A System for Using Perceiver Input to Vary the Quality of Generative Multimedia Performances

A Thesis
Presented to
The Academic Faculty

by

Byron A. Jeff

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2005

A System for Using Perceiver Input to Vary the Quality of Generative Multimedia Performances

Approved by:

Dr. Karsten Schwan, Chairman
College of Computing
Georgia Institute of Technology

Dr. Mustaque Ahamed
College of Computing
Georgia Institute of Technology

Dr. Kishore Ramachandran
College of Computing
Georgia Institute of Technology

Dr. Sudha Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. W. Gerry Howe
Clark Atlanta University

Date Approved: September 12, 2005

To

Mr. Morris F.X. Jeff Sr.

Reverend Richard W. Calvin Sr.

and

Dr. Morris F.X. Jeff Jr.

– my inspiration.

PREFACE

I have learned a great deal in the process of getting my PhD. I learned that sometimes I can be paralyzed by the seeming immensity of the task. I also learned that I can work past that by focusing on one small task at a time. I learned that some work can be done in solitude. However, I also learned that I need collaboration in order to be a complete researcher. I learned that advisors and mentors provide valuable assistance in the process. However, I also learned eventually I had to take ownership of my research. Usually a preface is about the research in the document. This preface is about the researcher.

ACKNOWLEDGEMENTS

I wish to thank my beautiful and supportive wife, Karen. Without her the paper would be meaningless. I also wish to thank our wonderful children: David, Jamila, Jaelyn, and Jihan. I did this for you guys too. Thanks to my mother, Florence. Everyone else in the family has been great as well.

I wish to thank Karsten Schwan and the other members of my committee. We have worked together on this for an extended time. I have appreciated their continuing support in my long struggle.

While working at Clark Atlanta University, I received support and encouragement from my many colleagues there. In particular, Ken Perry and Gerry Howe gave generously of their time and expertise. The institution also provided support in the form of leave and release time.

I am currently at Clayton State University. They gave me a chance when I really needed one. My thanks go out to the entire faculty of the College of Informational and Mathematical Sciences. I hope our strong relationship continues and that I may provide as much assistance to you as you have provided to me.

Other people have contributed greatly to this effort. Vernard Martin shoved me where I needed to go. He believed even when I didn't. Jeanette Allen showed me the researcher that I am, and the researcher that I hope to be. The I.N.C.I.T.E. lab crew gave feedback, provided suggestions, critiqued work, and shared more than I deserved. Thanks.

This work is the product of too many people to mention. I appreciate each and every one of you. It could not have happened without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Generative Multimedia (GM) applications are an increasingly popular way to implement interactive media performances. Our contributions include creating a metric for evaluating Generative Multimedia performances, designing a model for accepting perceiver preferences, and using those preferences to adapt GM performances. The metric used is imprecision, which is the ratio of the actual computation time of a GM element to the computation time of a complete version of that GM element. By taking a perceiver's preferences into account when making adaptation decisions, applications can produce GM performances that meet soft real-time and resource constraints while allocating imprecision to the GM elements the perceiver least cares about. Compared to other approaches, perceiver-directed imprecision best allocates impreciseness while minimizing delay.

CHAPTER I

INTRODUCTION

1.1 Problem Statement

Generative Multimedia applications are emerging as an important technique for implementing interactive media performances. Examples of Generative Multimedia applications include Interconnected Musical Networks [48], Video Conferencing using Video Avatars [24], Massively Multiplayer Online games [25], and Augmented/Virtual Reality Environments [50]. Current implementations process input events in real-time under unloaded conditions. However, under loaded conditions the algorithms that generate the digital media streams require more resources than are available. Under those conditions a number of undesirable effects occur including delay, jitter, or drops.

1.1.1 Generative Multimedia

Generative Multimedia (GM) is media such as audio and video that is algorithmically produced in soft real-time. GM output is triggered by events that are introduced from outside of the system. The defining characteristic of generative multimedia is that the media performance does not preexist at the time of the performance. Instead, a Generative Multimedia performance is generated in soft real-time. This differs significantly from preproduced performances delivered via video on demand [31] or streaming audio/video [30], both of which exist prior to playout and delivery. Generative Multimedia is analogous to live stage performances delivered to the audience, henceforth known as the ***perceiver***.

Generative Multimedia has several useful characteristics. First, trigger events are much smaller than the resulting performance triggered by the events. A few bytes specifying such events can map to several megabytes of output in the final performance. Second, Generative Multimedia performances can utilize the ever increasing computational power at the perceiver endpoint [45]. Third, the structure of the performance can be changed

during the performance. For example, a perceiver may really like the chorus of a particular song. In a preproduced performance, the length of the chorus is fixed. However, in a generative performance, the chorus could be lengthened during the performance.

A representative Generative Media application is a digital audio synthesizer. A synthesizer accepts input events for each instrument represented in the piece and renders a digital audio output for each such event. The synthesizer mixes the individual digital audio streams into a single output stream which is converted into analog audio format and presented to the perceiver. A model of such a synthesizer is shown in Figure 1.

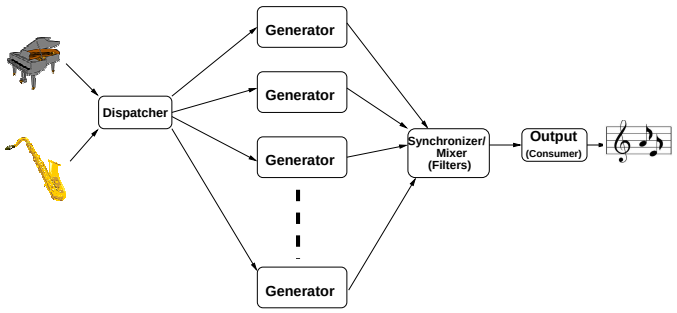

Figure 1: Digital Audio Synthesizer Structure

Events are received in real-time by the dispatcher. The synthesizer generates digital audio based on this input. Since the synthesizer receives this stream of events in real-time, the synthesizer has no a priori knowledge of the number or duration of the events to be processed until arrival.

Once the synthesizer has received an event, a generator maps the event into a digital audio stream. The synthesizer then performs other processing, including scaling the stream to the appropriate volume, synchronizing it with other streams currently being generated, and mixing them into a single final digital audio stream. The final output is presented to the perceiver via appropriate digital to analog conversion hardware, such as a PC sound card.

Our digital synthesis application exhibits properties common to applications in the Generative Multimedia domain. First, the generative actions of such applications are triggered by events in soft real-time. Second, the production of the digital output is based on potentially complex algorithms with resource requirements that are not known prior to the

performance.

An application in the Generative Multimedia domain such as a digital audio synthesizer processes input events with soft real-time output. As long as sufficient computational resources are available to execute the algorithms, the application does not need to balance competing demands. Algorithms generate the digital audio streams associated with the current set of events in the system. Event arrival times and execution duration are not available to the application a priori. Therefore, the application can't predict the computational resource requirements necessary to process the event stream for a particular performance. Thus, at some points, computational resources may be insufficient to complete all tasks fully.

Software synthesizers such as Csound [6],[33] often do not guarantee real-time response. But instead of failing to generate the requested output, CSound takes whatever time is required to generate all of the digital audio streams requested. If this time exceeds the duration of a single sample, the performance will be interspersed with pauses, clicks and pops. In all cases, the overextension of computational resources by applications in this domain presents a degraded performance to the perceiver.

1.1.2 Application-level Characteristics

Generative Multimedia applications are characterized by a need for:

- *Soft Real-Time Performance Requirements* - the performance is generated while the perceiver is engaged with it. Therefore, a limited window exists between the arrival of an event and the required generation of the output for that event. An excessive delay in the production of the output impacts the perceived quality of the performance.

- *Lack of A Priori Knowledge* - A source external to the application presents the events to the application. Therefore, the events the application processes during the performance cannot be known to the application. The application does know the class of possible events that it can handle. However, the quantity and arrival times of the events for a particular performance are unknown.

- *Variable Computational Requirements* - Since events are not predictable, their computational resource requirements are also not predictable.
- *Confined Resources* - The resources, including memory and CPU cycles, required to generate the performance are fixed during the performance.

The confluence of these characteristics creates an application environment that may prohibit the generation of a complete multimedia performance in soft real time with the available resources.

1.2 Existing Resource Constraint Approaches

Some approaches for solving the resource constraint issues of Generative Multimedia include:

- *Best Effort* chooses to produce a complete performance, typically using the highest cost methods.
- *Minimal Effort* uses least cost methods for processing each event.
- *Voice Stealing* chooses to reallocate resources from old events to new events.

Each of these approaches has weaknesses that our research addresses. Best effort violates the soft real-time constraint when resources are insufficient to generate the complete performance. Minimal effort underutilizes available resources and may deliver unsatifactory results to the perceiver. Voice stealing ignores perceiver preferences.

The general problem with these existing approaches is that they hardwire adaptations such as drops, delays, or resource reallocations. Fixed adaptations are used to tradeoff dynamic quality vs. resource utilization in a specific way. None of the existing approaches utilizes perceiver input before or during the performance. The specific approaches do not facilitate the perceiver having a decision in how adaptations are applied.

1.3 Metrics for Evaluation of Generative Multimedia Performances

We use two different measures to compare the effectiveness of Generative Multimedia constraint approaches. The first is impreciseness, which is the relationship of the difference

between a complete performance and the actual performance.The second is a measurement of overall delay during the performance.

In the general case, impreciseness is the ratio of resources used to produce an incomplete media element to the resources required to render that media element completely. This abstract definition holds across most computational domains, which is not true for relative metrics that compare complete to incomplete media elements produced by imprecise processes. Consider the production of an ornate tapestry graphic, for example. Here, a relative incompleteness metric would measure the quality difference of each pixel of the produced tapestry artifact compared to each pixel of the complete artifact. Such difference measurements must be provided by the particular application being run and will therefore differ across domains.

A set of tasks is required to generate the set of media elements for a frame. Delay is the amount of time taken to complete the set of tasks beyond the frame deadline. If the set of tasks completes before the end of the frame, then the delay for that set is zero. However, when the computation extends into subsequent frames, delay is exhibited.

Let impreciseness be defined by I. A complete performance would finish with $I = 0$ for each media element. Let delay be defined by D. A soft real-time performance would finish with $D = 0$ for each media element.

Let us define the approaches outlined above in terms of I and D.

- Best effort: This method produces a complete performance with no impreciseness. Best effort does not not take delay into account. So, when the production of a media element with no impreciseness takes longer than a frame to produce, then delay results in the performance. So, in Best Effort $I = 0$ and D is variable for the set of media elements produced in the performance.

- Minimal effort: This method maximizes impreciseness in an effort to minimize delay. Each media element is produced with the minimum resources that can be allocated to produce that media element. For example, in the audio domain each note is produced with a single partial. The problem with this approach is that resources that are

available during the performance may be underutilized being held in reserve just in case a rush of events come in. So, for minimal effort I is maximized for each media element in an effort to ensure that $D = 0$ for each element. In addition, with overload, you may have $D > 0$ for some elements even when produced at minimal effort. This may be addressed by simply not admitting events that result in delays. Such strategies will result in event loss. Such loss may be unacceptable to the perceiver. We will examine this topic further in the Future Work section.

- Voice stealing: This method attempts to minimize delay while maximizing resource usage. Each media element is assigned maximum resources, making $I = 0$, as long as $D = 0$ for the frame. However, when the admission of an incoming event would cause $D > 0$ for the frame, then some of the resources assigned to existing events are shifted to the incoming event. The impact is that for some set of media elements in the frame $I > 0$. However, $D = 0$ for the entire set of media elements in the frame. Voice stealing is a hardwired approach to resource allocation. The hidden issue of the scheme is who chooses the reassignment of resources. In voice stealing, the composer of the performance makes the decision about how resources will be reallocated.

- Perceiver directed impreciseness: Perceiver directed impreciseness also allocates resources so that $D = 0$ for the entire set of elements in the frame. However unlike vioce stealing, the allocation of I for each media element is determined by perceiver input.

Allocation of impreciseness minimizes delay in general. Perceiver-steered allocation of impreciseness takes perceiver preferences into account. This strategy allows for the customization of the performance without reintroducing jitter, delay, and drops.

In a similar vein, an individual perceiver's concept of a perfect or complete performance may fall short of the objectively complete performance that could potentially be produced. So, the resource requirements necessary to satisfy the perceiver's conception of the perfect performance may be less than the requirements for an objectively complete performance. Since the objective of a Generative Media performance is to be perceived, adaptations that

attempt to match a perceiver's preferences for what appears to be a 'complete' performance constitute an interesting approach to dealing with resource limitations during the performance.

1.4 The PEDALS Model

In this thesis, we describe a dynamic resource management model, PEDALS, the Perceiver Dynamically Adapted Library System. PEDALS is a model for building Generative Multimedia real-time applications. Dynamic adaptations are applied to the generative elements used to produce real-time performances within the constraints of available computational resources. As events enter the system, the resources are dynamically distributed among the set of generators. Each generator receives a portion of the computational resources to continue performing in soft real-time. Such adaptations vary the impreciseness of the performance to the perceiver while minimizing delay. Perceivers will evaluate the resulting performances in idiosyncratic ways.

Perception is unique to each perceiver. So, applications implemented in the PEDALS model dynamically adapt to match what an individual perceives as a good adaptation. It does not try to impose the same concept of good adaptation upon every perceiver. Thus, PEDALS exposes adaptation choices to each perceiver. The perceiver can then choose the adaptations that he or she prefers.

PEDALS exposes adaptations to the perceiver as follows. First, PEDALS exposes the generative elements to the perceiver. The perceiver rates the desirability of different kinds and levels of impreciseness of each element output. Second, PEDALS uses preferences stated by the perceiver about the acceptable impreciseness of element outputs to dynamically allocate resources to minimize overall impreciseness. Thus, impreciseness is steered by the perceiver within the given resource constraints.

We dynamically adapt the computational requirements of the application to the available resources. We do this by varying the imprecision of the media elements generated to create the performance, trading preciseness of output for additional computation resources. By planning for graceful degradation of quality in the face of increasing workload, PEDALS

can address the shortcomings of each of the approaches outlined in the Existing Resource Contraints section.

Applications in the PEDALS model measure their resource requirements dynamically. However, instead of requesting additional resources when computational capacity is exceeded, they will adapt the computational requirements of the generators to match the available resources. Instead of failing to produce output in real-time as some static hardware/software synthesizers do, or failing to produce output for all events, as done in some hardware implementations, we produce output for all events in real-time, but with some elements of the performance produced with impreciseness relative to the fully instantiated performance.

1.5 Thesis Statement

Generative Multimedia requires adaptive solutions to generate high quality performances. Perceiver preferences can be used to dynamically control the impreciseness of adaptive Generative Multimedia performances.

1.6 Thesis Demonstration

We have developed the PEDALS model, which exposes the perceiver to adaptation options with variable impreciseness. We also defined metrics that quantify the impreciseness of an adapted Generative Multimedia performance. The model is demonstrated with an event-driven digital audio synthesizer called PARSYNTH. We have measured and compared the impreciseness and delay of performances using Best Effort, Minimal Effort, Voice Stealing and Perceiver Directed Imprecision. Results from testing PARSYNTH indicate that dynamic adaptation reduces delay and improves the allocation of impreciseness, over those approaches.

1.7 Summary

Our contributions include making explicit the imprecision of Generative Multimedia performances and permitting the perceiver to influence the dynamic adaptations applied. By

taking a perceiver's preferences into account when making adaptation decisions, applications can produce Generative Multimedia performances that meet soft real-time and resource constraints while allocating imprecision to the GM elements the perceiver least cares about.

We implement the PEDALS model, instrumented into PARSYNTH. [19] PARSYNTH can perform under soft real-time performance constraints, while generating perceiver-directed adaptations. Compared to other approaches perceiver-directed imprecision best allocates impreciseness while minimizing delay.

CHAPTER II

RELATED RESEARCH

In this section, we analyze previous research relevant to this thesis and compare it with the concepts we have outlined for the PEDALS model. We address related research in the following areas:

- *Variable Quality of Service (QoS)* is a common technique for allocating constrained resources.
- *Autonomic Computing* addresses issues of complex systems performance self adaptation in order to meet high level user specified goals.
- *Perceiver Quality* addresses issues related to how perceivers interpret the quality of multimedia performances, both in the generative and the distributed multimedia domains.
- *Soft Real-Time Metrics* evaluate the efficacy of adaptations performed during execution and to measure the potential negative impacts of performing such adaptations.

Our research is grounded in the rich tradition of distributed multimedia with variable QoS, adaptive systems, and soft real-time systems. In this section we will examine related research in those areas and how the PEDALS framework extends work in these areas using perceiver input.

2.1 PEDALS Contributions

PEDALS extends research in the following areas. It utilizes perceiver input to guide adaptation decisions and to manage allocation of imprecision. It measures imprecision. It is an adaptive framework that facilitate the building of perceiver-driven adaptive generative multimedia applications. Adaptation is based on frames.

2.2 Variable Quality of Service

Variable QoS management has been discussed in the area of distributed network multimedia [3], [8], [16], [28], [53]. In general, these papers focus on the architectural requirements of distributed multimedia. They span a variety of approaches including CORBA-based object management to end-to-end QoS management.

Koliver [22] manages variations in network resources using quality functions. The quality function is mapped from QoS parameters. The parameters generate an overall quality value that is adapted during the delivery of the networked multimedia performance. A fuzzy logic controller is used to implement adaptation. The controller monitors the actual quality value from the stream of interest and computes an error value to the quality value emitted by the quality degree function. Based on this error value, the controller adapts low level parameters, such as bit rate, for the stream.

Our PEDALS model follows a similar pattern in the generative multimedia domain. We utilize quality mapping tables that directly map perceiver preferences, as opposed to a quality degree function which uses QoS parameters that have user preference implicitly embedded in those parameters. Instead of utilizing fuzzy logic for adaptation, our frame-based adaptation uses a simple heuristic to compute adaptations between frames as necessary.

In both PEDALS and Koliver's systems, the perceiver's preferences are decoupled from the resource usage of the performance. Perceivers specify high level quality goals, and the underlying system performs adaptations to meet those goals using the available resources.

In [10] variable QoS is coupled with admission control. Their setup consists of dividing tasks into high priority and low priority multimedia tasks. Upon admission of a new high priority task, if sufficient resources are not available, then the quality of some subset of the lower priority tasks is decreased in order to make resources available for the high priority tasks. When resources become available, the lower priority tasks have resources restored to them. Resources are partitioned into fixed sets of high, low, and shared resources.

Their concept shares several attributes with PEDALS including variable QoS upon admission of new tasks, and the subsequent restoration of resources as they became available.

However, the arbitrary division of tasks into high and low priority limits the possible adaptation strategies. Also, resource underutilization is likely if the mix of high/low priority tasks is skewed due to resource reservation of the fixed partitions of the task areas. Finally, their variable QoS model does not allow for fine gradations of resource assignment based on perceiver preferences, as PEDALS does.

Another approach to Quality of Service is taken by Poellabauer [34]. With Q-Fabric, he takes an integrated and multilevel approach to adaptation, involving the application, operating system, and network levels. This approach results in highly efficient adaption. QoS managers can specify policies, and the resource manager (a component of Q-Fabric) uses kernel level mechanisms to implement those polcities. One novel aspect of his adaption algorithm is using the amount of energy required for a task as input to the adaptation decision coupled with the integrated, multilevel approach. Although PEDALS is currently implemented at the application level, it would benefit from Poellabauer's multilevel, integrated architecture.

BBN Technologies has done considerable research in the area of object-oriented Quality of Service. Among many others, [52] [47] [27] describe QuO, which is CORBA-based middleware for implementing objects with QoS properties. QuO allows distributed object-oriented applications to specify dynamic QoS requirements. QuO also provides an adaptation mechanism in order to ensure that QoS constraints are met. PEDALS shares the adaptive and QoS management properties of the QuO system. However, it is currently neither distributed nor object oriented.

Another approach to QoS is taken by DFuse [23]. DFuse is an architectural framework for managing data fusion of sensor networks. DFuse has a QoS component related to the power budget of the sensors in the network. The framework tries to equalize load to maintain longest total application running time without running out of power. DFuse dynamically assigns aggregation roles to nodes in the sensor network. This assignment is based on a cost function designed to optimize on attributes such as node power and transmission costs. PEDALS shares the management of QoS and dynamic adaptation features of DFuse. However, instead of energy, our resource is CPU capacity. In addition, instead of cost

functions as input for adaptation decisions, PEDALS utilizes perceiver data.

2.3 Autonomic Computing

As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components. As a result, such issues are dealt with at runtime. Systems have become too massive and complex for system integrators to install, configure, optimize, maintain and merge. This complexity results in an inability to make timely, decisive responses, to the rapid stream of changing and conflicting demands.

Autonomic Computing [21] is one way to address these issues. Autonomic Computing systems manage themselves given high level objectives from administrators. Autonomic Computing research [49] incorporates a strong adaptation component. Autonomic element behaviors include establishing and maintaining relationships with other autonomic elements, meeting obligations via tuning, and offering a range of performance options.

The PEDALS model shares these behaviors of autonomic elements. Modules in our framework establish relationships via the use of administrative links. Modules meet their obligations of both soft real-time and resource constraints by performing adaptations. Dynamically adapted generators offer a range of performance options.

White [49] describes other required behaviors for autonomic computing elements. They must be self-managing and self-protecting. However, our framework does not completely fall under this definition. Our framework utilizes a centralized dispatching system for adaptations. PEDALS modules are adaptable, but not self-initiating. In its current configuration, modules make no attempt to validate adaptation requests because adaptation decisions are centralized. Adaptation requests are currently assumed to be valid and reasonable. Therefore, modules are not self-protecting.

The underlying basis of autonomic computing is dynamic adaptation [4], [36]. Dynamic adaptation can be implemented at multiple levels: hardware, Operating System, middleware libraries, and application. The demand for scalability in distributed and real-time systems has driven the integration of dynamic adaptation research into autonomic computing applications. The PEDALS model currently implements its dynamic adaptation at the

application level.

In [37], Rosu develops a model for describing dynamic adaptations and their costs. FARA is her framework for implementing dynamic adaptation. FARA is implemented as a middleware layer. In particular, FARA describes a model of application-specific adaptation costs and methods for using this information in selecting adaptation solutions with low impact on transitory state performance. PEDALS utilizes perceiver input in adaptation decisions. However, PEDALS does not apply application specific information to reduce the transitory impact of adaptation decisions.

2.4 Perceiver Preference Mapping

In multimedia soft real-time systems (particularly network distributed systems), metrics are often based on objective measures such as end-to-end delay and jitter. Several systems [14], [12], [41], [5] use perceptual data to guide adaptation. Work in the area of perception, including [46] and [1], establishes quality thresholds beyond which perceivers gain no further perceptual benefit of additional allocated resources.

[15] examines the effect of frame rate upon perceiver satisfaction and information understanding. In their study, participants were asked to rank the quality of performances on a scale. The results indicate that significant changes in the frame rate of multimedia clips have no significant impact upon the perceiver's perception of quality. In comparison, PEDALS varies the impreciseness of the generative elements of a performance at a fixed frame rate. In addition, we use the metrics of impreciseness and delay to evaluate the performance.

2.5 Soft Real-Time System Metrics

Brandt [7] discusses that soft real-time system metrics have a different priority than hard real-time metrics, where missed deadlines cannot be tolerated. Instead, he points out that soft real-time systems can be compared using the concepts of measuring the benefit of resource allocation, and the instability of the system due to these dynamic changes in resource allocation. Instability is defined as the rapid change in the state of the multimedia

performance. For example, a video performance is unstable when it rapidly changes state between full color and black and white. Previous metrics did not take instability into account.

PEDALS utilizes the concept of benefit in the adaptation of generators, where benefit is based on the perception of the perceiver. PEDALS currently does not take instability into account. The combination of the frame based approach and the selection of adaptations among the generators limits extreme changes in individual adaptations. However, high overall instability may result from the aggregation of adaptations across the entire set of generators.

Brandt points out that often benefit and instability work counter to one another. In addition, the instability and benefit of a prematurely dropped event, as can occur in best effort systems, will more severely impact the total benefit and instability of the system as compared to a variation in the quality of the processing of that event. PEDALS exploits this concept by varying the impreciseness of the media elements generated for an event. This behavior is analogous to the variation in quality discussed in Brandt.

Another approach to evaluating soft real-time multimedia performance is described by Liu [26]. They built a system wherein tasks are divided into manditory and optional components. The computation of each task is checkpointed at intervals. When the deadline for the task is reached, the most recent checkpointed result is submitted. They refer to the goodness of their result as the precision of the result. A complete computation is a precise result. PEDALS uses a variation on this concept of precision. However, instead of checkpointing, PEDALS preschedules the execution time of its tasks.

2.6 Summary

The PEDALS model draws from research in many areas including Variable Quality of Service, Autonomic Computing, Perceiver Quality and Soft Real-Time Metrics. However, these areas do not specifically address the use of perceiver input to make adaptation decisions. This perceiver input is critically important in generative multimedia performances, where previous systems have either not made adaptation decisions or have made hardwired

Figure 2: EverQuest Low Quality Character Model Troll

adaptations. Hardwired adaptations do not take perceiver preference or other situational characteristics into account. An example of such hardwired adaptations is used in the Massive Multiplayer Online Game EverQuest.

Figure 3: EverQuest High Quality Character Model Troll

Figure 2 shows an example of a Troll implemented in EverQuest Low Quality Character Model. Figure 3 is the same Troll, in the same spot, but using a High Quality Model. The low quality models lack visual detail but can generally be rendered without delay. The high quality model shows rich detail. However, because of the resources required to render high quality models, player may experience slower reaction times in busy scenes, where many models must be rendered simultaneously.

The quality of models has minimal impact during scenes where a limited number of models need to be rendered. However, when scenes become busy, the game's frame rate drops steeply when using high quality models. This drop negatively impacts game play

for the player. Because EverQuest players cannot dynamically adjust model quality, they are forced to choose model quality before entering the game. Any adjustments to model quality requires leaving the game, changing model quality parameters, and then reentering the game. Such activities represent a severe impact to game play. During this time, the player cannot participate in the game. In a busy battle sequence, a lack of participation represents extreme danger both to the player character, and to the other players in that battle sequence.

Another aspect of the lack of perceiver input is illustrated by the modality of the EverQuest player character. Many EverQuest characters have multiple roles, such as Healer and Fighter. The view that a player requires during a session depends on the particular role that player is exercising. Healers need to have views and information on other players in their group while Fighters need to have views that focus on the enemies in a battle. Facilitating a player's ability to switch models to exchange views during the course of a game session improves the game play for that player.

These points show that exposing adaptations to the perceiver of a generative media performance can enhance the perceptual quality of the performance for that perceiver.

CHAPTER III

PEDALS DESIGN

We have developed the PEDALS model for building generative interactive soft real-time multimedia applications. PEDALS-based applications dynamically adapt generators used to create a performance. The generators vary the impreciseness of the performance in a perceiver-specified way. The imprecision of the performance offsets delay. PEDALS exposes adaptation choices to each perceiver. The perceiver can then choose the adaptations that he or she prefers.

3.1 Motivation

There are two major concepts driving the development of PEDALS. The first is adaptability. Generative media space applications must have adaptability. Such adaptability will afford our framework the required flexibility to generate soft real-time multimedia output in the face of varying input and resource conditions. The second major concept is perceiver ordering of impreciseness.

3.1.1 Adaptability

Different algorithms also can produce quantitatively different digital audio streams that have a subjectively similar tonality to the perceiver. For example, both FM synthesis and additive synthesis can produce piano tones. In addition, some algorithms can be imprecisely computed. [26] discusses the use of imprecise computation for fault tolerance and graceful dedgredation in real-time systems. [51] has a survey of algorithms of the imprecise computation model. A class of imprecise algorithms generates results by summing an infinite series of terms. These algorithms can approximate the target output with varying degrees of impreciseness. The more imprecise output corresponds to a reduction in the computational load required to produce that output.

As an example, consider Figure 4 through Figure 6.A square wave can be generated by

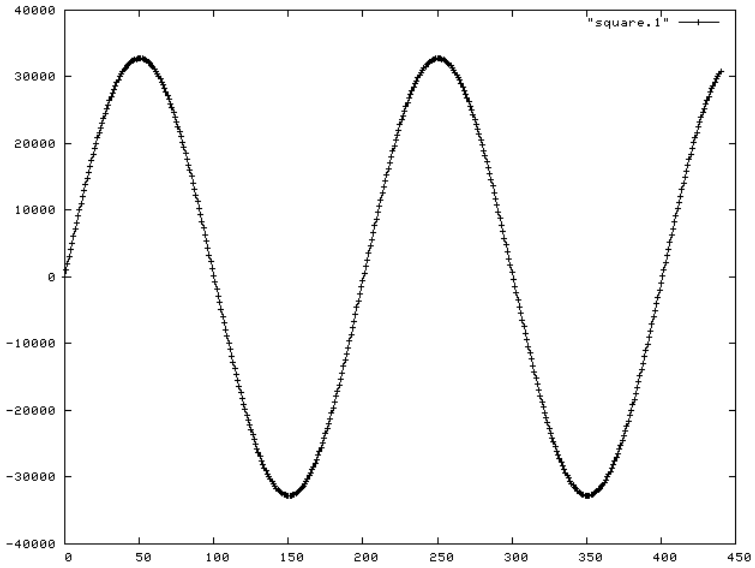

Figure 4: Square Wave Composed of 1 Sine Wave

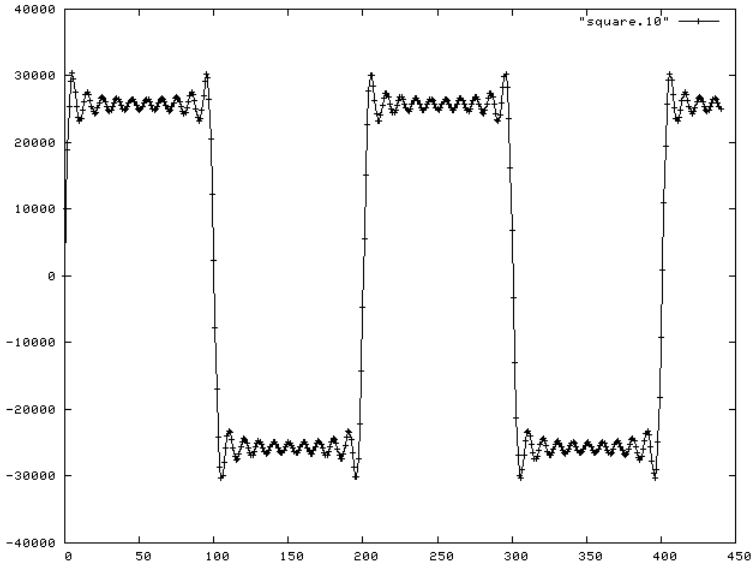

Figure 5: Square Wave Composed of 10 Sine Waves

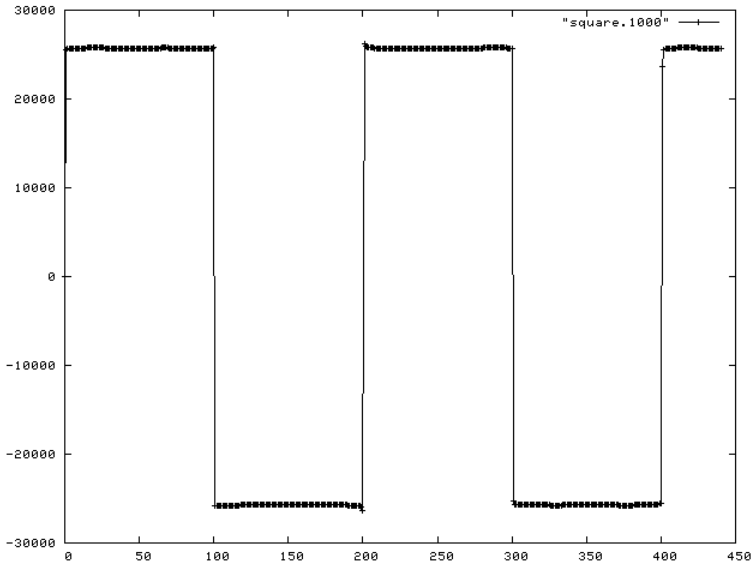

Figure 6: Square Wave Composed of 1000 Sine Waves

adding an infinite series of sine waves consisting of the odd harmonics of the fundamental frequency of the square wave. Note that a single sine wave at the fundamental frequency is a very crude approximation for the desired square wave. However, as more terms are added (10 and 1000 sine waves in Figure 5 and Figure 6 respectively), the resulting waveform more closely approximates the target waveform. For different perceivers, the subjective quality of these approximate tones varies. In addition, the computational requirements necessary to produce these tones also vary according to the algorithm producing the tones. The PEDALS model divides performances into frames. A frame is defined as a fixed computation time-frame that is used to generate a segment of the generative performance. Adaptations only occur between frames. Event admissions and releases only occur between frames. Frame intervals must be short enough so that adaptation response times can be relevant, but long enough so that adaptation computation doesn't overwhelm the computation requirements of the frame. The metric used for determining frame length is "Just Noticeable Difference" [29], which is the threshold for the perceiver to notice changes in the performance. Changes that occur below this threshold are not perceived as changes. Adapting faster than the perceiver can notice changes invalidates any advantage in increasing the adaptation rate.

Another issue is the fact that adaptations are event driven. As long as the system is in

a steady state where no additional events come along, no adaptations need to be executed. Furthermore, no evaluation for adaptations needs to be done. Generators processing existing events in this steady state use the same adaptation parameters issued the last frame that adaptation occured.

3.1.2 Perceiver Ordering of Impreciseness

The next conceptual element is the ordering of the perceived impreciseness of the possible generation options. However, such preferences are perceiver-specific. The perceiver experiencing the performance must assign a perceptual impreciseness value for each possible generation option. The ordering of perceived impreciseness is chosen solely by the perceiver's preference, without regard to the computational requirements to generate that preference.

The two conceptual elements of adaptability and perceiver ordering of impreciseness provide the basis by which generative interactive multimedia applications may be adapted. The framework and the applications offer the perceivers parameters for each class of tones. The PEDALS adaptation algorithm selects parameters that meet the computational requirements for each frame of the performance. The applications using the PEDALS model make these adaptive selections based on the preference assignment of the perceiver.

Our approach is to create a design with application mechanisms, framework, and policy, such that applications built using the PEDALS model will generate generative media output whose impreciseness is directed by the perceiver. In addition, the design and resulting infrastructure framework will simplify the process of building dynamically adaptable generative applications.

We also differentiate between reconfiguration, which changes the structure of the application, and adaptation, which changes the resource requirements of the application without necessarily changing the structure. Reconfiguration (adding or removing an instrument) causes adaptation of the rest of the application's modules to compensate for the reconfiguration.

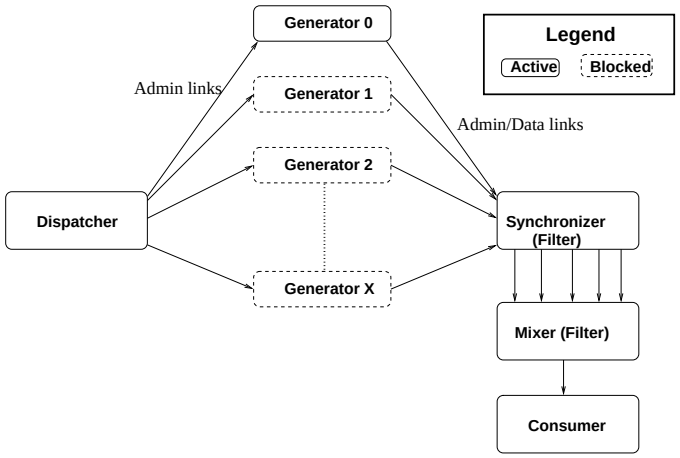

Figure 7: PEDALS Based Application Architecture

3.2 PEDALS Architecture

As shown in Figure 7, a PEDALS-based adaptive application consists of modules and links. Modules perform multimedia application specific activities such as generating, filtering and consuming media element data. Data and events directing module activity are passed between modules using links. This structure creates an application infrastructure that can be dynamically adapted.

3.2.1 PEDALS Events

Events are used to direct the actions of the modules of the application. The use of events unifies the communications infrastructure of a PEDALS-based application. The same event structure is used for both intramodule and extramodule events. This usage creates the ability to direct the application's modules using modules external to the application. The six kinds of events are:

- START events can be parameterized to vary the content generated. For example for a digital synthesizer application, a START event can carry the frequency and volume parameters for the note to be generated.
- STOP events terminates the generation of a media element. STOP events return the corresponding generator for that media element to an idle state.

- ADAPT events trigger an adaptation to the generator. ADAPT events can be parameterized to specify the algorithm that should be used to process the media element in subsequent frames.

- NEWWORK events facilitate frame based flow control. A NEWWORK event accompanies the data associated with a media element for a single frame.

- CONTINUE events facilitate are also frame based flow control. CONTINUE events are acknowledgements for NEWWORK events. CONTINUE events signal that work for the next frame can commence.

- DIE events permanently terminate the module receiving the event. Before terminating, the module retransmits the DIE event to any downstream modules. The net effect is to terminate all modules in the PEDALS-based application.

3.2.2 PEDALS Links

Links are the mechanism used to transport events and data between modules. Links are point to point one way communications channels. Links support both reconfiguration and adaptability. Reconfiguration is achieved by redirecting the sender and/or receiver of a link's content dynamically. Adaptability is supported by the link's ability to transport adaptation events between modules. Adaptation events are issued by the dispatcher and processed by generators between frames.

Links in the PEDALS model are differentiated by the type of information they carry. Data links are used to transport application data (samples, video frames, etc.) between modules. Administrative, or admin, links are specifically designed to transport events. Separation of data and admin links contributes to simplicity and performance. Admin events on separate channel are faster and simpler than having to locate admin events from a single unified link. Unified links could prioritize so that admin events are pushed to the head of the link, but that technique adds complexity.

3.2.3 PEDALS Modules

Modules are the computational elements of the generative application. They receive events from their administrative link. These events direct the module to perform their specified duties. Modules then generate events, specifically NEWWORK and DIE events, to direct the modules that are receiving the data generated or filtered by the module generating the data.

Modules in a generative application are differentiated by their execution activities and execution models. The four classes of modules are generators, filters, dispatchers, and consumers.

The first type of module are dispatchers. Dispatchers accept incoming events from outside the application and process them. As events arrive, the dispatcher sends START events to generators to initiate processing for those events. The dispatcher executes the adaptation algorithm to adapt the generators processing existing events. ADAPT events are sent to those generators as needed.

Generator modules generate data content under the direction of events received from the admin link. The START event directs a generator module to start generating data content. A generator module will continue to generate data content until a STOP event is received.

The next type is a filter module. Filter modules take one or more input data streams, filter the data, and generate one or more output streams.

Consumers are endpoints for data streams. Consumers take input data streams and transport them out of the application. Examples include writing the data stream to a file or presenting the final data stream to the perceiver.

3.3 Adaptation Algorithm

In an ideal situation, we would like to optimize the preference values of the performance given resource constraints. However, this optimization is an NP complete problem. Instead we use an approximation.

3.3.1 The Knapsack Problem

The Knapsack problem [38] is a classic integer optimization problem. For a set of N items such that each item i is worth profit p_i, and weights w_i, choose the subset of items such that the total weight is at most W and the total profit is maximized. Finding the optimal set of items is a known NP complete problem. However, many approximations algorithms have been proposed. See [17] for a survey of such algorithms.

3.3.2 Definitions

In order to describe our adaptation problem, we must introduce some terminology describing attributes of preference and cost.

- The perceiver must supply a preference value to each possible generation option. We call this preference (p).
- Each generation option also has a computation cost (c).
- The Total Preference (TP) is a quantitative representation of the total preference of all of the currently running generation options in a frame. $TP = \sum_{i=1}^{n} p_i$.
- The Total Cost (TC) is a quantitative representation of the total cost of all of the currently running generation options in a frame. $TC = \sum_{i=1}^{n} c_i$.
- The Available Resource Cost (ARC) represents the total computational capacity on the system the framework is running for a single frame.

3.3.3 Adaptation Problem Definition

We wish to chose the TP such that TP is maximized while $TC <= ARC$. This constrained optimization maps directly to the Knapsack problem. The computational complexity of choosing the optimal solution is prohibitive. As a result we have chosen one of many approximations. We chose an approximation on the basis of speed and simplicity. Other approximation algorithms could be substituted without loss of generality.

One additional constraint unique to this problem domain is that generation options are subdivided into classes. For each of the n events an option must be chosen from a specific

class. In the original Knapsack problem, no item was required to be chosen. But since events must be processed, a generation option must be mapped to that event.

In order to be explicit about which set of generation options are available for which types of events, both events and generation options will have an additional subscript t to indicate the mapping between a particular event type and its corresponding generation options set. For example, E_{it} and G_{tji} would indicate the event i uses generation option j from the generation options subset t. The subscript i to E and G is needed when multiple instances of the same generation option are assigned to different events.

The adaptation algorithm runs prior to every frame that has an incoming event. The algorithm must be efficient because the performance is generated in soft real-time. The algorithm executes as follows:

> Create a vector of N generation options, one for each event in the frame. The generation option must match the event type of the corresponding event and starts as the generation option with the highest p. While TC for the vector is greater than ARC and some lower cost generation option exists, replace a generation option with the generation option (from the same event type set) with the smallest decrease in preference. If multiple generation options with equal preference difference exist, choose the generation option with the smallest absolute preference.

For example, consider Table 1. If the $ARC = 750$ and a piano note event arrives, then the adaptation algorithm selects Piano generation option A. When a second saxophone note event arrives, the adaptation algorithm must choose a piano option and sax option. It will start with Piano A and Sax C ($TC = 1108$). The TC exceeds the ARC, so the algorithm continues. The algorithm will select Piano A and Sax D ($TC = 869$). The sax option is chosen because the difference in preference between Sax C and Sax D is smaller than the difference between Piano A and Piano B. The next selection is Piano B and Sax D ($TC = 732$). The algorithm ends by choosing generation options Piano B and Sax D for the two notes.

Table 1: Sample Adaptation Table

Generation Option	Preference	Cost
Piano A	900	600
Piano B	700	463
Sax C	600	508
Sax D	525	269
Sax E	275	103

The resulting vector of generation options represents an approximation of the highest total TP that can execute within the given ARC. The run time of the algorithm is $N * M$ where N is the number of events and M is the upper bound of the number of generation options among the generation option subsets. This algorithm does not find the optimal set of generation options because it does not implement backtracking. Once the preference for a generation option has been decreased, the algorithm does not allow for a higher preference generation option for that event to be selected at a later time. However, each new incoming event allows for a new recalculation of the generation options. Potential improvement of a generation option can occur in subsequent frames.

3.4 An Example of PEDALS-based Adaptation

Figures 8 through 10 illustrate snapshots of a sample execution of the PEDALS instrumented application. The figures show the impact on a PEDALS application of the events described in the previous section. For this example we utilize the preference and cost values described in Table 1. Figure 8 shows the initial idle state of the system. Each of the generators are preforked, but blocked on their administrative links, which are used to communicate event and adaptation information to the computation elements. The generators consume no computational resources during the execution of the application until activated by the arrival of an event on the administrative link of the generator.

Figure 8 also shows the activation of the silent generator. This generator exists as a timing mechanism for the application in the idle state. This occurs at the beginning of the execution of the application or when all active events have been satisfied. The silent generator generates digital audio samples with no volume, thereby creating silent output. In

addition the dispatcher monitors the output packet number of the consumer module. When the output packet number matches the packet number of the next event, the dispatcher injects that event into the system.

Figure 9 shows the arrival of the first piano NOTEON. The dispatcher runs the adaptation algorithm and assigns the generator with the value in the adaptation table with the highest perceiver preference for the specified instrument: 900.

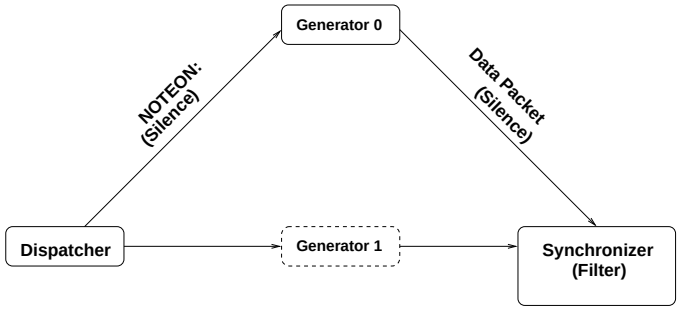

Figure 8: System Idle State

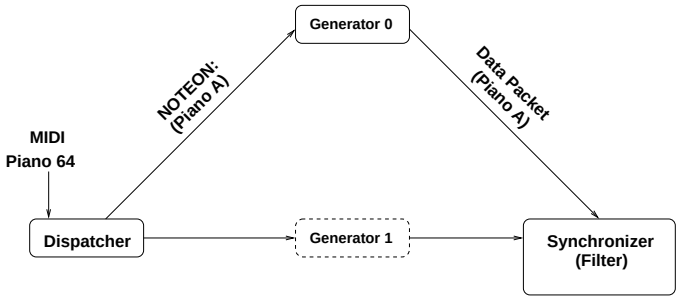

Figure 9: Introduction of First Note

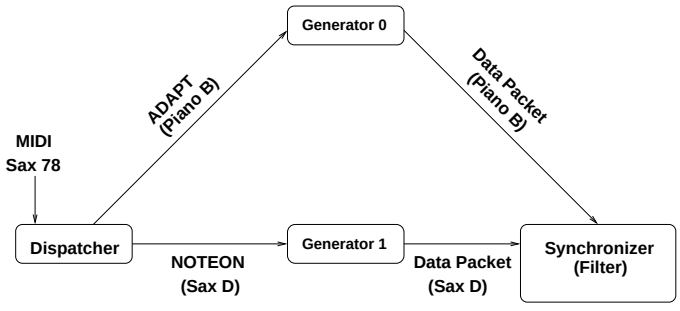

Figure 10: Adaptation when Second Note Added

Figure 10 illustrates the arrival of a second NOTEON event, a sax note, after an initial NOTEON to Generator 0 has already initiated processing. The initial NOTEON when

running alone had sufficient computational capacity to run at the maximum 600 (Piano A) indicated in the adaptation table. When the second NOTEON arrives the dynamic adaptation algorithm selects Piano B (cost 463) for the existing NOTEON event and Sax D (cost 269) for the new event. The dispatcher sends an ADAPT event of Piano B to the currently running Generator 0, and a NOTEON with algorithm Sax D to the new Generator 1. The adaptation has varied the adaptation of the generator generating the existing NOTEON event.

3.5 Summary

We described the PEDALS model for building generative interactive soft real-time multimedia applications. PEDALS uses dynamic adaptation to create a performance with imprecision and minimum delay. The PEDALS architecture consists of events, links, and modules. The adaptation algorithm is a variation of heuristic solutions designed for the knapsack problem. We discussed one application specific approximation and gave its runtime. Finally we worked through an extended example of the algorithm and then demonstrated that example for a sample PEDALS application.

CHAPTER IV

DIGITAL AUDIO SYNTHESIS

Digital Audio Synthesis consists of several components including synthesis algorithms, sine wave generation and representation, amplitude envelopes, note and score representation, and digital mixing. In the following sections we describe each of these components.

4.1 Synthesis Algorithms

Digital audio synthesis uses mathematical formulas to create a stream of digital samples. The sample stream generates sound when processed by a digital to analog converter. In an ideal environment, each sample would be individually created to match the exact sound desired. Unfortunately, the excessive storage and computational requirements preclude using this technique. Instead, algorithms generate the sample streams. Three broad classes of algorithms used are Physical Instrument Modeling, Frequency Modulation Synthesis, and Additive Synthesis. Each of these classes of algorithms have the ability to generate a wide variety of sounds. Each has a relatively simple parameterization of synthesis information. They can model natural musical tones.

In the following section, we will briefly examine each class of algorithm. In addition, we will examine the applicability of each algorithm type for adaptation in our framework.

4.1.1 Physical Instrument Modeling

Physical instrument modeling [44] uses wave theory and the physical changes to the waves caused by the embouchure, reed, and walls of the instrument to model the generation of the wave synthetically. Digital waveguides are more computationally efficient than the models of the sound. Adaptation of the technique is not straightforward. Due to the switching from integration of each point/sample to a digital waveguide, computation is decoupled from the number of digital audio samples produced.

4.1.2 Frequency Modulation

Frequency Modulation [11] utilizes the convolution of two or more sine waves, called operators, to generate inharmonic components that more closely model natural sounds than pure sine waves. Variations of the frequency of the component sine waves, the modulation index between the convoluted waves and the structure of the algorithm, provide the designer with a flexible system that produces a wide variety of sounds.

The difficulty in using FM synthesis is the lack of a clear mapping between algorithm parameters and the sounds produced. FM synthesis is difficult to adapt because small changes in the parameters or the number of operations leads to large changes in the timbre of the resulting output.

4.1.3 Additive Synthesis

Additive synthesis is the process by which complex periodic waveforms are created by the summation of a set of sine waves. Each sine wave component is parameterized by its frequency, amplitude, and phase. There are two methods of generating complex waveforms from individual sinusoid parameters. The first method simply generates each individual sine wave given the frequency, amplitude, and phase parameters and sums up the results. The second method [13] represents all of the sine parameters as frequencies and performs an Inverse Fast Fourier Transform (IFFT) upon the data. The IFFT produces the resulting complex waveform. The IFFT computation time does not depend on the number of sine waves calculated. Integrating IFFT into a performance is somewhat complex because the algorithm requires a sample window that is a power of two. In addition, noise can be introduced between sample windows.

Serra [42] explains that additive synthesis coupled with a residual noise component has the ability to generate any type of sound, both harmonic and inharmonic. In addition the set of parameters required to generate a particular sound can be extracted from a sample of the sound itself.

The major benefit of additive synthesis is that the number of individual sine waves required to generate complex waveforms can be varied. This variability gives PARSYNTH

the ability to adapt the computation time for each individual sound generated. Also the time required for each additional sine partial is linear. The execution time for a particular number of partials can easily be calculated.

The decrease in the number of sine waves produces a waveform that differs from the complete waveform. Therefore, impreciseness is introduced.

However the IFFT procedure can be utilized to generate all of the possible component sine waves without an additional marginal cost to the algorithm computation time.

4.2 Sine Wave Representation and Generation

Additive synthesis uses sine waves to build complex waveforms. Natural timbres are described by harmonic, amplitude, and phase information for each sine wave. In this section we describe sine wave generation techniques. We also outline the method for obtaining sine wave parameters for natural tones.

4.2.1 Sine Wave Generation

Efficient sine wave generation is important for maximizing the resource usage during a performance. In [9] several techniques for sine wave generation are compared. These techniques are:

- Using an interpolated look up table.
- Using the built in library sine function.
- Using a ringing infinite Q filter.

The interpolated look up table has the advantage of computation speed. However, errors in the interpolated values can cause distortion in the digital audio stream produced. The built in library sine function is generally easy to use. On the other hand, the combination of the set up costs to call a function coupled with the inefficient implementation can increase the computational cost to use the function.

We choose to use the ringing filter technique because it efficiently generates sine waves without accumulating long term errors. Typically, with integer computation, roundoff error

accumulates. The ringing Q filter technique does not.

4.2.2 Sharc Database

In order to reproduce realistic sounding timbres, additive synthesis requires parameterized information for each of the sine wave partials used to generate the sound. These parameters include harmonic, amplitude, and phase information. We obtained this information from the SHARC timbre database.

SHARC [39] is a database of musical timbre information by Gregory Sandell. It stands for "Sandell Harmonic Archive." Over 1300 different notes have been analyzed. Complete chromatic runs from the standard playing range of essentially all the non-percussive instruments of the modern orchestra have been included; for example, individual analyzes of 32 different oboe notes (the chromatic scale from the pitches a3 to f6) are available. For each note, a short portion corresponding to the sustain or "steady state" portion of the tone was selected and analyzed with a Fourier analysis. Each analysis consists of a list of amplitudes and phases for all the note's harmonics in the range 0-10,000 Hz. The source of the musical notes were the orchestral tones from the McGill University Master Samples (MUMS) Compact Discs. These are digital recordings of live musical performers.

4.3 Amplitude Envelopes

All of the digital audio synthesis methods outlined above fail to produce realistic sounding output when used in isolation. The physical models of instruments exhibit changes in amplitude during the generation of a tone.

Jensen [20] describes an envelope of partials as the evolution over time of the amplitude of a sound. It is one of the important timbre attributes. A faithful reproduction of a noiseless sound with no glissando or vibrato can be created using the individual amplitude envelopes of the additive parameters. Unfortunately, the analyzed amplitude envelopes often contain too much information to be easily manipulated. A model of the envelope is therefore necessary.

Jensen's envelope model is relatively simple, having only 4 split-points. The main characteristics of this model is the attack, the sustain or decay, and the release as outlined in

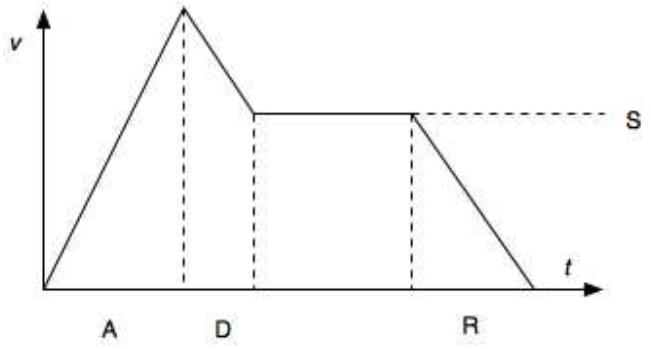

Figure 11: ADSR Envelope

Figure 11. Amplitude envelopes with the named split points are called ADSR envelopes. We utilize envelope durations and slopes for specific instruments as specified by Jensen.

4.4 MIDI and PARSYNTH events

The Musical Instrument Digital Interface (MIDI) [2] is a universal format transmitting music. The transmission link is divided into 16 logical channels. These channel allow for multiple instruments to be represented. Each channel usually represents a different instrument. The basic MIDI events are Note On and Note Off events. These events are parameterized with channel, note and volume. A MIDI stream is divided into command bytes and data bytes. Command bytes have their most significant bit set, whereas data bytes do not. Each event consists of one command byte followed by zero or more data bytes. For efficiency the MIDI format allows one Note On command to be followed by data from multiple notes, as long as the event type and channel do not change. This technique is known as running status. To further compress the transmission stream, a Note On data byte with a volume of zero can be substituted for an entire Note Off event. The MIDI file format stores MIDI performances with timing information for each event.

PARSYNTH requires MIDI events to be reformatted in three ways. First, the timing information for MIDI events in MIDI files needs to be converted into frame numbers. Second, the running status is converted into individual events. Finally, each MIDI note number must be converted to its corresponding frequency. This conversion is performed by mapping the

twelve equally tempered notes in the lowest musical octave into their frequencies. Each subsequent octave doubles the frequency of the same note in the previous octave. So by mapping the MIDI note number into a equally tempered note and octave, a conversion of that MIDI note to its corresponding frequency can be performed.

4.5 Digital Audio Mixing

A digital stream has a maximum amplitude. When mixing multiple streams, a mixer usually adds the values of the streams together. However, adding high amplitude streams together can result in a value greater than the maximum. This mathematical overflow results in a performance artifact known as clipping, where the mixed stream results in a sound considerably different from the intended one.

To avoid clipping, digital mixing consoles [35] commonly use one of three techniques. The first is automatic gain control (AGC) [18]. The mixer attempts to maintain an average volume for the performance by varying the gain of the mixed stream. Another technique is to limit the volume of each stream to a fraction of the total volume. As a result, the volume of each output stream is volume of the input stream divided by the number of input streams. This prevents clipping entirely at the cost of low average volume during the performance. The third technique takes advantage of the fact that all streams of a performance are exceedingly rarely at maximum volume at the same time. Instead of dividing the volume of each input stream by the number of input streams, the mixer divides the volume of each input stream by the square root of the number of input streams [40]. In pathological cases clipping can still result. The square root technique has higher average overall volume than the second technique. However, due to the simplicity of the second technique, PARSYNTH's mixer divides the volume of each stream by the number of streams.

CHAPTER V

PARSYNTH SYSTEM ARCHITECTURE

This chapter discusses the implementation of PARSYNTH [19], a PEDALS based digital audio synthesizer. PARSYNTH is an event driven system consisting of a set of modules interconnected by links. PARSYNTH exposes adaptation decisions to the perceiver. PARSYNTH manages impreciseness.

5.1 PARSYNTH Events

As described in the previous chapter, a MIDI stream is divided into command bytes and data bytes. MIDI allows for running status and for Note On/Note Off substitution. The MIDI file format stores MIDI performances with timing information for each event.

PARSYNTH requires MIDI events to be reformatted in three ways. First, the timing information for MIDI events in MIDI files needs to be converted into frame numbers. Second, the running status is converted into individual events. Finally, each MIDI note number must be converted to its corresponding frequency.

We started with an Open Source tool, MIDIFile to text (mf2t) [32], to transform binary MIDI files into text PARSYNTH format. We changed the output format to emit absolute frame numbers instead of relative offset times. mf2t also converts running status into individual events. MIDI notes are converted to frequencies in the dispatcher.

The PARSYNTH event file format is shown in Table 2. Each event is marked with an absolute frame number. Events in the file are ordered by frame number. Frame number zero represents the starting time of the performance. The dispatcher keeps track of frame number. When the dispatcher frame number matches the frame number of the next event in the PARSYNTH event file, that event is processed. Therefore, the dispatcher frame number is used to time the insertion of events into the system.

Table 2: PARSYNTH Event File Format

Entry	Description
frame number	sequence information
channel	from 1-16
event type	0 - Note On, 1 - Note Off
note type	0-127 MIDI standard
volume	0-127 MIDI standard

5.2 PARSYNTH Links

Links in PARSYNTH are implemented with the well known message passing technique [43]. Messages are sent to and received from mailboxes. Mailboxes are implemented using shared process memory. The data structure for mailboxes is a queue. The queue is statically allocated. Threads using the queue will block upon a read from an empty queue or a write to a full queue. Admin links and data links only differ in their use, not in their implementation.

5.3 PARSYNTH Modules

Each module has an administrative link. In addition, a module can have zero or more input and/or data links. Data links are dynamically allocated and deallocated during a performance. Mailbox addresses are attached to the corresponding NOTEON and NOTEOFF events that allocate or deallocate the data link. Each module is implemented with a user level thread.

PARSYNTH modules are structured in an event loop where each iteration of the loop processes exactly one frame of data. The beginning of each module's loop consists of processing administrative events in the module's administrative link. The module processes data for the frame. Finally, modules send output to the data link and control events to the admin links of downstream modules.

Because of the blocking nature of the link, a module will block until an administrative message is received. This blocking behavior drives the thread execution model of the system. The current implementation utilizes a FIFO thread run queue. Blocking on a link removes the thread from the run queue making it unavailable for execution until released from the block. This behavior has two effects. The first is that inactive threads that have been

preforked are not executed until they have received administrative messages. The second is that active threads are guaranteed to only execute their event loop once each frame because each module will block on its empty administrative link after completing the current frame's execution.

Modules block if no administrative messages are available. Each module that generates output on a data link is required to signal the other endpoint of that link that more data is available. The NEWWORK event provides that signal. The CONTINUE event acknowledges of receipt of new work and signals the receiver to continue processing for the next frame.

5.3.1 Dispatchers

The system architecture of PARSYNTH allows for adaptation during module execution. The two elements that facilitate adaptation are the single frame event loop structure and the processing of admin link events each frame.

Adaptation is performed by sending an ADAPT message to the module over its admin link. The ADAPT message consists of the ADAPT tag along with the parameters for adaptation. The adaptation parameters are module specific. Upon receipt, the module performs the requested adaptation before processing/generating the next frame. As such the minimum adaptation latency is a single frame.

PARSYNTH's adaptation manager is called the dispatcher. The dispatcher performs several functions:

- The dispatcher creates all of the module threads in the PARSYNTH system prior to beginning the performance.
- The dispatcher accepts the externally generated input events for the performance. In the current implementation, these external events from from the PARSYNTH event file. The dispatcher then sends the appropriate START and STOP events to the modules over their admin links. START events are marked with the dispatcher frame number.

- The dispatcher converts MIDI note values into their corresponding frequencies.
- The dispatcher monitors the frame numbers of the output of the PARSYNTH consumer module. It also logs the frame execution duration of individual frames by PARSYNTH. This logging data is used to generate some of the results in the next chapter.
- The dispatcher executes the PEDALS adaptation algorithm. Adaptations are sent as needed to keep the performance within soft real-time frame execution.
- The dispatcher also drives the dynamic linkages between modules by passing modules the output link of their target. This is done for each NOTEON event. As a result the dispatcher can dynamically reconfigure the application during execution.

5.3.2 Generators

The adaptive generator module produces digital audio samples. Based on parameters received from the dispatcher, the generator utilizes a ringing infinite Q filter to generate a bank of sine wave partials. The generator accesses the SHARC database [39] to obtain harmonic frequency, amplitude, and phase information for each of the sine wave partials in the bank. The partials are then combined via additive synthesis to produce the tone. The tone is then scaled to the given volume for the note. The result is a CD quality audio stream of 44100 16 bit signed integer samples per second. Due to the frame based configuration of PARSYNTH, each generator must produce 210 samples per frame. Generators accept ADAPT events which change the number of partials generated per frame.

PARSYNTH generators operate asynchronously generating a stream of digital audio samples. Each of the 210 samples of the frame have the same frame number assigned. The starting frame time comes from the START event sent by the dispatcher. The generator subsequently updates the frame number of the samples as appropriate.

5.3.3 Filters

Mixers are one type of filter implemented in PARSYNTH. In the previous chapter, we discussed three techniques to avoid clipping when mixing digital streams. They were AGC,

divide stream volume by the square root of the number of streams, and averaging. To avoid clipping, the PARSYNTH mixer averages by dividing the volume of each stream by the number of streams. Mixers are multi-input data channel modules. These input data channel are dynamically attached and detached during the performance. The mixer produces a single output data channel which is directed to the consumer module.

Synchronizers are the other type of filter. As discussed, generators asynchronously produce digital audio samples marked with frame numbers. A newly started generator may have frame numbers that are out of sync with other running generators. The synchronizer module accepts data from all generators and synchronizes the data so that all of the frame numbers match. The synchronizer gets NOTEON events from the generators as those generators begin to produce a digital audio stream. This NOTEON event contains the address of the new incoming data channel. The synchronizer then proceeds to deliver a NOTEON event, and a new output data channel address, to the target module that to which it is connected. Typically this is the mixer. Therefore, the synchronizer is a multiple input, multiple output channel module.

5.3.4 Consumers

The consumer module accepts digital audio samples from an input data channel and writes those samples to the Unix file as directed during setup. The consumer module also informs the dispatcher of the frame numbers of the digital audio samples it receives.

5.4 Summary

This chapter discussed the system architecture of the PARSYNTH digital audio synthesizer. PARSYNTH implements the PEDALS model. PARSYNTH events are mapped from MIDI events, augmented in particular by frame numbers. These events are transferred over dynamically configurable links. PARSYNTH modules include the one dispatcher, generators, filters, and one consumer. The dispatcher handles administrative tasks. The generators generate CD quality digital audio using additive synthesis. Generators can be adapted to generate a different number of partials each frame. Filters synchronize and mix the digital audio streams. The mixed stream is output by the consumer module.

CHAPTER VI

EXPERIMENTS AND RESULTS

In order to evaluate the effectiveness of our system, we must benchmark its critical attributes. In this chapter, we discuss what performance attributes are important, experiments that measure those attributes, the results of the experiments, and their meaning.

6.1 Testbed

For all experiments, we removed variations in computational resources in three ways. First, we used the Linux real-time FIFO scheduler. The scheduler suspended all non real-time processes while the real-time PARSYNTH process executed. Second, we recorded timing information to a memory buffer during the performance. After the performance, the buffer was then written to a file. Finally, for timing experiments the performance was not written to a file.

PARSYNTH has been instrumented with several environment variables in order to obtain benchmarks that are outside the scope of a normal performance. For example, the PARTIALS variable allows experimenter control over the number of partials generated.

Each frame produces 210 digital audio samples at a rate of 210 frames per second to maintain real time performance. Therefore, PARSYNTH must produce 210 frames per second for zero delay. In other words, one frame must be produced every 4761.9 μS.

6.2 Experiment 1: Real-Time Performance Microbenchmarks

The first critical attribute is real-time performance. The core problem we are solving is the maintenance of a real-time performance under varying input loads. We measured the cost of system overhead and cost of sine wave generation. We have designed experiments for each component to show that neither of these costs dominates the available computational load.

6.2.1 Generator Load

The first experiment examines the system overhead. The system overhead costs consist of the idle state operation of the dispatcher, synchronizer, mixer, and consumer. These modules are required in PARSYNTH for all productions. In order to measure their inherent cost, we have designed an experiment that measure the computation cost of processing frames of zero samples generated by a single generator. This represents the PARSYNTH system in its idle state. We measured the time it takes to generate a frame's worth of samples. We then compared that generation time to the actual frame length. We determined the percentage of system overhead time in the generation of a frame.

6.2.1.1 Testbed

Testbed 1 is comprised of a 600 Mhz AMD Duron processor with 128 MB of RAM. The machine runs Slackware Linux 9.1

6.2.1.2 Event File

The event file has no events for the first 500 frames. A new note is added to the performance every 100 frames. Each note activates another generator. The total length of the performance was 1000 frames.

6.2.1.3 Methodology

This benchmark measures both the idle and generator loads on PARSYNTH.

For this test the generators computed no partials in order to focus on the overhead of the generator.

6.2.1.4 Results and Conclusions

The results of the experiment are found in Table 3. The data shows the average frame execution time increases linearly with the number of generators. The average cost for an additional generator is 29.5 μS per generator. The overhead cost of one generator is 0.621 percent of the available frame time of 4761.9 μS.

Table 3: Generator Execution Time

Number of Generators	Average Frame Execution Time (μS)	Delta of New Generator (μS)
0	75.55711	N/A
1	80.33000	4.773
2	110.21000	29.88
3	139.07000	28.86
4	168.00000	28.93
5	198.46464	30.46

The idle state, represented by the 0 generator entry in the table, is somewhat misleading. As discussed in Chapter 3, even when no notes are present, one generator must be active. This silent generator maintains timing for performance. The first generator is different than the idle state is actually running a only in that computes volume for the actual note it generates.

6.2.2 Sine Wave Partial Generation Cost

This microbenchmark measures sine wave partial computation cost on PARSYNTH. Sine waves are used in the additive synthesis process to produce digital audio streams. These costs are used to populate the cost field in the adaptation table for the adaptation algorithm.

6.2.2.1 Testbed

Testbed 1 comprises of a 600 Mhz AMD Duron processor with 128 MB of RAM. The machine runs Slackware Linux 9.1

6.2.2.2 Event File

The event file has no events for the first 100 frames. A single note is added to the performance starting at frame time 100. The total length of the performance was 200 frames.

6.2.2.3 Methodology

This microbenchmark measures the execution time for sine wave generation. We ran PARSYNTH 21 times. Each run computed a different number of partials. The number of partials generated was controlled by the environment variable PARTIALS. We recorded

Table 4: Sine Wave Partial Execution Time

Number of Partials	Average Frame Execution Time (μS)	Average Time Per Partial(μS)
00001	11.78633	11.78633
00002	20.09946	10.04973
00003	28.54391	9.51463
00004	37.07926	9.26981
00005	42.60451	8.52090
00006	51.62471	8.60411
00007	59.64492	8.52070
00008	77.00855	9.62606
00009	83.90754	9.32306
00010	89.09946	8.90994
00020	172.43279	8.62163
00030	252.64492	8.42149
00040	330.92774	8.27319
00050	408.89744	8.17794
00100	808.74593	8.08745
00200	1611.17017	8.05585
00300	2406.98835	8.02329
00400	3201.03885	8.00259
00500	4002.91764	8.00583
01000	7984.41259	7.98441
02000	15971.34188	7.98567

the average frame execution time for the 100 frames during which the generator was computing the sine wave partials.

6.2.2.4 Results and Conclusions

As shown in Table 4, we measured a range of a number of partials from 1 to 2000. The average frame computation time for the entire range was 8.75069 μS with a range from 7.98441 μS to 11.78633 μS. We hypothesize that the decreasing trend in averages is due to decreased impact of perturbation of the first few frames.

We can use the results from this experiment to populate the adaptation table with the cost of sine wave partial computation.

6.3 Adaptation

This experiment compares PARSYNTH's adaptation algorithm against best effort generation. This experiment examines how PARSYNTH's adaptation algorithm functions in the face of constrained resources.

6.3.1 Testbed

Testbed 2 is a Dell desktop with a 2.8 Ghz Intel Pentium 4 processor and 512 MB of RAM. The machine runs RedHat Linux 9.

6.3.2 Event File

The event file was a synthetic file. A note is added to the performance every 200 frames starting at frame time zero. The total length of the performance was 1500 frames.

6.3.3 Adaptation Table

In order to create sufficient load for testing in overload conditions, the best effort generation run produced notes with a fixed cost of 200 partials each. For the adaptation run Adaptation Table 5 was used. This adaptation allocated each note equal preference. A range of adaptation options were given for each note. Due to the adaptation algorithm using the average partial computation time to compute cost, the cost for each option is specified by the number of partials generated for that option. The preference values matched the cost for this experiment. The 200 partial option that heads the table represents the maximum number of partials that could be generated for the given note. So it represents a complete generation option. The available resource cost for this testbed is 860 partials.

6.3.4 Methodology

For this benchmark we recorded the average execution time for each note.

6.3.5 Results and Conclusions

As shown in Figure 12, the average execution time of the adapted performance matches the time of the best effort performance as long as resources are available. Starting with note 5, the resource cost of best effort exceeds the available resources. As a consequence,

Table 5: Adaptation Table

Preference	Cost
200	200
150	150
75	75
30	30
15	15
5	5
1	1

the average execution time of 5506 μS exceeds the length of the frame. On the other hand, the adaptation algorithm adapts the generators for the adapted performance. These adaptations constrain the execution time of the generators to the frame.

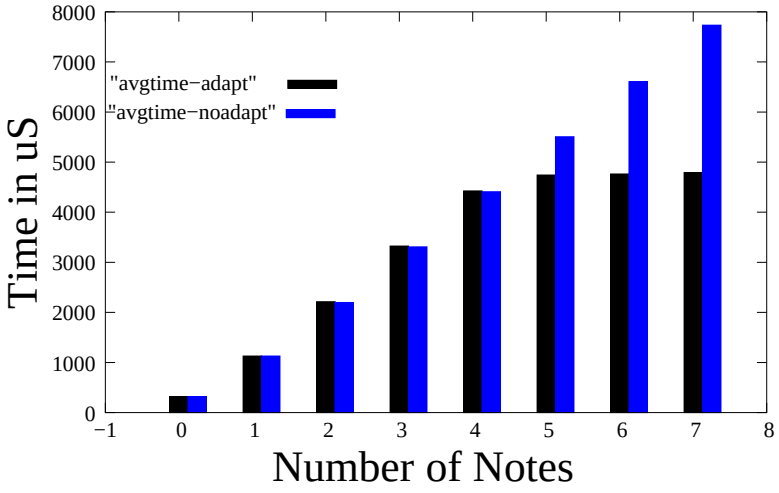

Figure 12: Adaptation vs. Best Effort: Delay

6.4 Impreciseness

To evaluate impreciseness, we set up an experiments with a pair of notes, whose cost exceeded the ARC budget. We computed the total preference of the options chosen. We then evaluated the impreciseness of the performance. In order to benchmark the selections of the preference based adaptation algorithm, we compared the impreciseness of that performance to a performance that is adapted strictly based on cost. Cost is one type of hardwired adaptation that does not use perceiver preference for adaptation decisions.

6.4.1 Testbed

Testbed 3 is a Dell laptop consisting of an Mobile Intel Pentium 4 1.7Ghz CPU with 512MB of RAM. The operating environment is Knoppix Linux 3.6 running Real-Time FIFO scheduling as root.

In order to force adaptation, the ARC for each of these experiments are artificially limited using the environment variable MAXPARTIALS. The ARC was limited to 13 sine wave partials.

6.4.2 Event File

The event file for the experiment consists of two NOTEON events. Each event is assigned to a different instrument. The total run time of the performance is 5 frames.

6.4.3 Adaptation Table

In order to compare adaptations using preference and adaptation that only use cost, we need to construct equivalent adaptation tables. For this experiment, we compared preference based adaptation with cost based adaptation. The preference based adaptation is represented by a performance with a high preference instrument and a low preference instrument as shown in Table 6. For the cost based algorithm, we linearly map the cost in sine wave partials to preference by scaling the cost by a factor of 100. The cost based adaptation table is shown in Table 7.

6.4.4 Methodology

In this benchmark, we compute the total preference for the options chosen. However, due to the linear scaling of the preference values for the cost based adaptation, the preferences from the adaptation table shown in Table 7 have no correlation to the preferences given by the perceiver in Table 6. In order to assess the total preference from the perceiver perspective, the preference values from Table 6 are mapped onto the corresponding generation options in Table 7.

We compute the impreciseness of the preference of the performance by dividing the difference in the total preference of a complete performance and the total preference of the

Table 6: Preference Based Adaptation Table

Generation Option	Preference	Cost
A1	900	10
A2	850	9
A3	800	7
A4	750	4
A5	700	3
A6	650	1
B1	300	8
B2	250	6
B3	200	4
B4	150	3
B5	100	2
B6	050	1

Table 7: Cost Based Adaptation Table

Generation Option	Preference	Cost
A1	1000	10
A2	900	9
A3	700	7
A4	400	4
A5	300	3
A6	100	1
B1	800	8
B2	600	6
B3	400	4
B4	300	3
B5	200	2
B6	100	1

actual performance by the total preference of a complete performance. Then we compute the total cost for the generation options set chosen. We compute the impreciseness of the cost of the performance by dividing the difference of the cost of a complete performance and the cost of the actual performance by the cost of the complete performance. Finally, we compute the impreciseness of each of the two instruments used in the performance using the same metric as the total performance

6.4.5 Results and Conclusions

The preference based adaptations chose options A1 and B5. The cost based adaptations chose options A5 and B1. Figure 13 shows the total preference for the performance and for each instrument. The preference and cost based options are compared to the complete performance, which is options A1 and B1.

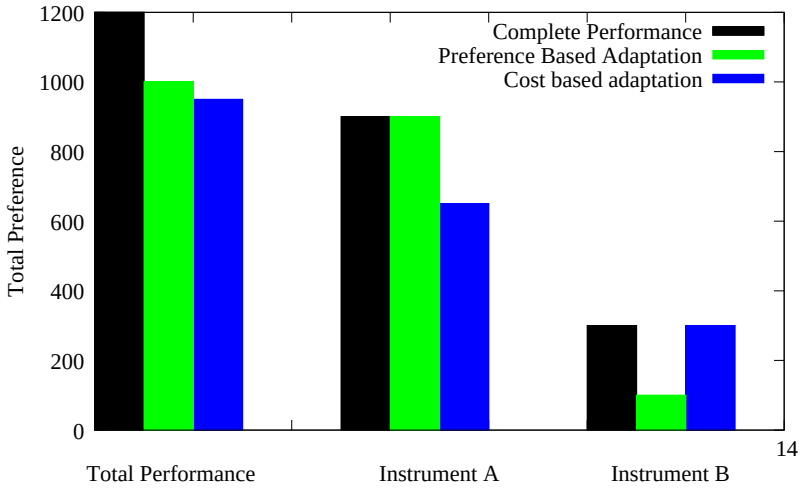

Figure 13: Total Preference of Preference vs. Cost based adaptation

While the total preference for both preference and cost based adaptation show similar values, the allocation of preference is illustrated in the per instrument elements. Note that the high preference instrument retains a high preference value, while the preference value of the low preference instrument is dropped.

The total cost, measured in partials generated, for each of the selected options is shown in Figure 14. The total cost of the selected options are similar for both adaptation strategies. However, the instrument allocation shows that much more of the resource budget is allocated

to the higher preference instrument. Note that the cost of the ideal performance (18 partials) exceeds the ARC for the frame, which is 13 partials.

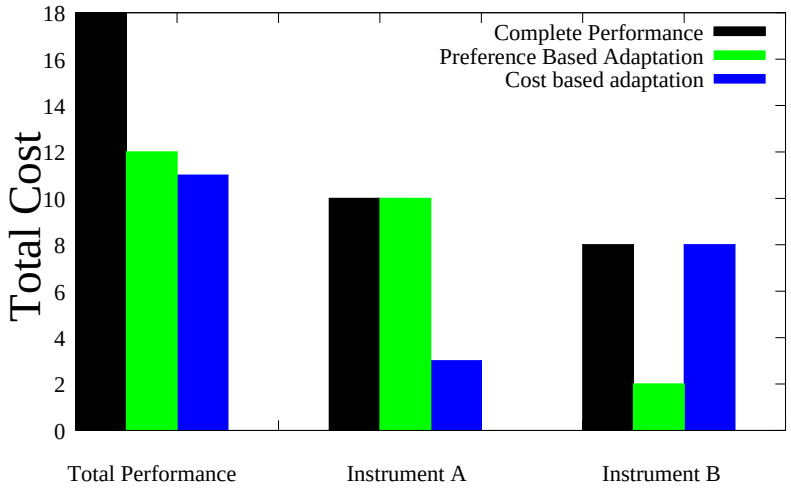

Figure 14: Total Cost of Preference vs. Cost based adaptation

The allocation of impreciseness of preference for the total performance and both instruments is shown in Figure 15. As outlined in the methodology section we compute impreciseness against a complete performance, which in this case would be options A1 and B1. Because the impreciseness of a complete performance is zero in all cases, the complete performance is omitted from this figure.

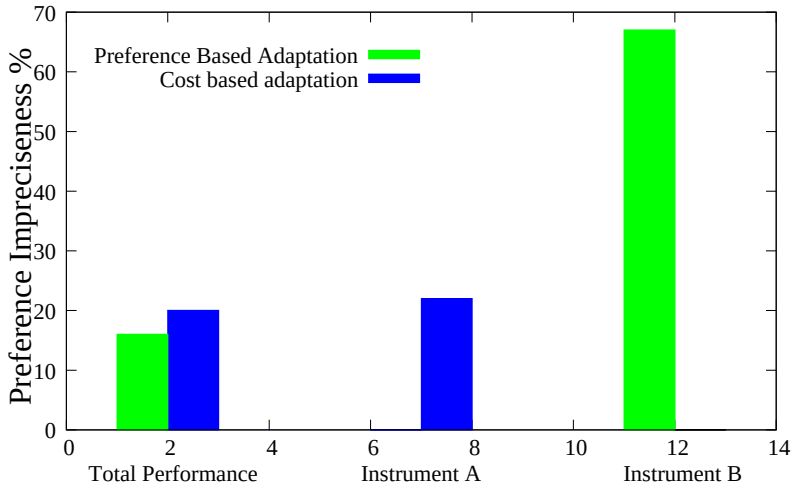

Figure 15: Impreciseness of Preference for Preference vs. Cost based adaptation

The results show that the impreciseness of the total performance of both adaptation algorithms are similar. The difference is shown in the allocation of impreciseness to each

instrument. Instrument A, the high preference instrument has a complete performance for that instrument when preference based adaptation is used. On the other hand, Instrument B has a high imprecision of 67 percent. Cost based adaptation allocates imprecision to the high preference instrument. The low preference instrument is adapted to have no imprecision.

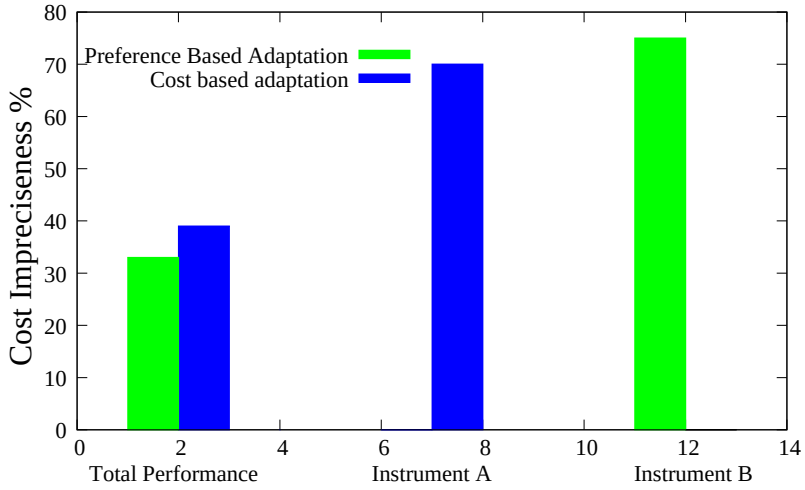

Figure 16: Impreciseness of Cost for Preference vs. Cost based adaptation

The imprecision of the cost, shown in Figure 16, further illustrates the differences in the two strategies when allocating resources. Imprecision of cost is again computed against the cost of the complete performance options of A1 and B1. The higher relative impreciseness of the total cost of both strategies reflects the unachievable selection of the complete performance. Cost constraints limit the cost of available options for both strategies. The instrument entries shows how resources are spent. In the case of preference based adaptation, the preferred instrument again has no imprecision. In contrast, the low preference instrument has high imprecision. Cost based adaptation allocates the majority of the available resources to the lower preference instrument. The preferred instrument has high impreciseness for cost.

6.5 Experiment with Bach:Toccata & Fugue in D-Minor

In this experiment, we examine the effects of perceiver input on delay. We utilize four test cases over a passage from *Bach: Toccata & Fugue in D-Minor*. We compared the following

four cases:

- A complete performance with unlimited resources.
- A complete performance with limited resources.
- An imprecise performance with minimum computation.
- An adapted performance with perceiver input with all channels given equal preference.

We measured the delay of generating the frames for the passage. We also observed the qualitative impact of delay and CPU utilization in each of the four cases.

6.5.1 Testbed

Testbed 3 is a Dell laptop consisting of an Mobile Intel Pentium 4 1.7Ghz CPU with 512MB of RAM. The operating environment is Knoppix Linux 3.6 running Real-Time FIFO scheduling as root.

6.5.2 Event File

We tested this over the opening passage of *Bach: Toccata and Fugue in D-minor*. The passage consisted of 260 note events.

6.5.3 Adaptation Table

The adaptation tables for the complete performances were simulated by setting the PS-MAXPARTIALS variable to 10000 partials. For every note represented in the SHARC database, this number of partials generates a complete note for each event.

The adaptation table for the minimal performance had one entry for each channel. The number of partials for each entry was set to 1.

The adaptation table for the perceiver input had multiple entries for each channel. In addition the corresponding entries for each channel had identical values for preference and cost.

Table 8: Delay for *Bach: Toccata and Fugue in D-minor*

Type	Average Delay (in μ)	Percentage Late Frames	Longest Frame
Complete Unlimited	0	0	0
Complete Limited	9.876ms	15.90%	6700
Minimal	0	0	2200
Adaptation	0	0	4800

6.5.4 Methodology

The first case of a complete performance with unlimited resources is a theoretical absolute. We did not perform that actual experiment. Instead we used it as a baseline for comparison.

The second case of a complete performance with limited resources was run with the PSMAXPARTIALS environment variable set to 10000 partials.

The other two cases used their corresponding adaptation tables. Please note that the minimal case did not perform any adaptation due to the single entry for each channel.

As described earlier, in order to maintain soft real-time performance, a frame must be computed within 4761 μS. Frames that take more than 4761 μS are declared as delayed frames. We measured the percentage of delayed frames in each of the four cases.

6.5.5 Results and Conclusions

Table 8 shows that the most delay occurs with the complete performance with limited resources. This result is consistent with insufficient resources to generate the complete performance in soft real-time. There is no delay with the minimal performance. Note that the maximum computation time for a frame in the minimal performance is 2200 μS. This shows that the CPU was underutilized because each frame was computed in less than half the available time. The adapted performance displayed neither delay nor CPU underutilization.

Listening to the complete performance on limited resources, the perceiver notices gaps and noise artifacts. Listening to the minimal performance, the perceiver hears a tinny timbre, similar to the sounds from a child's toy. This timbre is representative of single sine wave tones. In constrast, the adapted performance had no delay-based artifacts. The earlier segment of the passage has fewer notes. As a result, it is produced with rich tonality similar

to the complete performance with unlimited resources. The last segment of the passage has more notes that need to be simultaneously processed. In this segment, the perceiver observes thinner timbres. These thinner timbres are due to the adaptation of the notes in the segment. However, due to adaptation, the greater CPU utilization results in timbres that are never as tinny as those of the minimal performance.

6.6 Summary

In order to evaluate the effectiveness of our system, we benchmarked critical attributes including system overhead and sine wave partial generation. Results show that even with a reasonably low powered system, adequate computation capacity is available to produce digital audio performances in soft real-time with PARSYNTH.

We next investigate the PEDALS adaptation algorithm. Results show that delay is curtailed when the adaptation algorithm is utilized. Results also indicate that in underloaded situations, the adaptation algorithm matches resources allocation with best effort strategies.

Next, we compare the allocation of impreciseness using preference based adaptation as opposed to hardwired cost based adaptation. Results show that both adaptation strategies result in imprecise performances. However, the preference-based adaptation strategy produces performances where impreciseness of both preference and cost is steered away from the preferred instruments and towards low preference instrument. Cost based adaptation shows comparable impreciseness values for the total performance. However, preferred instruments are not protected from being adapted to more imprecise generation options.

Finally, we perform an experiment that examines the qualitative effects of perceiver input upon delay using a passage from *Bach: Toccata and Fugue in D-minor*. Results show that a complete performance generated with limited resources results in significant noise artifacts, due to delay, in the resulting performance. In comparison, both the minimal performance and the adapted performance using perceiver input produce performances with no delayed frames. However, with the minimal performance, the CPU is not fully utilized. As a result, the adapted performance with perceiver input produces a performance with a richer timbre than the minimal performance.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

This dissertation explores the area of Generative Multimedia (GM) Applications. GM Applications share three basic characteristics: specifying performances concisely, exploiting ever increasing endpoint computational capacity, and adapting the performance structure. The constraints on a GM application are: soft real-time requirements, lack of a priori knowledge of incoming input events, variable computational requirements, and confined resource constraints.

General solutions to the problem of resource constraint are best effort, minimal effort, and voice stealing. Their respective weaknesses are excessive delay, underutilized resources, and ignored perceiver preference. Each utilizes hardwired adaptation for resource allocation. Our approach provides flexibility to the perceiver by allowing the perceiver to specify preferences. The system makes adaptation decisions based on the perceiver's input.

The metrics we use to measure the efficacy of adaptation decisions are impreciseness and delay. Perceiver directed impreciseness minimizes delay via imprecise computation of media elements of a performance. The imprecision of these elements is determined by the preference of the perceiver.

The PEDALS model exposes adaptation decisions to the perceiver. PEDALS allows the perceiver to choose adaptations that he or she prefers. Applications in the PEDALS model dynamically adapt the performance to match the computational constraints of the environment. The adaptations chosen allocate the imprecision of the media elements based on the preferences of the perceiver.

We contend that Generative Multimedia requires adaptive solutions to generate high quality performances. Perceiver preference input can be used to control the impreciseness of GM performances. Towards this end, we propose the PEDALS model, that exposes the perceiver to adaptation options with variable impreciseness. We also define metrics that

quantify the impreciseness of an adapted Generative Multimedia performance.

The components of the PEDALS model are its architecture and its adaptation algorithm. The architecture is based on events, links, and modules. This lends itself to distributed applications. The adaptation problem is an approximation to the well known Knapsack problem. Since this is an NP hard problem, PEDALS provides an approximation algorithm.

PARSYNTH is a digital audio synthesizer that implements the PEDALS model. It generates CD quality audio samples using an additive synthesis algorithm. It also provides instrumentation facilitating testing and measurement of delay and imprecision.

Finally, we ran a series of experiments to evaluate the performance of PARSYNTH. We determined that PARSYNTH has sufficient capacity to generate digital audio in real-time. We measured the computational overhead of the system and the computational cost of the additive synthesis algorithm.

7.1 Future Work

Currently, PARSYNTH relies on a static cost function using valuesare derived from empirical testing. It would be desirable to have PARSYNTH tune itself. PARSYNTH could run a short diagnostic to determine the current cost of its generative algorithms.

Self tuning is the first step to implementing PARSYNTH in a distributed environment. The other major issue is handling synchronization. When generators run asynchronously on heterogeneous hardware, the task of synchronization and mixing of digital audio streams becomes more difficult.

Testing up to this point has used synthetic streams of input events. Instead of music, event files contained arbitrary sequences of evenly spaced events. A more meaningful test of the performance of the algorithm would be music played by musicians in real-time. With more meaningful tests, we could determine the actual effectiveness of the adaptation algorithm.

Generative Multimedia spans several application domains. Our experiments thus far have focused on the specific domain of digital audio. We believe that our model and our results generalize to other GM domains. We plan further research in this area beginning

with studying the interaction between animation and digital audio.

Finally, we continue to look for meaningful ways to gather preference input from the perceiver. We have considered a variety of techniques from console style sliders, to questionnaires, to observation. We hypothesize that future systems will use a variety of input modalities. In some domains, such as digital audio, the perceiver is normally passive. Also, researchers have found it difficult to get perceivers to rate distributed multimedia. In other domains, there are strong links between application semantics and current behavior and preferences. For example in a video game, players will care about high display fidelity when they are passively interacting with a scene. However, game speed is critical in action scenes that are filled with many visual elements. In that case a different set of preferences apply.

Even if we succeed in gathering meaningful preference data in isolation, that information cannot be naively combined. Preference data exists in a context and that context is dynamic in a GM performance. One consequence of this dynamic context is that perceivers tend to focus on one specific media domain when presented with a multimedia stimulus. Also perceivers shift focus based on their needs. This shift is evident in the game example . Therefore, the implementation of the adaptation algorithm needs to handle changes in preference data during the performance. Therefore another area of research is to dynamically capture these shifts in focus and use them to steer impreciseness via adaptation.

The domain of adaptive systems holds great promise for the future. As systems become more complex, and perceiver expectations increase, the need for responsive systems, models, and frameworks increases. Adaptation based on perceived driven impreciseness offers rich possibilities.

REFERENCES

[1] Apteker, R. T., Fisher, J. A., Kisimov, V. S., and Neishlos, H., "Video acceptability and frame rate," *IEEE MultiMedia*, vol. 2, no. 3, pp. 32–40, 1995.

[2] Association., M. M., "The complete midi 1.0 detailed specification," 1996.

[3] Aurrecoechea, C., Campbell, A. T., and Hauw, L., "A survey of qos architectures," *Multimedia Systems*, vol. 6, no. 3, pp. 138–151, 1998.

[4] Bihari, T. E. and Schwan, K., "Dynamic adaptation of real-time software," *ACM Trans. Comput. Syst.*, vol. 9, no. 2, pp. 143–174, 1991.

[5] Boring, R. L., West, R. L., and Moore, S., "Helping users determine video quality of service settings," in *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, (New York, NY, USA), pp. 598–599, ACM Press, 2002.

[6] Boulanger, R., *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming.* Cambridge, MA, USA: MIT Press, 2000.

[7] Brandt, S. A., "Performance analysis of dynamic soft real-time systems," in *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pp. 379–386, 2001.

[8] Campbell, A., Coulson, G., and Hutchison, D., "A quality of service architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 2, pp. 6–27, 1994.

[9] Chamberlin, H., *Musical Applications of Microprocessors.* Rochelle Park, NJ, USA: Hayden Books, 1985.

[10] Cheng, S.-T., Chen, C.-M., and Chen, I.-R., "Dynamic quota-based admission control with sub-rating in multimedia servers.," *Multimedia Syst.*, vol. 8, no. 2, pp. 83–91, 2000.

[11] Chowning, J., "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the Audio Engineering Society*, vol. 21, no. 7, pp. 526–34, 1973.

[12] Claypool, M. and Tanner, J., "The effects of jitter on the peceptual quality of video," in *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, (New York, NY, USA), pp. 115–118, ACM Press, 1999.

[13] Coulter, D., *Digital Audio Processing.* Lawrence, KS, USA: CMP Media, 2000.

[14] Cranley, N., Murphy, L., and Perry, P., "User-perceived quality-aware adaptive delivery of mpeg-4 content," in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, (New York, NY, USA), pp. 42–49, ACM Press, 2003.

[15] Ghinea, G. and Thomas, J. P., "Qos impact on user perception and understanding of multimedia video clips," in *MULTIMEDIA '98: Proceedings of the sixth ACM international conference on Multimedia*, (New York, NY, USA), pp. 49–54, ACM Press, 1998.

[16] Gopalakrishna, G. and Parulkar, G., "Efficient quality of service in multimedia computer operating systems," Tech. Rep. WUCS-TM-94-04, Department of Computer Science, Washington University, Aug. 1994.

[17] Hoffman, K. L., "Combinatorial optimization: Current successes and directions for the future."

[18] Horowitz, P. and Hill, W., *The Art of Electronics*. New York, NY, USA: Cambridge University Press, 1989.

[19] Jeff, B. and Schwan, K., "Parsynth: A case study on implementing a real-time digital audio synthesizer," in *Proceedings of 4th International Workshop on Parallel and Distributed Real-Time Systems. IEEE*, 1996.

[20] Jensen, K., *Timbre Models of Musical Sounds*. PhD thesis, Dept. of Computer Science, University of Copenhagen, 1999.

[21] Kephart, J. O. and Chess, D. M., "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[22] Koliver, C., Nahrstedt, K., Farines, J.-M., Fraga, J. D. S., and Sandri, S. A., "Specification, mapping and control for qos adaptation," *Real-Time Syst.*, vol. 23, no. 1-2, pp. 143–174, 2002.

[23] Kumar, R., Wolenetz, M., Agarwalla, B., Shin, J., Hutto, P., Paul, A., and Ramachandran, U., "Dfuse: a framework for distributed data fusion," in *Proceedings of the first international conference on Embedded networked sensor systems*, pp. 114–125, ACM Press, 2003.

[24] Leung, W. H., Tseng, B. L., Shae, Z.-Y., Hendriks, F., and Chen, T., "Realistic video avatar," in *Proceedings of IEEE Intl. Conf. on Multimedia and Expo*, 2000.

[25] Lin, Y.-J., Guo, K., and Paul, S., "Sync-ms: Synchronized messaging service for real-time multi-player distributed games," in *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, pp. 155–164, IEEE Computer Society, 2002.

[26] Liu, J., Lin, K., Bettati, R., Hull, D., and Yu, A., "Use of imprecise computation to enhance dependability of real-time systems," 1994.

[27] Loyall, J., Schantz, R., Zinky, J., and Bakken, D., "Specifying and measuring quality of service in distributed object systems," in *Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98)*, 1998.

[28] Nahrstedt, K. and Smith, J. M., "The QoS Broker," *IEEE MultiMedia*, vol. 2, pp. 53–67, Spring 1995.

[29] Neely, S. T. and Allen, J. B., "Predicting the intensity jnd from the loudness of tones and noise," *Psychophysical and Physiological Advances in Hearing*, pp. 458–464, 1998.

[30] Östergren, M., "Sound pryer: Adding value to traffic encounters with streaming audio.," in *ICEC*, pp. 541–552, 2004.

[31] Pâris, J.-F., "An interactive protocol for video-on-demand," in *IEEE International Performance, Computing and Communication Systems*, April 2001.

[32] Peterson, M., "mf2t: Midi file to text converter," tech. rep., World Wide Web, `http://www.sm5sxl.net/~mats/src/unix/music/mf2t`, 2005.

[33] Phillips, D., "Csound for linux," *Linux J.*, vol. 1999, no. 58es, p. 2, 1999.

[34] Poellabauer, C., *Q-Fabric: System Support for Continuous Online Quality Management*. PhD thesis, Atlanta, GA, USA, 2004.

[35] Polhmann, K. C., *Principles of Digital Audio*. New York, NY, USA: McGraw-Hill, 2000.

[36] Rencuzogullari, U. and Dwardadas, S., "Dynamic adaptation to available resources for parallel computing in an autonomous network of workstations," in *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, (New York, NY, USA), pp. 72–81, ACM Press, 2001.

[37] Rosu, D., Schwan, K., and Yalamanchili, S., "FARA - a framework for adaptive resource allocation in complex real-time systems," in *IEEE Real Time Technology and Applications Symposium*, pp. 79–84, 1998.

[38] Sahni, S., *Data Structures, Algorithms, and Applications in C++*. McGraw-Hill Pub. Co., 1999.

[39] Sandell, G., "The sharc timbre database," compressed Tar File (832K), World Wide Web, `http://people.cs.uchicago.edu/~odonnell/Scholar/Data/sharc.tar.Z`, 2005.

[40] Self, D., "Noise and headroom in mixers," tech. rep., World Wide Web, `http://www.dself.dsl.pipex.com/ampins/mixer/noisehd.htm`, 2005.

[41] Serif, T., Gulliver, S. R., and Ghinea, G., "Infotainment across access devices: the perceptual impact of multimedia qos," in *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1580–1585, ACM Press, 2004.

[42] Serra., X., *Musical Signal Processing*, ch. Musical sound modeling with sinusoids plus noise., pp. 91–122. Swets and Zeitlinger, 1997.

[43] Silberschatz, A. and Galvin, P., *Operating Systems Concepts*. Addison-Wesley, 4th ed., 1994.

[44] SMITH, J., "Physical modeling synthesis update," 1996.

[45] (SPEC), T. S. P. E. C., "Specmark benchmarks," tech. rep., World Wide Web, `http://www.spec.org`, 2005.

[46] Steinmetz, R., "Human Perception of Jitter and Media Synchronization," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 61–72, Jan. 1996.

[47] Vanegas, R., Zinky, J., Loyall, J., Schantz, R., and Bakken, D., "Quo's runtime support for quality of service in distributed objects," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, 1998.

[48] Weinberg, G., "The aesthetics, history, and future challenges of interconnected music networks," in *Proceedings of the 2002 Computer Music Conference*, 2002.

[49] White, S. R., Hanson, J. E., Whalley, I., Chess, D. M., and Kephart, J. O., "An architectural approach to autonomic computing.," in *ICAC*, pp. 2–9, 2004.

[50] Woods, E., Billinghurst, M., Looser, J., Aldridge, G., Brown, D., Garrie, B., and Nelles, C., "Augmenting the science centre and museum experience," in *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and Southe East Asia*, (New York, NY, USA), pp. 230–236, ACM Press, 2004.

[51] Zilberstein, S., *Operational rationality through compilation of anytime algorithms.* PhD thesis, Berkeley, CA, USA, 1993.

[52] Zinky, J., Bakken, D., and Schantz, R., "Overview of quality of service for distributed objects," in *Proceedings of the Fifth IEEE Dual Use Conference*, 1995.

[53] Zinky, J. A., Bakken, D. E., and Schantz, R. E., "Architectural support for quality of service for CORBA objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.

VITA

Dr. Byron A. Jeff received his PhD., M.S. and B.S. in Computer Science at Georgia Institute of Technology in 2005, 1989, and 1987 respectively. Jeff was named a UNCF Fellow in 1999, while pursuing his doctoral research on perceiver directed adaptive Generative Multimedia systems.

Dr. Jeff is an Assistant Professor in the Information Technology Department at Clayton State University. Dr. Jeff was the director of hardware development at Spellcaster Telecommunications based in Toronto, Canada. Dr. Jeff started his teaching career at Clark Atlanta University as an Assistant Professor in the Computer and Information Sciences Department. Dr. Jeff has compiled a textbook on Unix/Linux Usage and System Administration.