murmurized

by

Raffi Krikorian

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by . . . . . .
Judith Donath
Associate Professor of Media Arts and Sciences, MIT Media Laboratory
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

murmurized

by

Raffi Krikorian

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Traditional visualizations present quantitative data in a static format suitable for a person to gather exact data from it. Qualitative visualizations, on the other hand, try to evoke a "feeling" in the viewer instead of providing a precise interface. *murmurized* involves a trio of projects: *AudioWho* a system for monitoring the presence of people in virtual communities, an agent-based party sound modeler named *SimParty*, and the *Nonsense Machine* which is a system for the fully artificial generation of crowd sounds. These projects are all are studies on how to properly create a qualitative auralization which therefore inherently has the ability to be experienced in the background while the listeners focus on another task, but simultaneously can facilitate an emotional feeling in the listener about the social group that it is rendering.

Thesis Supervisor: Judith Donath
Title: Associate Professor of Media Arts and Sciences, MIT Media Laboratory

Acknowledgments

I would like to thank:

The members of the Sociable Media Group: danah boyd, Kelly Dobson, Andrew Fiore, Karrie Karahalios, Hyun-Yuel Lee, Derek Tang, and Fernanda Viégas. You all have made me feel welcome and have helped me grow in many ways. I would also like to thank my thesis advisor and the head of the group, Judith Donath, for accepting me into and allowing me to be amongst these people.

My friends from my old home: Jonathan and Rosanna Blandford, Neil Chungfat, Brian and Delphine Dean, Teresa DiGenova, Vincent Fish, Jessica Fox, Ethan O'Connor, Julie Park, Andrew and Wendy Russell, Jacob Schwartz, Qian Wang, and Ben Yoder. You have always been there and have been supporitive of me throughout all my crazy ideas.

Nelson Minar, a great friend and mentor – thank you for all you have taught me.

Shannon Cheng, my best friend. You are the best Best Turtle a person can ever hope to have. Thank you for everything.

My parents, my sister, and Winston for encouraging me and helping me be who I am.

And lastly, Kelly Dobson for challenging me to see things differently than I have ever seen them before. Thank you for your love and your support.

Contents

List of Figures

Chapter 1

On the Use of Sound

Graphical visualizations of social dynamics is a field being explored and developed by the Sociable Media Group of the MIT Media Laboratory. The goal of projects such as Loom 2 [DdbLR] are to visualize social data in a way that is both computational and expressive: through these visualizations, the viewer should be able to understand subtle social cues and structures about the space. Creating these visualizations involves using both cognitively and culturally meaningful patterns to create the graphical forms. Loom2 designs these representations of Usenet in order to provide an analogue to the situation of having a newcomer come and scan a room to help him or her adapt to the surroundings – via these graphics, a newcomer to a Usenet group may get subtle hints of personal interaction within the space. The danger in expressive visualizations is that viewers still read them as exact representations even though as they become more expressive, they become less exact.

murmurized creates auralizations that may provide subtle background cues to accompany another graphical or tangible display to further help a person navigate through different virtual social settings. Auralizations have two very interesting attributes: they implicitly fill a space without without forcing a direction[1] whereas visualizations exhibit the opposite feature. Also, when used properly, sound can be a fully background process; people listen to music or chatter all the time while working on other projects.

Sound, however, lacks certain quantitative qualities that graphical representations have; graphical representations have the following intrinsic qualities:

- Visual information is easier to distinguish in comparison to the information presented to our other senses
- Graphical representations are not inherently time-based
- Many different visual data "channels" can be processed simultaneously (multidimensional visual data can be accepted, ie it is "high bandwidth")

[1] A sound installer has to go through explicit work to make the sound directional via Dolby 5.1 surround sound, Audio Spotlights [Pom99], or through binaural loud speakers [Gar97]

Figure 1-1: A visual sketch of three people in a row

Try to visually illustrate the positions of people in a row, an obvious option is to present a line and to "draw" the people (figure 1-1). However, to render the same information aurally, a series of problems need to be faced. For example, try to represent the three people in figure 1-1 as notes played on the piano – one possibility is to map these persons onto a scale and to place the equivalent of the black dot at a larger interval away than the interval between the two notes assigned to the gray and white dots. This, in theory, provides distance information as the difference in frequencies between the notes maps to the distances between the people. The overriding issue, however, is the presentation of these notes: are they sounded simultaneously or sequentially?

If the notes are played simultaneously then most listeners hear a single unit or chord and not each individual note that comprise the collective sound (for example a the notes C, E, and G played simultaneously is identified as a C chord, no matter in what octaves the notes are played in. While some will be able to distinguish this, most will just notice the C chord, and most others will just note a chord was played but not be able to identify it). An obvious and simple fix may be to choose different instruments for each person. Although it may help the listener identify that there are three people in the room, it may hinder the understanding of the distances between the people as many listeners find it difficult to compare tones that are produced with different underlying timbres.

If played sequentially, then performing the comparison of the becomes hard as the auralization becomes time dependent. A comparison between the three notes becomes difficult as playing a tone for the black, gray, then white provides a good comparison for the immediately successive pairs but not for the black and white pair; unlike the visual representation that can be parsed at the viewer's own pace, the aural data either requires the viewer's intervention and interaction or the playback of the auralization a few times – it does not have the ability to stand on its own.

The limitations of aural representations becomes even more apparent as we attempt to show more than one piece of information within a single rendering. Figure 1-1 shows only a sketch containing one dimension worth of information – however when more than one piece of information is displayed (such as the complication of showing two dimensions, see figure 1-2) creating an auralization becomes even harder; the ability to separate information and to "scan" through a multidimensional representation becomes necessary, however providing both of these traits in an auralization is difficult. Unlike a graphical representation, which has two "inherent" dimensions to manipulate (horizontal and vertical), aural presentations do not have multiple inherent dimensions. It may be possible to present more than one axis of information aurally through the combinations of frequency, volume, and timbre; whereas these options are single-dimensional when used on their own: frequency moves up and down through hertz ranges, volume through decibels, and timbre differs

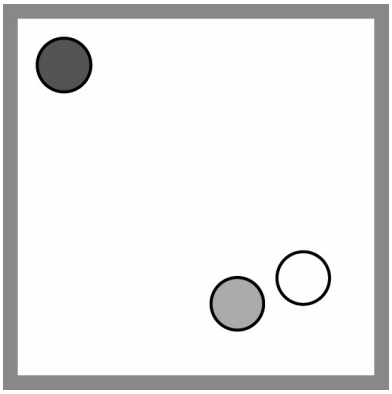

Figure 1-2: A visual sketch of the same three people from figure 1-1 in a room

as it is the quality (or fingerprint) of the tone – when used in combination, one receives independent axes of control.

Aural renderings may prove to be inappropriate for communicating quantitative information, however they may still be very well suited for qualitative information. Musicians have showed us that aural qualitative communication is possible. For example, Sergei Prokofiev's "Peter and the Wolf" has distinct characters tell their stories without anything more than clever combinations of instruments (timbre), music (frequency), and volume. Nothing exact is conveyed through this piece, but mood, feelings, and emotions are. Extending this qualitative model to apply to people on a line is possible: each circle can be assigned its own voice but notes played by the white and gray can be intertwined, almost in a conversation with a harmonic and rhythmic connection, while the black sounds on its own without any significant melodic or harmonic interplay to the first two. The use of volume may allow the model to be extended to people in a room as the black circle may be given a softer sound to make it seem further away when compared to the other two. The exact information of the persons' distances are lost, but the "feeling" that the gray and white are interacting whereas the black stands on his own may be communicated.

What exactly can be auralized is dependent on the data source, but ideally an auralization follows the same path as a qualitative visualization: if the data is "busy" the auralization should be "busy", if the data is "angry" then the auralization should be "angry", etc. All other real variables should be mapped onto variables that feel intuitive in the auralization. However, just like a visualization, the context of the chosen variables matters a lot too. For example, "anger" may be mapped to a red color in a visualization where the more angry something is, the redder it becomes; in an auralization anger could be mapped to volume where the angrier something is, the louder the auralization becomes. However, just as in a visualization where really deep red roses will not convey anger, a really loud "pop" song will not convey it either. A clever combination of variables needs to be chosen that take into account context.

The goal of *murmurized* is to create a framework that enables other people to begin to answer the fundamental question of how to create auralizations of social data – specifically, how to create auralizations using

the human voice en masse as the palette. Voices are being used instead of sound effects and instead of musical instruments because that isolates one of the variables of the auralization and helps ground it into the social realm. If doing an auralization of social data using piano notes, then that leaves the auralization unbounded and anything variable can change. Instead of tackling an entire domain of auralizations, *murmurized* only focuses on this specific one.

Voices are intriguing for use as a auralization tool as the ear is bombarded with those specific sounds throughout the day and creating an auralization that makes use of common sounds in intriguing ways is very tempting; the mind is familiar with the voice and it already has its own ways of understanding and processing the sounds. Through the clever manipulation of voices, a good auralization should be able to evoke certain feelings and recognizable memories of the listener. There is one problem which needs to be addressed with the use of sounds that is not directly addressed in this thesis: some people complain when in a room with people talking as that means they cannot concentrate – how do you properly create an auralization that uses human voices, but still retains the qualities of allowing the auralization be backgrounded by people working on different tasks.

murmurized encompasses a trio of projects: *AudioWho*, *SimParty*, and the *Nonsense Machine*. All these projects make specific inroads into the design and implementation of a system to allow for auralizations of social data. *AudioWho* attempts to study using crowd sounds to represent awareness, *SimParty* is a piece that simulates (in the most simplest sense) the sound dynamics of a party through agent based modeling, and the *Nonsense Machine* is a conceptual study of a system that is far less computationally draining as *SimParty* by taking advantage of the human ear's ability to be sensory overloaded.

Chapter 2

AudioWho

The goal of *AudioWho* is to find a simple and intuitive way of revealing the electronic activities of a community. The solution is something trivial, and the basis of much of the other work in *murmurized*. *AudioWho* creates sounds that through ebbing and flowing motion attempts to give the listener a feeling for the activity level of the population.

2.1 Inspirations

As its name reveals, this piece is directly inspired by Donath's Visual Who [Don95] work, especially drawing from her "patterns of presence" concepts.

> Activity in the electronic Lab community similarly ebbs and flows. At four a.m. there are not only the late working students, but also alumni and traveling faculty logged in from distant time-zones. By noon, well over a hundred people are usually present on media-lab, busy sending mail, checking announcements, and reading the news. Yet the usual window onto this community - the terminal screen - shows none of this. It looks like any other screen: a prompt, some text and a cursor. One has little sense of the presence of others.

This is exactly the type of information that this piece hopes to provide. The overall goal is to provide listeners information about the other people logged onto the system; its purpose is to provide an interface to give the computer users a sense of community.

AudioWho, preliminarily does not tackle the "patterns of association" concepts that are also stated in Donath's paper as that is probably too much information to provide audiotorily. As stated in the previous chapter, auralizations are not appropriate for quantitative information encoding – people cannot parse complex patterns provided to them through sound. Associative relationships may be possible on the small scale with a few distinct individuals and relationships (possibilities melodies which harmonize with each other when individuals are associated and which become very dissonant when disassociated), however when considering that

the average number of people logged onto `ml.media.mit.edu` is 60, this data set is too large to provide fine grained relational information in the auralization.

2.2 Data Gathering

The data being used is limited, but rich in the necessary information. Most modern Unix computers are distributed with the finger [Zim91] daemon which usually reveals which users are logged onto the system and their current idle time[1]. By implementing trivial networking software, *AudioWho* becomes privy to and drives its auralization from live data of the `ml` community.

What is important to note about the finger interface is that it does not provide that much granularity in information. Most implementations of finger (and our test computer, `ml.media.mit.edu`) only provides information about the idle times of users with a granularity of minutes. In order to provide a real time auralization, the renderings have to remain "busy" for a minute before it receives more data, at which point an unknown number of variables may change simultaneously. When a change occurs, the rendering needs to make a gracious change to accommodate the new data.

2.3 Design

AudioWho, renders its information directly to the speakers as it is intended to be running as a live installation. The system is very dynamic and the patterns of sounds it produces over periods of time can vary greatly.

The initial version of *AudioWho* is not based on human crowd sounds, but is worthwhile to describe as more recent versions (with crowd sounds) derive from this release. This first version attempts to evoke a jazz room type of feeling by playing well tuned and timed piano notes. As it is currently written, there are three "dimensions" – notes, volume, and tempo. Notes (the tone itself) directly represent the people in the system, volume describes the activity levels, while tempo is a rudimentary way of keeping persons in the system auralized.

2.3.1 Notes

The tones being played are selected from a 4 octave C jazz scale[2] surrounding middle C. This scale is chosen as it it is used extensively in musical improvisation, so therefore it is fairly easy to harmonize with anything (especially with itself); the usage of this scale prevents the "music" being produced from sounding atonal and/or dissonant. On start up, each username is assigned a random note from this scale, and that username retains that tone for the duration of the rendering (there may be a few "collisions" in tones with people as

[1] The finger protocol is relatively trivial to parse and understand – it merely requires a connection to port 79 of the requesting computer and the sending of a newline (ASCII character 10). The receiving computer then sends text back that can be easily tokenized to usernames, real names, and idle times.

[2] The scale C - D - E - F - G - A - B♭ - B is also known as the melodic minor scale.

there are usually more than 32 people logged in). Every minute, the set of users logged in may change, and these changes are reflected with the addition and deletions in the mapping from username to notes.

2.3.2 Volume

Volume characteristics of the rendering are tied directly to a user's idle time: for users who are idle less than 5 minutes, they are played at maximum volume and those idle for more than 45 minutes are not auralized at all. There are many users who log into `ml` and hide the window or not use it; these users are not being active in the community and therefore need not be auralized. Those who fall between the 5 minute and 45 minute region have their volume lineally fall off.

This dimension allows the listener to get a feel of how active people are and the whole community in general. As the activity of the people in the virtual space ebbs and flows, the volume of the space fluctuates with it; as people stop using their computers, their imprint on the rendering fades out with time.

2.3.3 Tempo

Finally, tempo also makes up an important part of this auralization. A decision was made for *AudioWho* to not hold the notes for in indefinite amount of time as that would mean there would be constant tones throughout the piece that would sound horrible. Instead, this auralization only plays the notes for short amount of times and to keep reminding the listener that the people are still there, the username to note mappings are are replayed with a uniform probability of 0.005 every 100 milliseconds (effectively once every 20 seconds). As there is no real way to rescan an auralization as people can with a visualization, this repeatedly sounding is a vital part of the rendering.

Because users may appear or disappear every minute due to updates, these users need to also be announced into the auralization. When a new user enters the system, the user is announced by repeating his or her note quickly five times sometime in the next minute (a random variable is used that is uniform over 60 for the number of seconds) The goal is to have the listener detect the announcing pattern from within the rest of the noise of the auralization to inform him or her that somebody new has entered.

2.4 Implementation

The initial version of *AudioWho* is authored in Java and relies heavily on the Java Sound API [jav]. The program runs as two concurrent threads – the first handling updating the internal data structures from outside data, and the second performs the rendering. A multi-threaded model is specifically being used to allow this piece to run in real-time; these threads are needed to prevent the rendering from pausing due to the unpredictability of IP-based networks. If, for some reason, the finger daemon takes time to connect to or to respond to a query, the auralization should not be blocked in turn. As this is designed to be run as a continuous installation, it should be protected against the variability of the Internet.

2.4.1 Data structures and updating

There is a shared data structure between the two threads which models the online population of `ml.media.mit.edu`. The `edu.mit.media.smg.audiowho.finger.Person` object maintains a mapping between username, a real name, and the user's idle time – and the shared data structure simply holds an array of these objects.

In a naïve implementation, the two threads (the one updating the data structures, and the one auralization the data structures) synchronize on the same data structures. This, needs to be done with special care, however, as network latencies may prevent the updating thread from flashing the data structures in a time manner, and therefore may lock out the other thread from properly auralizing. The two optimal solutions for this situation is to either buffer the entire finger stream before parsing it or to have the updating thread keep its own copy of the data structures, and once they are fully updated, obtain a lock on the shared data structure and replicate the data. This implementation relies on the former solution merely to prevent a waste of memory resources with significant amounts of object creation.

2.4.2 Music

Using the Java Sound API, *AudioWho* is able to get a handle to the MIDI subsystem in the running operating system. Using a few simple method calls, it can map all the users to different MIDI tones, the instrument for playing can be selected (a by product of using MIDI for auralization is that it can simply switch the instrument being used without needing to sample a new instrument), and it can play a note at a given volume and note. All this calls are abstracted behind an `edu.mit.media.smg.audiowho.Auralizer` interface in the `edu.mit.media.smg.MidiAuralizer` class.

2.5 Analysis

AudioWho did as expected – it provides a very simple and background interface to the list of people and their activities on `ml.media.mit.edu`. No detailed information is available via *AudioWho*, however it does provide a sense of "activity" when listening to it; there is a notably large difference to the sounds of the program when listening to it at 1PM on a Wednesday and at 3AM on a Monday. As more people log onto the system, the more cacophonous the music becomes. As there are less people using the system, the music changes to a tone every so often and far between.

The main issue with the design of *AudioWho* is that notes from a grand piano do not evoke the feeling of people. This interface strives to be unlike the ambientROOM [IU97] and more similar to an actual social space; the design goal of *AudioWho* is to evoke a sense of people, not of some other tangible interface. A secondary issue is to create an interface that does not have tonality problems. Computationally generating music is difficult, and artificially composing many different themes that can inter-mix in harmonious and

dissonant ways is even harder. These two problems are similar to the ones explained earlier when it was stated that the design restriction behind *murmurized* included focusing on voices. Revision 1 does exactly that.

2.6 Revision 1

The goal for the first revision is to tackle the design problem of making *AudioWho* evoke more of a personal feeling. The new version takes direct inspiration from Borofsky's "Three Chattering Men" [Bor] and instead of playing notes, the word "chatter" is used instead.

2.6.1 Implementation

Voices of a series of people were all recorded saying "chatter" and loaded into the second version of the *AudioWho* system. Instead of using Java Sound's ability to play MIDI notes, the WAV playback capability is instead used by the back-end to randomly access the WAV files and play those back; the code for this version of *AudioWho* is exactly the same as the previous version except instead of using a `edu.mit.media.smg.audiowho.MidiAuralizer` class, this version uses a `edu.mit.media.smg.audiowho.ChatterAuralizer` that maps and plays the WAV files instead of MIDI notes (both of these classes implement the `edu.mit.media.smg.audiowho.Aurilizer` interface). The algorithms for volume and tempo control are the same as the original version.

2.6.2 Analysis

Comments from people listening to this piece include

- "What are they saying?"
- "It sounds really confusing."
- "Way too many voices."
- "Are they saying 'Cheddar'?"

The word "chatter" is pretty hard to distinguish as opposed to other words, as the comment said. Either listeners would need to be told ahead of time that the word being said is "chatter", or they may never figure it out for themselves – "chatter" apparently needs context in order to parse as it sounds alike to other words.

AudioWho could possibly use more continuity across sound notifications. Right now, with only discrete notes or voices being used, there is no continuous pieces being played. For example, if using the system late at night, one may hear one voice being played every couple of minutes, if any voices at all. The system needs to be auralizing continuously so that a listener can tune in at any time and hear something that makes sense.

Also, it would be very useful to have *AudioWho* render some sort of interconnection between the people in the system – perhaps, as suggested before, a type of harmony could be played when multiple related people need to be auralized together. This needs to be carefully approached as, said before, it is a very hard problem to create good auralizations computationally.

On a high level however, this auralization (in either revision) does not produce satisfying crowd sounds. Replacing the words “chatter” with words or phrases may evoke a feeling of people talking to each other, but it is doubtful as simply playing random words or phrases at random intervals produces a random sound, not the organized (albeit chaotic) sound of a crowd. There are not enough variables of control to make this a very useful and general purpose renderer.

Chapter 3

SimParty

SimParty takes a bottom up approach to creating crowd sound auralizations by fully simulating a crowd (specifically a party). This project is an attempt at reaching the goals that *AudioWho* is not able to achieve in crowd sound generation.

A party is made up of the space the event is to be held in and the party goers. There may be music or food, but none of that is necessarily relevant – the basis of a party are the party goers themselves. Using simple agent based modeling techniques [MBLA96], *SimParty* attempts to model party goers with a state machine with probability weightings on the transitions between states. If a model can be derived for a single person's interaction behavior a room can be populated with many of these people and then the simulation can "listen" to this crowd.

This also leaves the ability to play with the characteristics of the different party goers. A simulation programmer can tweak the agents to be more likely to start a conversation, more likely to continue a conversation, or visa versa to make the parties "sound" different; the interaction qualities of the agents themselves, just like a real party, entirely control the overall effect.

3.1 Design

3.1.1 Party Goer

A party goer's interactions in the *SimParty* system, in a loose definition, is being modeled as a Markov chain[1]. There are a few exceptions to this definition as will be described below. A party goer is modeled by this state machine, two variables x and y, and the constants r and c. On every tick of the global party clock, a transition in the party goer's machine is possible.

[1]The listing of the transitions between the states can be described as generated by a finite-state machine with probabilities on the transitions.

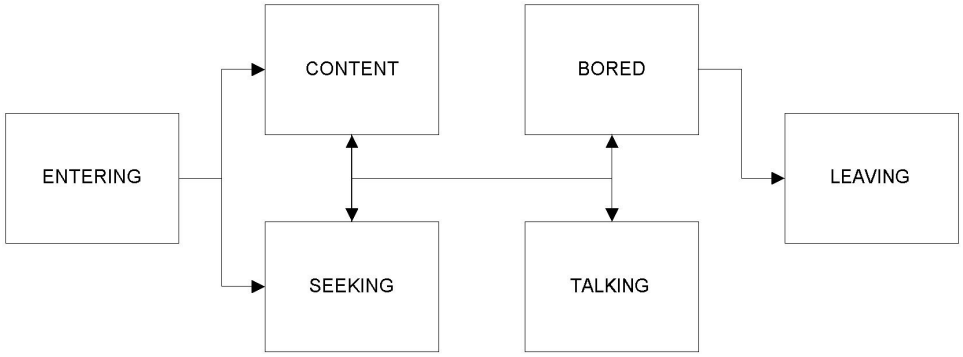

Figure 3-1: A limited view of the state machine behind each party goer in SimParty. The CONTENT, BORED, SEEKING, and TALKING states all point back to themselves, also.

State Machine

The states in the person's state machine are related to the party goer's "mood" at the time. They are displayed in figure 3-1 and are described as follows: ENTERING for when the party goer first enters the room, CONTENT if the party goer is happy just "hanging out", when the person is looking for another person to talk to he or she is SEEKING, if the party goer is on the verge of possibly leaving he or she is BORED, TALKING is the state describing that a person is engaged in conversation, and LEAVING occurs when the person disappears from the party. The transitions between these states have probability weightings that are name as the states that are being transitioned between (for example, the probability that a party goer will switch from ENTERING to CONTENT will be called ENTERING→CONTENT). ENTERING is a special case because in one "tick", a transition is guaranteed to occur to either CONTENT or SEEKING.

However, things get a little complicated when dealing with multiple people that may be within r. To solve this, a trumping order is necessary in case multiple state transitions are computed to different states. The trumping order is as follows: LEAVING, TALKING, CONTENT, SEEKING, BORED (a state of LEAVING supersedes everything, while a state of BORED is superseded by everything).

Location Variables

In a party, the party goer needs to physically move in order to make real interactions occur; if a person does not move, then he or she will not encounter other people to converse with and he or she will get bored very quickly. As an addition to the state machine, the party goer maintains two state variables, x and y, which denote the person's location. When in the SEEKING state, the party goer enters a random walk (simulated by adding a random value from -1 to 1 to both x and y).

Whether a party goer enters into conversations is directly related to its location. Therefore, the location is tied directly to the BORED→TALKING, SEEKING→TALKING, and CONTENT→TALKING transitions; all the *→TALKING are special cases and the transition is only taken into account when the party goer is within r of another person (r models the party goer's "personal space" or the space within which he or

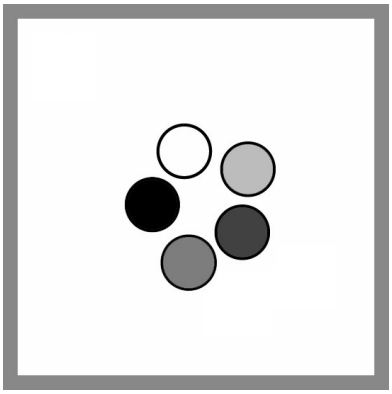

Figure 3-2: Ideally, what a group of agents in conversation (in TALKING would group together as.

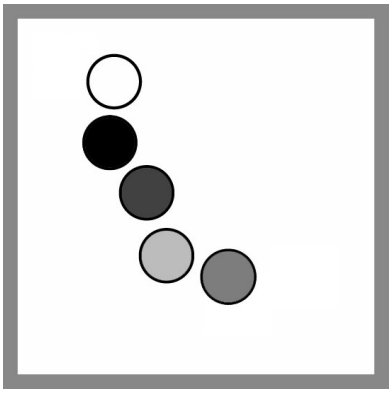

Figure 3-3: How a group of agents in conversation could look like using the simple modeling employed.

she feels obligated to consider beginning a conversation to prevent an awkward moment). When two agents come within r of each other, if neither is in TALKING then they both attempt to transition to TALKING – if both transition, then they begin a conversation. If an agent wanders into r of other agents in TALKING, it only needs to roll to see if it begins conversation with that group.

Transition probabilities are chosen depending on the personality of the person that is being modeled. A person that is easily bored is probably one that when not talking to somebody is too shy to seek out other people. The agent modeling that person has a very high *→BORED probabilities with very low *→SEEKING probabilities – this will prevent the party goer from entering the SEEKING state, but very easily will enter the BORED state (which means that this agent could very easily leave depending on the BORED→LEAVING transition). In contrast, a person who is very talkative will have very high *→TALKING transitions with very low *→BORED and *→CONTENT transition probabilities.

There are obvious flaws in this simple model. Ideally a group of people in conversation looks like a cluster (see figure 3-2) – this allows all agents to converse with each other. One defect in the simple modeling and conversation system being described above, is that conversational groups could form that look like figure

3-3 which is more of a game of “telephone” rather than a cluster.

Conversation Length

An agent needs to be able to transition out of the TALKING state, so to handle this possibility the conversation length variable, c comes into effect. When an agent enters into TALKING (another agent also needs to enter TALKING otherwise one agent is talking to him or herself), their individual conversational length clock begins to count down – each clock is individually set to c. When that clock is not at 0, the agent stays in the TALKING state, however when that clock hits 0, then the agent does have a chance of transitioning through the TALKING→BORED, TALKING→CONTENT, and TALKING→SEEKING arrows.

3.1.2 Party Space

A party space restricts the values of x and y that the party agents can wander in. It may also have much more specific conditions about space – for example, a restaurant or a cafe has a very different architectural setup than an open party room as they may have a bar, they may have tables, they may have counters, etc. Acoustics, in real spaces also have a direct effect on the mix. Many restaurants are setup with high ceilings or with tapestries to provide noise dampening to allow people to feel that they have private space. This is very different than the acoustics in a club where it is setup to have acoustics that cause the beat to reverberate throughout the space. The ambient music or noise in a space also effects the agent system greatly as faster paced music causes real people to interact with each other faster while the slower paced music causes people to linger more.

These different architectures cause differences in the type of crowds (and indirectly the type of crowd sounds) that will emerge: people may feel the need to restrict their movements or the type of behavior that is exhibited in certain areas of the room. The party space is the appropriate place to put all these considerations (although, the specific behavior modifications need go into the party goer) and for right now this is left to be open without any consideration for any of this except to restrict the location variables.

3.1.3 Listener

Finally, if we are going to listen to these crowd noises, a listener has to be placed appropriately in the party space. For a generic space, a listener can be placed effectively anywhere. However, with more detailed party spaces for the agents to wander within, the location of the listener can change the experience greatly.

In the above example of a cafe or a restaurant, a listener placed at a table will experience something very different from a listener that is wandering the space or is waiting next to the maître de or with a waitress. The one placed at table will experience a single intimate conversation with the other conversations in the periphery, whereas the maître de or the waitress experiences all the conversations, but all at a muted level.

3.2 Implementation

Comprising all the sound that is heard is the database of conversational fragments and on top of that, there are two major objects in the implementation of *SimParty*, the party goer agent, and the party space. These objects model a real party – the room itself is very dumb, and all the interactions occur because there are people with real agendas and people to talk to. This system is all implemented in Java with a multi-threaded model in mind to allow the agents to interact with each other and converse simultaneously.

3.2.1 Conversational Fragments

The sound that *SimParty* emits is generated from the simultaneously playing of selections from a large conversational fragment database. This implementation of *SimParty* assumes that the parties being simulated are large enough that individual conversations will not be able to be extracted from the noise – this assumption is key to the way conversations are played; although modeling conversations would seem to be an obvious extension to *SimParty*, it seemed out of scope.

The simulator has access to a database of conversational fragments that have been recorded from students in the Sociable Media Group that it randomly strews together. Fragments are classified as "greetings", "partings", and "filler" and are played appropriately. These three groups are important because as people tend to use a different volumes and inflections of speech for greeting and parting. Therefore, it is important to get that "feel" into the sound by playing a greeting first, a lot of filler, and then a parting when the agent transitions out of TALKING. Listening to a single conversation that *SimParty* strings together will not make much sense, however, the hope is that when enough of these are played simultaneously, one will not be able to hear an individual synthetic conversation anyway.

3.2.2 Party Goer

`edu.mit.media.smg.voice.simparty.AudioPartyGoer` is the agent class that can carry on a conversation. This agent is managed by the state machine, variables, and constants described above. A static method to construct a random `AudioPartyGoer` is provided in the class and it initializes all the transition values to be random (it also chooses a random conversational length that is about 10 minutes long with a Gaussian window that has a standard deviation of 3 minutes around it). It is possible to specify more exactly the transitions by using the standard constructor as this is much more useful when trying to create a very specific crowd for auralization. Once an `AudioPartyGoer` is created, it is assigned a $< x, y >$ coordinate inside the party space and it is let free.

After a number of `AudioPartyGoer`s are created and placed inside the party space, the simulation can be started by ticking the global clock. On each tick, all those agents not in the TALKING state check roll a die to see what transition they should make inside their state machine (those that transition to SEEKING need to also randomly alter their $< x, y >$ within the party). If an agent wanders within r of another agent

(or has another agent that wanders within r of it) then it needs to determine whether or not it will transition to TALKING. For those other agents that are in TALKING, they simply decrement their conversational counts until those reach 0. Those who have reached 0 need to start rolling dies to see if they need to transition out of TALKING. Finally, those that go into LEAVING are removed from the party space.

Of course, those that are in TALKING play conversational sounds by accessing the database as described above.

3.2.3 The Party Space and the Listener

The party space and the listener are joined into one object in this implementation – while the two play very distinct roles, it made more sense to join the two objects into an object that sets bounds on the $< x, y >$; when the party goer wants to move, the party space performs a check on the new coordinates first to make sure that they are within bounds. The space, in this implementation, does not have any distinguishing characteristics.

Located at the center of the space, conceptually, is the listener (at $< \frac{x_{\max}}{2}, \frac{y_{\max}}{2} >$). It faces up towards $< \frac{x_{\max}}{2}, y_{\max} >$ and effectively divides the room into two halves from which we can render the auralization. Any conversations occurring to the right of the listener go to the right audio channel, and those occurring to the left of the listener go to the left audio channel. Volume is controlled by the square of the distance of the conversation to the listener.

3.3 Analysis

The *SimParty* system is very versatile and capable of rendering a wide variety of different crowd sounds. A very talkative and interactive crowd can be designed by setting the c to a very high value, and making all the *→TALKING probabilities be very high (one would also probably set the *→CONTENT and *→BORED transitions to be very low. Whereas a very apathetic crowd, or a crowd that is most likely to dissipate, can have all the *→BORED transitions be very likely and the *→TALKING transitions be very unlikely so that all the agents can all go into the LEAVING state.

Aside from the defect in the current release, *SimParty* requires a few major changes in order to make it better suited for its job. Right now, the system assumes that all people will want to talk to all people; in this framework, it is hard to model environments similar to a lecture hall, or similar to a Usenet scenario where people talk, but nobody responds. The simulator also needs more work to take into account how the setting interacts with the individual agents as this is crucial for the modeling of cafe's versus dance halls versus lecture halls. The setting directly modifies how the agents interact with each other through physical barriers and possibly social norms in the space.

However, the problem with this setup is that it is logistically "hard" to maintain a good sounding crowd. *SimParty* not only has to maintain and move a list of party goers around the space, but it has to maintain access to a large library of conversational fragments so it can auralize them. The *Nonsense Machine* exists to

combat these resource heavy problems.

Chapter 4

The *Nonsense Machine*

The *Nonsense Machine* is the most ambitious of the *murmurized* projects as it is a study in artificially generating crowd sounds for the use in auralizations. The goal of this auralization technique is to develop a system that requires less resources than the previous large scale auralizer, *SimParty*. *SimParty* requires access to a library of conversational fragments, a library that is too large to fit in memory and instead needs to be stored on the hard disk or some other temporary storage medium. Another limitation in the design of *SimParty* is that it requires a decent amount of computational resources as it needs to track the location of each agent and as each of the agents auralizes in its own thread so that the conversations may be played in parallel.

When a person steps back from within a crowd and listens to it, the individual conversations are lost and all that becomes of the crowd is "well created noise" with a discernible mood or affect to it. The individual conversations of the people involved gets lost. By monopolizing on this, the *Nonsense Machine* becomes effectively single threaded (it has other threads for house keeping) and it does not require secondary storage for conversational threads. This machine stores all it needs in memory (as its requirements are very small) and can recreate everything from that.

4.1 Design and Theory

Psychologically, when actively listening to a large crowd, the conversations within become incomprehensible. There are a few reasons for this, but it mainly is due to sensory overload; there are simply too many signals being presented to the human mind simultaneously, that on a first glance, it cannot separate them out. The *Nonsense Machine* attempts to monopolize on this sensation.

There are two main "tricks" that can be used to produce sounds that can make this all happen properly: the manipulation of voiced and unvoiced sounds and the use of the "crowds" effect.

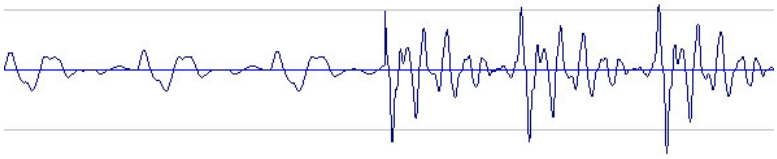

Figure 4-1: A simple concatenation of the “r” unvoiced and “a” voiced phonemes. Note the sharp transition between the two phonemes.

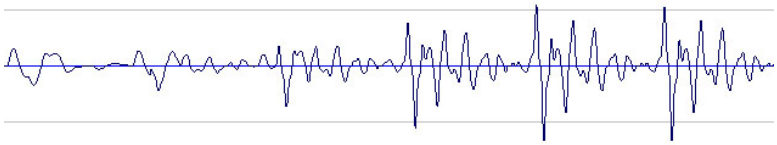

Figure 4-2: Instead of a simple concatenation (as in figure 4-1), cross-fading them provides a much cleaner transition.

4.1.1 Voiced and Unvoiced sounds

In the *Nonsense Machine* full conversations are no longer used – all the ear needs to hear are voiced and unvoiced sounds. If one takes an average of all the words in common use in English, one gets approximately 70% voiced sounds and 30% unvoiced sounds[1]. By presenting the ear a sound that is comprised of “nonsense sentences” that have this ratio, the ear may be able to be fooled into thinking that it is hearing English conversations as nothing can be parsed from the noise, but it comprised of the proper sounds.

Generating these nonsense sentences is partially possible through phoneme manipulation. By recombining these phonemes in the 7/3 ratio of voiced and unvoiced, nonsense type words are created. Careful blending of phonemes have to be taken into account to make sure that the waveforms of the two words cross fade together and there is not a sharp transition. When simply appended to each other (figure 4-1), the transition seems very harsh and the ear locks on to the transition clicking instead of the sound itself – by performing simple cross fading (figure 4-2), the transition becomes a lot smoother and the ear notices the transition less and it can concentrate on the sound instead.

A future piece of overall work in this space will have to work more carefully with the phoneme combinations. Phonemes only make sense within context, ie the CH sound sounds differently if it is at the beginning of a sentence, surrounded by certain other phonemes, or at the end of the sentence. These context models could used by creating a matrix of sounds that key a phoneme with the previous and the next phoneme to allow the program to get the proper sound. The current *Nonsense Machine* does not take this into account.

[1] Voiced sounds require the vocal cords to be moving, whereas unvoiced sounds comprise clicks and hissing type noises.

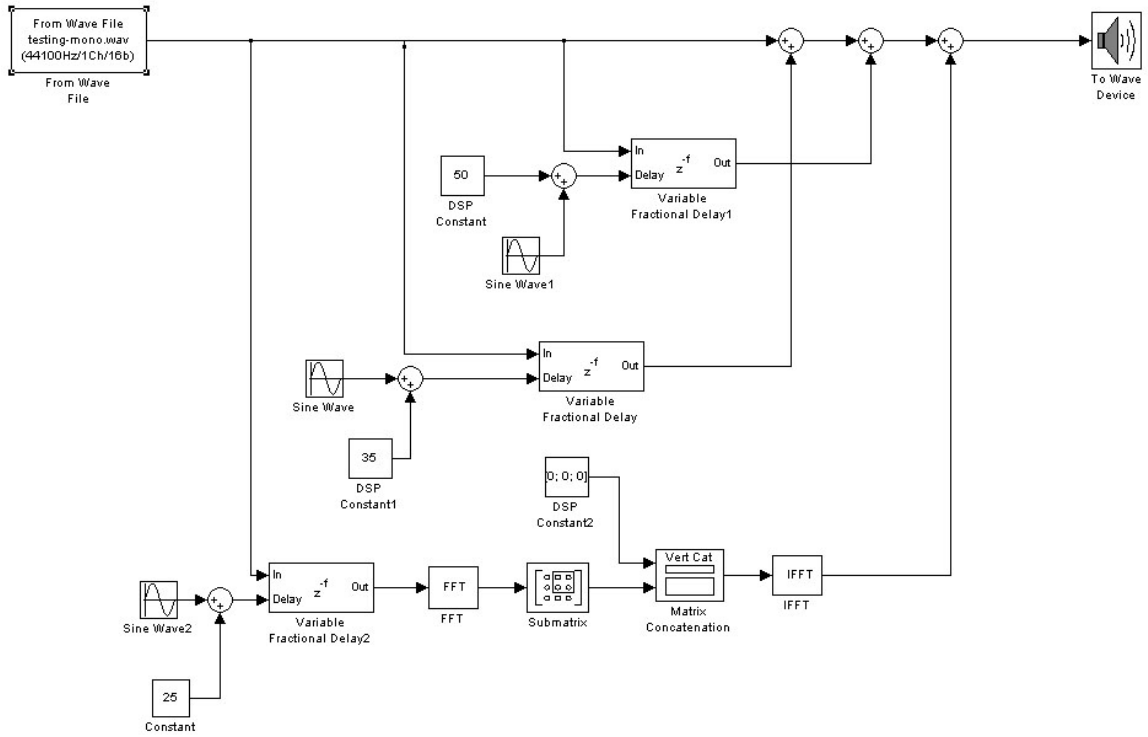

Figure 4-3: A model of a possible crowds effect generator (courtesy of Simulink). This model performs three modulated delay loops with one of the delay loops having a frequency shift.

4.1.2 Chorus Effect

There is a special effect used in music and in other sound related work called flanging which uses two copies of the same signal, with the second signal delayed slightly to produce a swirling effect (Conceptually to produce this effect, take two tapes of the same recording and synchronously play them back. While they are playing, repeatedly slow down the speed of one of the tapes and bring it back up to the original speed. This will produce the “swooshing” effect). This effect has a direct relationship to the effect used to make singer’s voices bigger during the post production process. The chorus effect is manually produced by having a singer first record his or her voice singing a portion of the song, and then recording another track by singing with his or her first recording. The minor differences in the qualities of the voice between the two recordings has an effect of making the singer’s voice sound bigger. This effect may also be produced digitally by playing a signal with itself delayed approximately 20 to 60 ms (anything less than 20 ms will cause much phase cancellation, and anything larger than 60 ms usually causes an echo in the sound). Slight modulation in amplitude and frequency adds some reality via vibrato.

To further the *Nonsense Machine*’s effectiveness, chorusing is applied to the signals being created in the system (a sample model for the implemented effect is demonstrated in Simulink and included in figure 4-3). This addition of extra signals not only makes the crowd sound bigger, but it also causes the sounds to be more

realistic as they sound fuller and very much unlike the monotonous phonemes that are being combined for the synthesis.

4.2 Implementation

The *Nonsense Machine* implementation is effectively single threaded (it could be considered to be multi-threaded simply because the way that Java Sound handles playback. It spawns its own asynchronous objects for notification and management). On start-up, the program creates a whole slew of nonsense sentences and applies the chorus effect as modeled in figure 4-3 to all the sentences. All these sounds are held in memory as `javax.sound.sampled.AudioInputStream`s, and therefore can be manipulated by Java methods and objects. The end goal of the start-up methods in the *Nonsense Machine* is to create a set of `AudioInputStream`s to be recombined during the auralization.

For playing the sounds back, this implementation simply makes sure there are 10 ± 4 nonsense sentences playing at any given time. This gives the illusion that at sometimes there can be more conversations than other times (taking the *SimParty* example) people are in SEEKING or CONTENT, so therefore they are not talking).

4.3 Analysis

At first glance, this machine seems perfect! It has all the capabilities of *SimParty*, yet requires far less computational resources. There are a few limitations to the design, however which become immediately evident when attempting to use it in a real auralization scenario; while the *Nonsense Machine* works well "in the lab", it has some serious limitations when attempting to deploy.

Currently, the parameters to tweak to make the auralization sound as desired are very limited. All that can be modified are

- Rate at which phonemes are combined to make nonsense sentences
- Number of simultaneous nonsense sentences
- Phoneme samples being used

This is the main flaw in this piece; there are just too little parameters to play with in an auralization. Currently, there are problems with manipulating any of these parameters. Phoneme samples need to be setup to be lengths that correspond to normal speakers. Anything longer or shorter than the norm will be characterized as the "speaker" speaking too quickly or slowly.

Playing with the number of simultaneous sentences does not "break" the machine, unless the number is being driven to a very small level (ie 2 ± 1). When the *Nonsense Machine* is told to generate a crowd with

such a small simultaneous speaking system, a similar problem occurs when the *SimParty* is asked to generate a small crowd. While in *SimParty*, the problem with not having enough simultaneous conversations is that the conversational fragments do not make sense with one another; the problem in the *Nonsense Machine* is that real sentences are not being used and instead only a mishmash of phonemes will be heard.

Finally, the machine is coded with the phoneme library from `ibiblio.com`, or the standard phoneme library with a male voice pronouncing all the sounds. The machine's output when using this library directly, has been summed up as "Buddhist monks chanting". The only recourse to prevent this is to use a different phoneme library in conjunction, or attempt to implement frequency filters to manipulate these phonemes before constructing nonsense sentences from them as the voice tends to change pitch while speaking.

Right now, the *Nonsense Machine* is appropriate as a curiosity and can be played without having a tie into live data. To make the leap to live access, work needs to be performed on how to smooth out the variables (number of people in the room, different affect on the phonemes, etc.) in order to allow for live data to manipulate it in real time.

Chapter 5

Building upon *murmurized*

The work on *murmurized* is no where near complete. At best, this trio is only a stab in the direction of using human voices for auralization purposes. All three projects have their strengths and their weaknesses, however there is clearly a lot of research involved in order to make any of these projects fully available for prime-time use; there is not going to be *murmurized* toolkit anytime soon that programmers can simply import into their source code and use.

Each part of *murmurized* is good at performing a specific task – *AudioWho* is suited for presence notification, *SimParty* is good for generic crowd creation and auralization, and the *Nonsense Machine* is suited for computationally limited environments. Crucial work needs done on each project and on how to unify all three of these characteristics.

There are a few applications that *murmurized* can be tied into usefully. Instant messenger applications currently rely on a stack based model to inform users that "buddies" have logged on and off the system – when somebody comes onto the system a door opens, and when one logs off the system a door closes. If between these door openings and closings one wants to know the state of his social online network, that user needs to pull the instant messenger window to the foreground to take a look. If, instead, *murmurized*-type sounds were being played through the computer, the user may never need to pull up the buddy list unless he or she wants detailed information of who is online. Otherwise, the constant chatter of the auralization in the background could provide the representational information.

A second tool that can be built is a type of Usenet surfing device where Usenet group names are listed across the screen and when the user places his or her mouse over the name, the "sounds" of that group are played. This tool could allow a user to be able to decide which group he or she is most comfortable interacting with similar to how he or she may walk into a party and hear a mood that he or she may or may not want to converse with at the time. This could be further broken down by allowing a user to "listen" to a thread of a conversation to decide (without having to read the text) whether or not he or she would like to join that conversation.

Chapter 6

Conclusion

This thesis presented *murmurized*, a trio of projects that attempts to aid other people who are interested in auralizing social data through the instrument of the human voice. Through these types of auralizations, listeners can begin to understand patterns of social interaction. *murmurized* attempts to build up these auralizations automatically using three different models for the crowd sound generation.

murmurized's projects each works in different ways to model and present social interactions by presenting parameters that others can tweak and tie to real data to help auralize social presence and social interactions. Each of the three pieces, *AudioWho*, *SimParty*, and the *Nonsense Machine* are first steps in understanding how to properly use the sounds of the human voice and crowd sounds to understand data sets comprised of social interactions. I have, throughout this thesis, explained how these pieces fit into *murmurized* and have also detailed questions that have been left unanswered.

Understanding social patterns can be very intensive, and using only visually based tools require the full concentration and attention of the viewer. Sound, however, is something which can be easily moved into the foreground or the background as according to the listener's desires; he or she can simply "tune out" anything he or she does not want to listen to, but still keep the sounds in the background to pay attention to subconsciously. I feel *murmurized* represents the appropriate first steps in creating these types of interfaces.

Bibliography

[Bor] Jonathan Borofsky. Three chattering men.

[DdbLR] Judith Donath, danah boyd, Hyun-Yeul Lee, and Dan Ramage. Loom2. http://smg.media.mit.edu/projects/loom2/.

[Don95] Judith Donath. Visual who: Animating the affinities and activities of an electronic community. In *ACM Multimedia*, 95.

[Gar97] William Gardner. *3-D Audio Using Loudspeakers*. PhD thesis, MIT Media Lab, 1997.

[IR98] Leland K. Irvine and Roy L. Richards. *Acoustics and Noise Control Handbook for Architects and Builders*. Krieger Publishing Co., 1998.

[IU97] Hiroshi Ishii and Brygg Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *CHI*, pages 234–241, 1997.

[jav] Java sound api. http://java.sun.com/products/java-media/sound/.

[LB92] H. Lehnert and J. Blauert. *Applied Acoustics*, volume 36, chapter Principles of Binaural Room Simulation, pages 259–291. 1992.

[Mar] Jane Knowles Marshall. An introduction to film sound. *filmsound.org*.

[MBLA96] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system, a toolkit for building multi-agent simulations, 1996.

[Pom99] F. Joseph Pompei. *J. Audio Eng. Soc.*, volume 47, chapter The Audio Spotlight, pages 726–731. 1999.

[WID+98] C. Wisneski, H. Ishii, A. Dahley, M. Gorbet, S. Brave, B. Ullmer, and P. Yarin. Ambient displays: turning architectural space into an interface between people and digital information. In *CoBuild'98: First International Workshop on Cooperative Buildings*, pages 22–32, 1998.

[Zim91] D. Zimmerman. RFC 1288: The Finger User Information Protocol, December 1991.