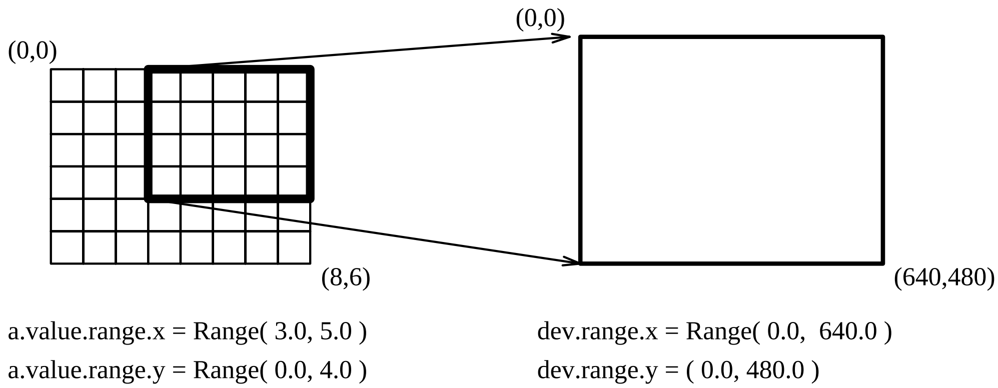

Figure 6: An example of an Assignment of values from a portion of a source matrix onto a logical image output.

This formula constructs a set with $s.n + 1$ `Assignments`, each specifiying the ith `ValueSource` and time $(i \cdot s.duration/s.n)$, for $i = 0$ to $s.n$.

Although both `Basic` and `Periodic` scripts assign new values to outputs at discrete points in time, there is no reason why we can't specify outputs that vary continuously with time. A `Continuous` script s is like a `Periodic` script except that it does not specify the number of samples and the function $s.value$ is indexed with a time value instead of a sample number. A `Continuous` script s may specify a different value for the logical output at every instant of the script's duration by using a continuous function for $s.value$ as in the following example:

$$s.value(t) = \texttt{ValueSource}(sin(t), RangeSpecs(Range(-1, 2), -, -))$$

This equation says that at any time t, the value of the logical output defined in s is $sin(t)$.

3.1 Complex Scripts

Given some set of `Basic`, `Periodic`, and `Continuous` scripts that each define a single logical output, we would like to edit these scripts to create arbitrarily complex compositions. A minimal set of script structures is described below that support temporal cut, paste, stretching and shrinking of content, and synchronization between logical devices. Although other features are desirable, such as the ability to mix several logical outputs together, the complex script structures described are sufficient for editing useful time-based multimedia presentations.

It is natural to view scripts as abstract objects that may themselves be synchronized in time. We can express arbitrary scalar transformations of time values with scripts that represent addition and multiplication operations. `Shift` and `Scale` scripts specify the same content as the scripts that they reference, but over shifted and scaled logical time intervals respectively. Let $value(dev, x, y, t, s)$ be a function that returns the value of a logical output dev at (x, y) and logical time t as defined by a script s. For audio logical devices, we can ignore the values of x and y. A `Shift` script s specifies that for all time t and logical outputs dev defined in s, $value(dev, x, y, t + s.shift, s) = value(dev, x, y, t, s.script)$. A `Scale` script s specifies that for all times t and logical outputs dev defined in s, $value(dev, x, y, t \cdot s.scale, s) = value(dev, x, y, t, s.script)$. Let a script s have start time zero and duration d. Then $Shift(s, t)$ has start time t and the same duration, while $Scale(s, f)$ has start time zero and duration $d \cdot f$.

Since we define synchronization through the time values in a script, synchronizing multiple logical outputs amounts to a specification that their scripts refer to the same time scale. The `Synch` script s has just that meaning for its children in the list $s.scripts$. Each child of a `Synch` script specifies a disjoint set of logical outputs. We refer to a script's logical outputs by number, according

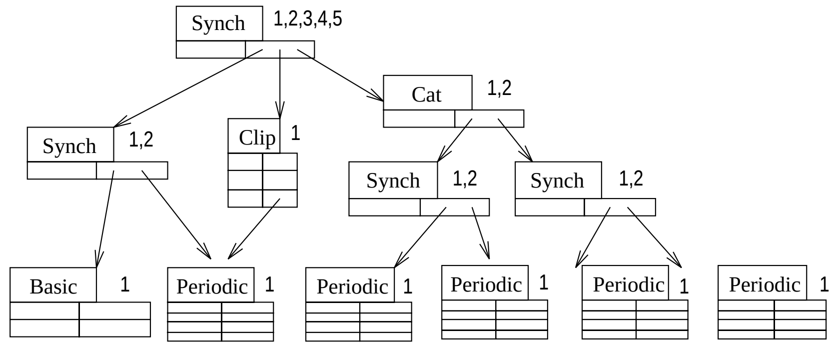

Figure 7: Every script defines a list of logical outputs. Logical outputs 1-5 in this complex script are defined by the children of the root node. Outputs 1 and 2 are defined respectively by outputs 1 and 2 of the first child, where they are in turn defined by the single output of each of its children. Output 3 is defined by output 1 of the clip script which is derived from the same Periodic script that defines Output 2. Outputs 4 and 5 are defined respectively by outputs 1 and 2 of the Cat script. Outputs 1 and 2 of the Cat script are defined in turn by each of the Cat script's children over disjoint time intervals.

to an in-order traversal as illustrated in Figure 7.

A common form of synchronization is concatenation in which one script immediately follows another in time. A `Cat` script is semantically equivalent to a `Synch` script whose elements are appropriately shifted in time:

$$Cat([s_1, \ldots, s_n]) \equiv Synch([s_1, \ldots, Shift(s_n, \sum_{i=1}^{n-1} duration(s_i))])$$

except that the nth logical output of each child is unified to specify the nth logical output of the parent. For example, Figure 7 shows a `Cat` script with two logical outputs that are defined by each of its children over separate intervals. Since the duration of each child of a `Cat` script cannot overlap, there is no conflict between the specifications of output values.

Finally, a script $Clip(s, t_1, t_2)$ represents a new set of logical outputs with start time t_1, duration $t_2 - t_1$, and the same values as s over the interval $[t_1, t_2)$. Figure 7 contains a `Clip` script that creates logical output 3.

4 View Specification

The logical outputs of a content specification have both temporal and spatial proportions, like the aspect ratio of an image, but they have no physical size or real duration. 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. Figure 8 shows a view specification that allocates a 200 by 150 pixel window on a monochrome (black and white) display for presentation of the bicycling script. 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 that must choose how to resample the source and whether to use dithering to represent gray levels. The combination of content and view specifications serve as a device independent specification of a perfect quality presentation. In the next section, we define less-than-perfect quality based on the difference between a presentation and this ideal specification.

The data structure for a view shown in Figure 9 has a *map* field that assigns logical outputs in a script to distinct `PhysicalOutputs`. The mapping is represented by a list of `Allocation` structures,

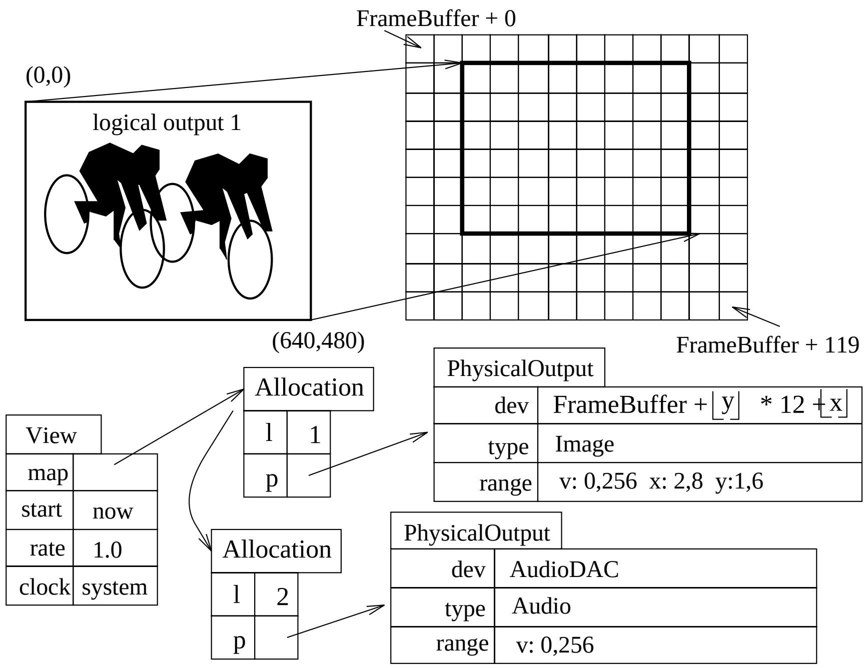

Figure 8: Example of a view that allocates an 8x6 pixel window on a display device for presentation of the bicycling video.

```
View = { map:List of Allocation start:Real rate:Real clock:Real }
Allocation = { l:Int p:PhysicalOutput }
PhysicalOutput = { dev:SinkLocation type:DevType range:RangeSpecs }
SinkLocation = Location | (Real,Real)->Location
```

Figure 9: Data types for a view.

each with an integer index l for a logical output and a field p for the allocated `PhysicalOutput`. A `PhysicalOutput` has a field *dev* that names the location of the physical device, a field *type* that indicates whether the device handles `Audio` or `Image` outputs, and a field range that gives the coordinate and value ranges just as described earlier for a `ValueSource`. We consider only the case where Audio and Image logical outputs are mapped respectively to Audio and Image `PhysicalOutputs`. The view structure also has fields for the *start* time, *rate*, and a real time *clock* for a presentation. These fields map the logical times in a script to the time scale of the real-time clock.

We refer to a pair (C, V) of a content specification script C and a view V as an *ideal specification* of presentation output values. Let $\mathcal{S}(p, x, y, t)$ represent the value of a `PhysicalOutput` p at a point (x, y) and time t according to an ideal specification (C, V). If i is the index of a logical output defined in C, l is the `LogicalOutput` structure describing that logical output, p is the `PhysicalOutput` given by $V.map(i)$, x and y are in the range specified in $p.range$, and t is in the interval $[V.start, V.start + duration(C)/V.rate)$, then:

$$\mathcal{S}(p, x, y, t) = trans(\ value(i, x_i, y_i, t_i, C),\ l.range.v,\ p.range.v\ )$$

where

$$\begin{aligned} x_i &= trans(x,\ p.range.x,\ l.range.x) \\ y_i &= trans(y,\ p.range.y,\ l.range.y) \\ t_i &= (t - V.start) * V.rate + start(s) \end{aligned}$$

This equation says that the ideal specification for a physical output p at x, y, and t is the linear transform from logical to physical ranges of the value of a content specification C (at the corresponding transformed coordinates). The value of $\mathcal{S}(p, x, y, t)$ is undefined otherwise. If $p.type$ = `Audio` then the x and y parameters can be ignored. We will adopt the convention that an `Audio` p has a constant value over all points (x, y) so that we can treat both output types uniformly in the remainder of the paper.

5 Quality Specification

We define the *quality* of a presentation to be the ratio of the worth[1] of an actual presentation to the worth of an ideal presentation. In this section, we provide a model for computing the worth of an actual presentation and a mechanism for specifying the worth of an ideal presentation. First, we derive an error model for measuring the difference between ideal and actual presentations. Then, we define mechanisms for specifying the worth of a presentation and the affect of errors. Finally, we propose a function that computes average quality over any portion of a presentation, and syntax for specifying constraints on that function.

5.1 Defining an error model

Quality is lowered by decreasing resolution, adding noise, or other actions that distort the output values away from the specification. The definition of an ideal specification $\mathcal{S}(p, x, y, t)$ in the last section provides an unambiguous definition of desired output values over a period of time. Since it is possible to measure and record the *actual* output values over time, we can directly compare the actual presentation with a specification. Let $\mathcal{P}(p, x, y, t)$ be a function that gives the actual value of an output p at a point (x, y) and time t. We can take the pointwise difference between a presentation and a specification, $\mathcal{E}(p, x, y, t) = \mathcal{P}(p, x, y, t) - \mathcal{S}(p, x, y, t)$, as an error measurement upon which to base our definition of quality. This simple approach is illustrated in Figure 10. During the presentation, $\mathcal{E}(p, t, x, y)$ computes the error for each output, at each point and time. Where $\mathcal{S}(p, x, y, t)$ is undefined, we take the error to be zero.

Unfortunately, this simple approach does not yield error values that correspond well to human perception. The second case in Figure 10 shows that a simple startup delay produces large error

[1] We use the term *worth* instead of *value* because we refer to output signal levels as *values*.

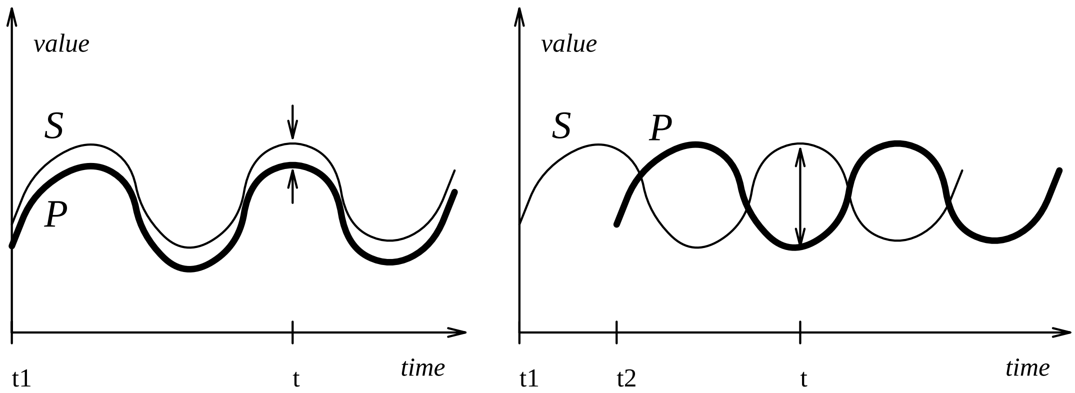

Figure 10: Measuring presentation error at time t when timing is perfect and when the presentation is delayed.

measurements. A person judging the quality of a presentation recognizes a delay in starting the presentation, but then sees a good match after compensating for the delay. In fact, a presentation may suffer from many errors in timing and spatial presentation, in addition to distortions in the output values. Let us refer to a tuple (v, p, x, y, t) as an *event* that means value v occurs on output p at point (x, y) and time t. We can capture all error in a presentation by defining a mapping from events in presentation to corresponding ideal events in a specification. Equation 1 in Table 1 formally defines such a mapping in terms of error functions $\mathcal{E}_x^p$, $\mathcal{E}_y^p$, $\mathcal{E}_t^p$, and $\mathcal{E}_v^p$. This equation says that if a point (x, y) at time t for an output p during a presentation $\mathcal{P}$ corresponds to a point $(x+\mathcal{E}_x^p, y+\mathcal{E}_y^p)$ and time $t+\mathcal{E}_t^p$ in a specification $\mathcal{S}$, then $\mathcal{E}_v^p$ is the difference in their values. If these error functions are zero for all outputs in a presentation, then the presentation is perfect, by definition.

It is important to note that for any presentation and its specification, there are an infinite number of error functions $\mathcal{E}_x^p$, $\mathcal{E}_y^p$, $\mathcal{E}_t^p$, and $\mathcal{E}_v^p$ that satisfy Equation 1. Equation 1 does not uniquely define these error functions, but only requires that theycompletely account for differences between presentation and specification.

Let an *error model* be a set of function definitions that completely describe all possible error between an presentation and a specification. Equation 1 is the simplest error model since each of the error functions in it are fully orthogonal, but this error model fails to quantify the errors that a user perceives. For example, users are sensitive to errors in audio-video synchronization. Consider the content from Figure 3 and the view specification from Figure 8. If the video is presented 5 seconds late and the audio only 3 seconds late, then the 2 second error in synchronization between the audio and the video is even more annoying than the start-up delays.

Table 1 shows an error model that formally defines error measures for user-perceived presentation artifacts. This set of error measures includes well understood artifacts such as temporal jitter and spatial blurring and generalizes these concepts in all dimensions. These error measures are briefly described below.

Table 1 defines *shift*, *rate*, and *jitter* errors to model user perceived temporal and spatial errors. $\mathcal{E}_{shift_t}$ is the amount by which a presentation is seen to be behind schedule, $\mathcal{E}_{rate_t}$ is the rate of change of $\mathcal{E}_{shift_t}$, and $\mathcal{E}_{jitter_t}$ measures small timing errors not already accounted for by $\mathcal{E}_{shift_t}$. The same error measures are defined for x and y dimensions since `Image` presentations can suffer from displacement, scaling and small distortions that are analogous to shift, rate and jitter.

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 meaningful. The problem is that humans don't perceive independent values at infinitesimal points, but instead in-

For each output p:

$$\forall x, y, t : \mathcal{E}_v^p = \mathcal{P}(p, x, y, t) - \mathcal{S}(p, x + \mathcal{E}_x^p, y + \mathcal{E}_y^p, t + \mathcal{E}_t^p) \tag{1}$$

$$\begin{aligned}
\mathcal{E}_x^p &= \mathcal{E}_{shift_x}^p + \mathcal{E}_{jitter_x}^p \\
\mathcal{E}_y^p &= \mathcal{E}_{shift_y}^p + \mathcal{E}_{jitter_y}^p \\
\mathcal{E}_t^p &= \mathcal{E}_{shift_t}^p + \mathcal{E}_{jitter_t}^p \\
\mathcal{E}_{rate_x}^p &= \frac{\partial \mathcal{E}_{shift_x}^p}{\partial x} \\
\mathcal{E}_{rate_y}^p &= \frac{\partial \mathcal{E}_{shift_y}^p}{\partial y} \\
\mathcal{E}_{rate_t}^p &= \frac{\partial \mathcal{E}_{shift_t}^p}{\partial t} \\
\int\int\int_{\mathcal{N}(x,y,t)} \mathcal{E}_v^p \, dx \, dy \, dt &= \int\int\int_{\mathcal{N}(x,y,t)} \mathcal{E}_{offset}^p + \mathcal{E}_{scale}^p \cdot v_{ideal} + \mathcal{E}_{noise}^p \, dx \, dy \, dt
\end{aligned}$$

where $v_{ideal} = \mathcal{S}(p, x + \mathcal{E}_x^p, y + \mathcal{E}_y^p, t + \mathcal{E}_t^p)$ and $\mathcal{N}(x, y, t)$ is the neighborhood around a point (x, y, t) defined by:

$$\mathcal{N}(x, y, t) = \{(x', y', t') | (|x - x'| < \mathcal{E}_{blur_x}^p) \wedge (|y - y'| < \mathcal{E}_{blur_y}^p) \wedge (|t - t'| < \mathcal{E}_{blur_t}^p)\}$$

For each pair of outputs p and q:

$$\mathcal{E}_{synch}^{p,q} = \mathcal{E}_{shift_t}^p - \mathcal{E}_{shift_t}^q$$

Table 1: Example error model. All error measures are functions of x, y, and t, but we write $\mathcal{E}_v^p$ instead of $\mathcal{E}_v^p(x, y, t)$ for easier reading.

tegrate 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 average values. Let $\mathcal{E}_{blur_x}$ be the width of the smallest resolvable vertical stripe in a presentation. We define $\mathcal{E}_{blur_y}$ and $\mathcal{E}_{blur_t}$ similarly. Then the interesting measure of value error is the difference in average value over a region with dimensions $\mathcal{E}_{blur_x} \cdot \mathcal{E}_{blur_y} \cdot \mathcal{E}_{blur_t}$. This separates value errors into what humans perceive as resolution loss and actual "wrong" values.

It is also useful to distinguish a picture that is too bright or an audio signal that is too loud from random noise. Table 1 defines $\mathcal{E}_{offset_v}$, $\mathcal{E}_{scale_v}$, and $\mathcal{E}_{noise_v}$ for value errors as components of $\mathcal{E}_v^p$ when averaged over the blurring intervals in each dimension.

In addition to measuring the error in reproducing a specified signal on an output, the relationships between outputs carry information and should be considered an independent source of error. For example, lip synch between the audio and video tracks of a speaker is important. Both tracks may be reproduced perfectly except for a 1/3 second difference in start times, yet the persistent error in lip-synch is annoying [27]. We define $\mathcal{E}_{synch}^{p,q}$ to measure the synchronization error between two outputs p, and q at every point in time.

5.2 Choosing error measures

The definitions of error measures in our model are intentionally circular. The determination of error functions is inherently ambiguous because there is no information in an output signal about the

$$w^p(t) = \frac{1}{duration(C)} \qquad p.type = \texttt{Audio}$$

$$w^p(x,y,t) = \frac{1}{area(p) \cdot duration(C)} \qquad p.type = \texttt{Image}$$

$$w^{p,q}(x,y,t) = \frac{w^p(x,y,t) \cdot w^q(x,y,t)1}{w^p(x,y,t) + w^q(x,y,t)} \qquad p.type = \texttt{Audio}, q.type = \texttt{Image}$$

Figure 11: Timesynch worth functions for all p and q in a view V with content specification script C where $area(p) = p.range.x.size \cdot p.range.y.size$. No worth is assigned to outputs that do not match one of these functions.

intended correspondence with a specification. Each user perceives error in a presentation subjectively, and may assess the error differently than another user. Let an *interpretation* I be a choice of continuous functions that satisfy an error model. There are an infinite number of interpretations for a presentation, each with a different affect on presentation quality. What matters is that a presentation allows an interpretation with acceptable errors. We assume that user's are good at recognizing the intended presentation content and that they therefore will perceive the interpretation with the most acceptable errors. To complete a definition of quality specification then, we need to be able to compute the affect of errors on a presentation.

5.3 Modeling the worth of a presentation

Timesynch assumes that the worth of a presentation is the sum of its parts. That is, if a presentation is composed of parts $(p_1, \ldots, p_n)$, each with worth $w(p_i)$ in an ideal presentation, and each is diminished in worth in an actual presentation by a factor q_i, then the worth of the whole is the sum:

$$\sum_{i=1}^{n} w(p_i) \cdot q_i$$

A *worth model* defines a *worth function* $w^p(x,y,t)$ for each output p that gives the relative worth of that output per unit area and unit time at a point (x,y) and time t. If two outputs function synergistically, as in the audio and video streams of a person talking, we include a worth function $w^{p,q}(x,y,t)$ that gives the added worth of both streams playing together. Note that this model allows us to specify that a given output may have worth *only* in combination with a second output. Figure 11 gives an example of worth functions that assign equal worth to all outputs. This definition of worth functions is implicit in Timesynch specifications.

5.4 Computing quality

We define *quality* to be the ratio of the worth of an actual presentation to the worth of an ideal presentation. A *quality function* computes this ratio from the error measures of an interpretation and has the following properties:

- Quality is one when all errors in interpretation are zero.
- Quality is monotonically decreasing with any increase in error.
- Quality is zero when all errors are maximal or infinite.

A *partial quality function*, $q^{ps}_m(x,y,t)$, gives the instantaneous ratio of actual to ideal worth for an output or pair of outputs ps, considering only error measure m. A QOS specification must define a partial quality function for every output or pair of outputs ps in a view and every error measure m in the error model. For example, the following equation defines a partial quality function for every error measure $\mathcal{E}^{ps}_m(x,y,t)$:

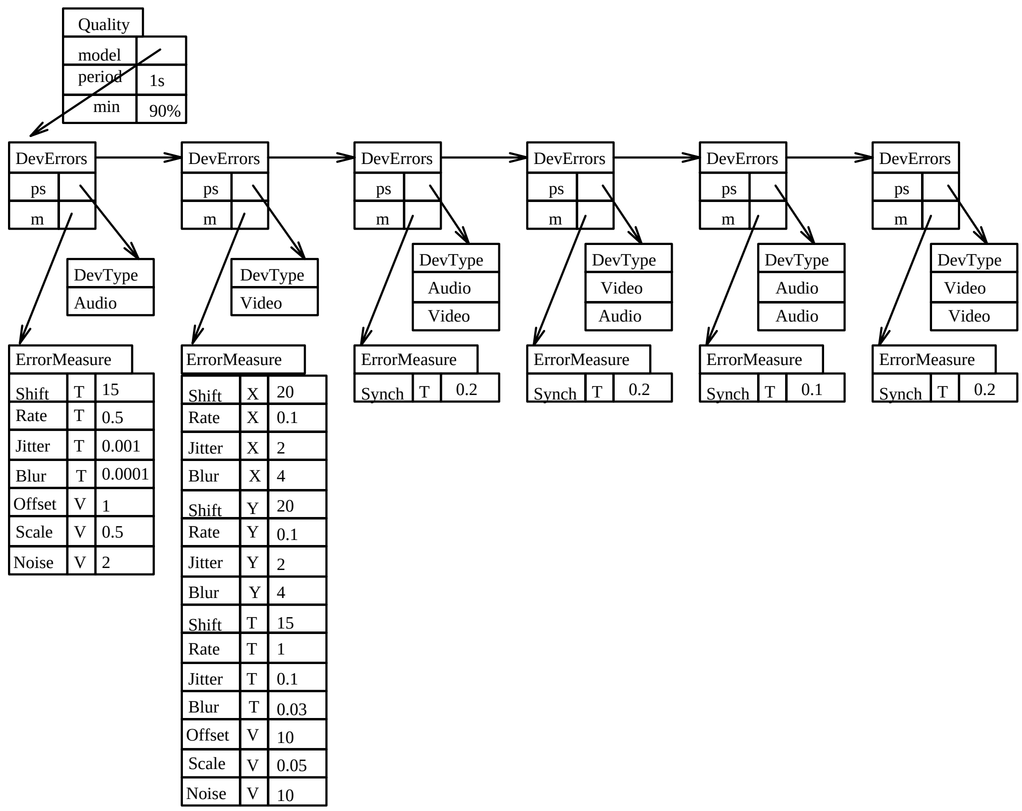

Figure 12: Example quality specification. Critical error values for temporal error measures are given in seconds. Values for spatial error measures are given in pixels, and critical error values for value-error measures are given in value quantization levels.

$$q_m^{ps}(x,y,t) = e^{-\frac{|\mathcal{E}_m^{ps}(x,y,t)|}{C_m^{ps}}} \tag{2}$$

If the error measure $\mathcal{E}_m^{ps}$ is zero in an interpretation, then the partial quality function is one. As the error increases, the partial quality decays exponentially. We call the constants C_m^{ps} *critical error values*. When $\mathcal{E}_m^{ps}(x,y,t) = C_m^{ps}$, the partial quality is approximately 0.37 so we choose these critical error values to correspond to decidedly poor quality. Figure 12 shows an example of critical error values for all the error measures defined in Table 1.

Given an error model E, worth functions w^{ps} for each output or pair of outputs ps, and partial quality functions q_m^{ps} for the same output(s) ps and each error measure $m \in E$, we propose a formal definition of presentation quality as follows. The average presentation quality for a set of worth functions in W, over x, y, and t in $\mathcal{N}$, according to an interpretation I is:

$$Q_{avg}(W,\mathcal{N},I) = \frac{\sum_{w^{ps}\in W} \int\int\int_{\mathcal{N}} w^{ps}(x',y',t') \prod_{m\in E} q_m^{ps}(x,y,t)\,dx\,dy\,dt}{\sum_{w^{ps}\in W} \int\int\int_{\mathcal{N}} w^{ps}(x',y',t')\,dx\,dy\,dt} \tag{3}$$

where the worth functions are computed from the corresponding points in the specification: $x' = x + \mathcal{E}_x^p$, $y' = y + \mathcal{E}_y^p$, and $t' = t + \mathcal{E}_t^p$.

This formula computes the ratio of the actual worth for a portion of a presentation to its ideal worth. The acutal worth, for a portion of a presentation defined by W and $\mathcal{N}$, is the sum of the actual contributions for each worth function. The contribution of each worth function $w^{ps}(x,y,t)$

```
Quality = { model:List of DevErrors period:Real min:Real }
DevErrors = { ps:List of DevType m:List of ErrorMeasure }
ErrorMeasure = { name:ErrorName dimension:Dimension critical:Real }
ErrorName = Shift | Rate | Jitter | Blur | Offset | Scale | Noise | Synch
Dimension = X | Y | T | V
```

Figure 13: Declarations for quality specification structure.

is the integral over $\mathcal{N}$ of the product of $w^{ps}(x, y, t)$ with all partial quality functions that assess the impact of presentation errors on output(s) ps. The ideal worth is just the sum of the integral of each worth function over $\mathcal{N}$.

A *quality model* supplies the error model, worth functions, partial quality functions, and the average quality function that is used to compute presentation quality. We believe this definition is completely general in that a quality model exists for every mapping from presentation error to quality assessment. Conversely, a particular quality model determines a unique mapping that can only approximate user perception. The utility of a particular choice of a quality model for an application depends on how well it approximates user perception for the type of presentations that occur. This paper provides an example of a quality model through the error model in Table 1, the worth functions defined in Figure 11, the partial quality functions defined in Equation 2, and the average quality function of Equation 3. The error model gives formal definitions for shift, rate, jitter, blur and other error measures that are a superset of the QOS parameters proposed by other researchers [13, 19, 29]. Further work is needed to evaluate the utility of this quality model.

5.5 Specifying minimum quality

The framework outlined above for a quality model resulted in a definition of average quality for a portion of a presentation. The Timesynch language offers a `Quality` structure that specifies a quality model, an averaging interval and a minimum value for average quality. Figure 13 shows the declarations for the `Quality` structure. The quality model is mostly implicit, with worth functions as defined in Figure 11, partial quality function as defined in Equation 2 and average quality as defined in Equation 3. The `Quality` structure has a *model* field that represents the error model and critical error values for computing partial quality functions. The `DevErrors` structure associates a list of device types with a list of error measures. Figure 12 illustrates how the model can associate the singleton list of device type `Audio` with error measures for temporal `Shift`, `Rate`, `Jitter`, `Blur`, and value error types `Offset`, `Scale`, and `Noise`. Each `ErrorMeasure` is represented by its *name* field, a *dimension* field, and a *critical* value for computing partial quality as defined in Equation 2. The *dimension* field can be `T` for time or `V` for value, but can also be `X` or `Y` if the output is an `Image`.

The meaning of a Timesynch quality specification Q, for content and view specifications C and V, is stated as follows: there exists an interpretation I such that for all times t_0, $Q_{avg}(W_V, \mathcal{N}, I) \geq Q.min$, where W_V is the set of worth functions for the view V as defined in Figure 11 and $\mathcal{N}$ is the set of all points (x, y, t) with $t_0 \leq t < t_0 + Q.period$. It is important to note that we do not need to actually compute the best quality measure for all possible interpretations. We only need to reason about whether a particular presentation plan will achieve a certain quality.

6 Using Quality Specifications for Resource Reservation

A multimedia player can frequently meet a QOS specification with fewer resources than are needed for a maximal quality presentation. Consider the bicycle racing script of Figure 3, the quality specification in Figure 12, and a new view specification shown in Figure 14. The view represents

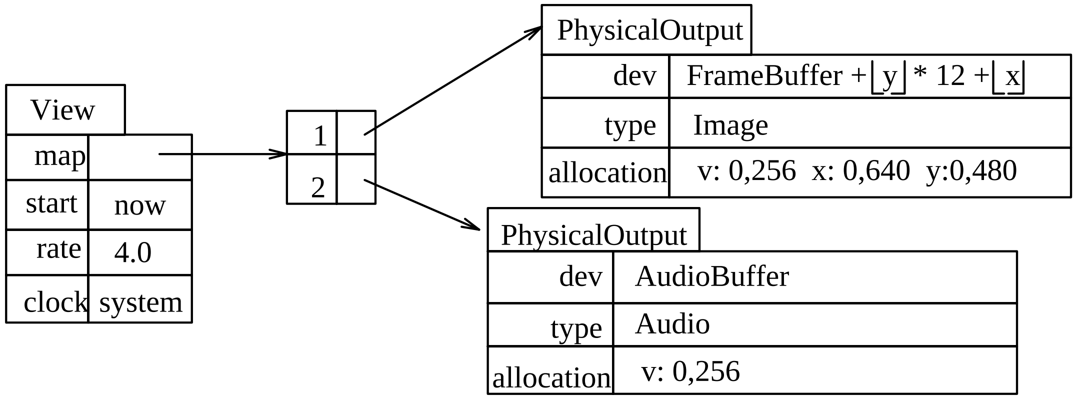

Figure 14: View specification for playback of bicycle racing video at four times normal rate.

a user request to play the script at 4 times the normal rate. The resulting ideal specification then calls for 120 fps of video. Fortunately, the quality specification only requires 90% average quality over any 1 second period of the presentation. If all aspects of the presentation were perfect except for video jitter, the quality specification would admit a presentation with average jitter less then or equal to 0.1 second, which allows the playback algorithm to drop frames. This result follows from Equation 3 by setting $Q_{avg}(W, \mathcal{N}, I)$ greater than or equal to 90% with the following definitions. Let W include worth functions from Figure 11 for audio output a and video output v. Let $\mathcal{N} = \{(x, y, t) | 0 \leq x < 640, 0 \leq y < 480, t_0 \leq t < t_0 + Q.period\}$ and let I be an interpretation that finds all error measures to be zero except for $\mathcal{E}^{video}_{jitter_t}$. Since the partial quality functions are equal to one when error is zero, we get:

$$0.9 \leq \frac{\int_t^{t+1} \int_0^{480} \int_0^{640} w^v(x, y, t) q^v_{jitter_t}(t)\, dx\, dy\, dt \;+\; \int_t^{t+1} w^a(t) dt \;+\; \int_t^{t+1} w^{a,v}(t) dt}{\int_t^{t+1} \int_0^{480} \int_0^{640} w^v(x, y, t)\, dx\, dy\, dt \;+\; \int_t^{t+1} w^a(t) dt \;+\; \int_t^{t+1} w^{a,v}(t) dt} \tag{4}$$

Figure 11 defines worth functions $w^v(x, y, t) = \frac{1}{640 \cdot 480 \cdot 15}$, $w^a(t) = \frac{1}{15}$, and $w^{a,v}(x, y, t) = \frac{1}{15}$ that each integrate to $\frac{1}{15}$ over $\mathcal{N}$:

$$0.9 \leq \frac{\int_t^{t+1} \frac{1}{15} q^{video}_{jitter_t}(t) dt \;+\; \frac{2}{15}}{\frac{3}{15}} \tag{5}$$

Simplifying and substituting the partial quality function from Equation 2, with the critical value for temporal video jitter from Figure 12, we get:

$$0.7 \leq \int_t^{t+1} e^{-|\frac{\mathcal{E}^{video}_{jitter_t}(t)}{0.1}|} dt \tag{6}$$

Let n be the number of frames that can be skipped in sequence from an otherwise perfect presentation without violating the above constraint. Then, as Figure 15 shows, $\mathcal{E}^{video}_{jitter_t}(t)$ is a periodic function with period $\frac{n+1}{120}$. For a frame that is presented at the specified time $t_1 - \frac{1}{120}$, interpretation I defines jitter to be zero for the 1/120 of a second duration specified for that frame. From t_1 to $t_1 + \frac{n}{120}$, the presentation falls behind as the next n frames are skipped. During this interval, $\mathcal{E}^{video}_{jitter_t}(t) = t_1 - t$. Since the integral of a periodic function is the same over each period, and we assume that the period $\frac{n+1}{120}$ is small relative to the one second interval for the integral, we can approximate the last equation with:

$$0.7 \leq \frac{120}{n+1} \left(\int_{t_1 - \frac{1}{120}}^{t_1} e^0 dt \;+\; \int_{t_1}^{t_1 + \frac{n}{120}} e^{-|\frac{t_1 - t}{0.1}|} dt \right) \tag{7}$$

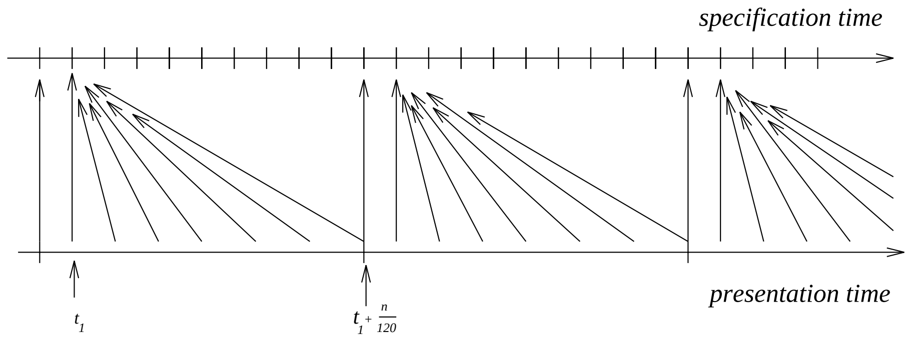

Figure 15: Jitter function maps presentation times onto ideal specification.

Taking the definite integrals gives:

$$0.7 \leq \frac{120}{n+1}(\frac{1}{120} - 0.1e^{-\frac{n}{12}} + 0.1e^{0}) \tag{8}$$

which yeilds:

$$n \leq \frac{1}{7}(123 + 120e^{-\frac{n}{12}})) \tag{9}$$

Values of n less than or equal to 10 satisfy this specification so that a presentation plan that displays only every tenth frame can satisfy the QOS specification.

Analysis of a QOS specification can identify a range of presentation plans that might satisfy the specification as illustrated above. To guarantee that a particular presentation plan will satisfy a QOS specification a player must reserve resources for storage access, decompression, mixing, and presenting processes. The attempt to reserve resources is called an acceptance test. The acceptance test may invoke resource reservation protocols for network and file system resources with resource-level QOS parameters derived from the process timing requirements. If the player can not find a presentation plan that both satisfies the QOS requirements and meets the acceptance test, then the QOS requirements must be renegotiated.

7 Related Work

It is now well understood that time-based multimedia systems require some form of resource guarantees for predictable performance. We consider related research in the categories of content specification, QOS specification, scheduling mechanisms and reservation protocols.

All authoring and playback tools that we are aware of produce informal specifications of multimedia presentations. The Muse system [12] was one of the earliest full-featured authoring tools that allows multi-track timeline synchronization of media objects. Objects may also be composed in spatial and other arbitrary dimensions. Muse provides extensive support for specifying interactive navigation, both through hypermedia links and graphical controls such as scroll bars. During non-interactive presentations, the accuracy of synchronization is determined by the playback mechanism and is not formally constrained. MAEstro [8] is another authoring tool that supports timeline synchronization of objects. The salient feature of MAEstro is that editing and playback functions are distributed among media-specific editors that may reside on remote machines. MAEstro's TimeLine editor is an X-windows-based program that supports both specification and playback of multimedia compositions by dispatching messages to the other media editors, such as the Digital Tape Recorder and the Image Editor. The TimeLine editor and the media editors rely on UNIX timer interrupts, Sun remote procedure calls, and the Unix scheduler to achieve coarse-grained synchronization. Xavier and Mbuild [10] are an experimental C++ class library and an editor, respectively, that support composition of multimedia objects with "glue" in a manner similar to

TEX. The CMIFed [24] editing and presentation environment provides some minimal support for specification of allowed deviations in timing. These and other similar authoring and playback tools implement best effort presentation plans and, in contrast to our approach, do not allow specification of QOS requirements independent of content and view.

Researchers have suggested a variety of parameters for multimedia QOS specifications. Continuous media stream access is generally described by throughput and delay or jitter bounds [2, 19, 23, 29]. Hutchinson, et al. [13], suggest a framework of categories for QOS specification including *reliability*, *timeliness*, *volume*, *criticality*, *quality of perception* and even *cost*. They provide only a partial list of QOS parameters to show that current QOS support in OSI and CCITT standards is severely limited. While these lists suggest many important ways to describe service categories, they go beyond user requirements and into specification of implementation. Our definition of QOS specification excludes volume, throughput and cost values because these values are secondary and can be derived from the combination of user requirements and system configuration. The Capacity-Based-Session-Reservation-Protocol (CBSRP) [29] supports reservation of processor bandwidth from the specification of a range of acceptable spatial and temporal resolutions for video playback requests. The resolution parameters are intended only for providing a few classes of service based on resource requirements and not for completely capturing user quality requirements. In particular, they do not adequately specify the accuracy of image values and ignore questions of clock drift and inter-stream synchronization.

Many researchers have demonstrated that quality can be traded for lower bandwidth requirements during a presentation. A variety of scaling methods may be applied to reduce the bandwidth requirements of video streams [6, 7]. Software feedback techniques have been used to dynamically adjust stream processing workloads to available system bandwidth [5, 22, 25, 29]. These techniques can be used agressively by a presentation planner to reserve minimal resources for a formal QOS specification.

Resource requirements may be derived from a presentation plan that satisfies a QOS specification. When the resource requirements are known, resource reservation protocols are needed to guarantee predictable access. Several groups have reported reservation protocols for network resources [1, 34, 35]. Processor capacity reservation has been implemented in the Real-Time Mach operating system [20] and file systems have been developed to support reservations for continuous media streams [3, 19, 23, 32]. These protocols can be used effectively within the architecture suggested at the end of Section 6.

8 Conclusions

This paper has described a new framework for QOS specification in multimedia systems. The primary contributions of this framework are the clear distinction between *content*, *view* and *quality* specifications, and the formal definition of presentation quality. The Timesynch language provides relatively simple constructs for the formal specification of complex multimedia content as well as constructs for view and quality specifications. Because every component of our QOS specifications have an unambiguous meaning it is possible to prove the correctness of a presentation plan as shown in Section 6. Furthermore, it is simple to specify quality constraints for complex compositions because the quality specification refers only to the outputs and not to the content specification structures.

Our formal definition of presentation quality is based on a mapping from presentation events and values to an ideal specification. This mapping provides a completeness criteria for error measurements in a QOS specification: that the error measurements completely define such a mapping. No other definitions of QOS parameters that we are aware of satisfy this completeness criteria. The error model of Table 1 formally defines a set of error measures that are a superset of the QOS parameters suggested by other researchers. Because this set of measures uniquely determines the mapping functions $\mathcal{E}_x^p$, $\mathcal{E}_y^p$, $\mathcal{E}_t^p$, and $\mathcal{E}_v^p$, we can be sure that they are complete.

The definition of quality given in Section 5 depends an *interpretation* that assigns a consistent set of functions for the error model. An important achievement of this definition is the recognition that presentation quality is not not uniquely determined by the presentation mechanism.

We plan to validate the utility of this work by implementing a playback system that uses these QOS specifications. We expect that it will be difficult to write tractable algorithms that find optimal presentation plans for a given QOS specification and system configuration. Initially, we will be content to make incremental improvements on the capabilities of existing systems. Further work also needs to be done with human perception to determine how to improve our user model.

References

[1] David P. Anderson, Ralf Guido Herrtwich and Carl Schaefer: SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet. Tech. Rep. 90-006, International Computer Science Institute, February, 1990.

[2] David P. Anderson, Yoshitomo Osawa and Ramesh Govindan: A File System for Continuous Media. ACM Transactions on Computer Systems, Vol. 10, No. 4, November 1992, pp. 311-337.

[3] David P. Anderson: Metascheduling for Continuous Media. ACM Transactions on Computer Systems, Vol. 11, No. 3, August 1993, pp. 226-252.

[4] Apple Computer, Inc.: Inside Macintosh: QuickTime. Addison-Wesley, 1993.

[5] Andrew Campbell, Geof Coulson, Francisco Garcia and David Hutchison: A Continuous Media Transport and Orchestration Service. Proceedings ACM SIGCOMM '92, Baltimore, MD, August 1992.

[6] Tzi-cker Chiueh and Randy H. Katz: Multi-Resolution Video Representation for Parallel Disk Arrays. ACM Multimedia 93 Proceedings, August 1993, pp. 401-410.

[7] Luca Delgrossi, C. Halstrick, D. Hehmann, R.G. Herrtwich, O. Krone, Jochen Sandvoss and Carsten Vogt: Media Scaling for Audiovisual Communication with the Heidelberg Transport System. ACM Multimedia 93 Proceedings, August 1993, pp. 99-104.

[8] G.D. Drapeau: Synchronization in the MAEstro Multimedia Authoring Environment. ACM Multimedia 93, August, 1993, Anaheim, CA., pp. 331-340.

[9] Jim Gemmell and Stavros Christodoulakis: Principles of Delay-Sensitive Multimedia Data Storage and Retrieval. ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992, pp. 51-90.

[10] Rei Hamakawa and Jun Rekimoto: Object Composition and Playback Models for Handling Multimedia Data. Multimedia Systems, Vol. 2, No. 1, June 1994, pp. 26-35.

[11] B.B. Hehmann, M.G. Salmony and H.J. Stuttgen: Transport Services for Multimedia Applications on Broadmand Networks. Computer Communications, Vol. 13, No. 4, May 1990, pp. 197-203.

[12] M.E. Hodges, R.M. Sasnett, M.S. Ackerman: A Construction Set for Multimedia Applications, IEEE Software, January 1989, pp. 37-43.

[13] David Hutchison, Geoff Coulson, Andrew Campbell and Gordon S. Blair: Quality of Service Management in Distributed Systems. Tech. Rep. MPG-94-02, Lancaster University, 1994.

[14] K. Jeffay, D.L. Stone, T. Talley, F.D. Smith: Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, November 1992, pp. 1-12.

[15] A. Lazar, G. Pacifici: Control of Resources in Broadband Networks with QOS Guarantees. IEEE Communications Magazine, October 1991.

[16] Didier Le Gall: MPEG: A Video Compression Standard for Multimedia Applications. CACM, Vol. 34, No. 4, April 1991.

[17] T.D.C. Little and A. Ghafoor: Network Considerations for Distributed Multimedia Object Composition and Communication. IEEE Network Magazine, November 1990, pp. 32-49.

[18] T.D.C. Little, A. Ghafoor: Interval-Based Conceptual Models for Time-Dependent Multimedia Data. IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 4, August 1993, pp. 551-563.

[19] P. Lougher and D. Shepherd: The design of a storage server for continuous media. The Computer Journal, Vol. 36, No. 1, February 1993, pp. 32-42.

[20] C.W. Mercer, S. Savage and H. Tokuda: Processor Capacity Reserves: Operating System Support for Multimedia applications. Proceedings of the International Conference on Multimedia Computing and Systems, May 1994, pp. 90-99.

[21] Jason Nieh, James G. Hanko, J. Duane Northcutt and Gerard A. Wall: SVR4UNIX Scheduler Unacceptable for Multimedia Applications. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, November 1993, pp. 35-47.

[22] Calton Pu and Robert M. Fuhrer: Feedback-Based Scheduling: a Toolbox Approach. Proceedings of Fourth Workshop on Workstation Operating Systems, October 1993.

[23] K.K. Ramakrishnan, Lev Vaitzblit, Cary Gray, Uresh Vahalia, Dennis Ting, Percy Tzelnic, Steve Glaser and Wayne Duso: Operating System Support for a Video-On-Demand File Service. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, November 1993, pp. 225-236.

[24] G. van Rossum, J. Jansen, K.S. Mullender, and D.C.A. Butlerman: CMIFed: A Presentation Environment for Portable Hypermedia Documents. ACM Multimedia 93, pp. 183-188.

[25] Lawrence A. Rowe, Ketan D. Patel, Brian C. Smith, Kim Liu: MPEG Video in Software: Representation, Transmission, and Playback. High Speed Networking and Multimedia Computing, IS&T/SPIE, February 1994.

[26] R. Staehli, J. Walpole: Constrained-Latency Storage Access. Computer, Vol. 26, No. 3, March 1993, pp. 44-53.

[27] Ralf Steinmetz and Clemens Engler: Human Perception of Media Synchronization. Tech. Rep. 43.9310, IBM European Networking Center, 1993.

[28] Andrew S. Tanenbaum: Computer Networks – 2nd Edition. Prentice-Hall, 1988, pp. 278.

[29] Hideyuki Tokuda, Yoshito Tobe, Stephen T.-C. Chou and José M.F. Moura: Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network. SIGCOMM '92, 1992.

[30] Hideyuki Tokuda and Takuro Kitayama: Dynamic QOS Control based on Real-Time Threads. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, November 1993, pp. 113-122.

[31] Hideyuki Tokuda: Operating System Support for Continuous Media Applications. In Multimedia Systems, Ch. 8. Addison-Wesley, 1994, pp. 201-220.

[32] H. M. Vin and P. V. Rangan: Designing a Multi-User HDTV Storage Server. IEEE Journal on Selected Areas in Communication, Vol. 11, No. 1, January 1993.

[33] Gregory K. Wallace: The JPEG Still Picture Compression Standard. CACM, Vol. 34, No. 4, April 1991.

[34] Marek Wernik, Osama Aboul-Magd and Henry Gilbert: Traffic Management for B-ISDN Services. IEEE Network, September 1992, pp. 10-19.

[35] H. Zhang and D. Ferrari: Improving Utilization for Deterministic Service in Multimedia Communication. Proceedings of the International Conference on Multimedia Computing and Systems, May 1994, pp. 295-305.