THE ARCHITECTURE OF AURACLE: A VOICE-CONTROLLED, NETWORKED SOUND INSTRUMENT

Jason Freeman
Georgia Institute of Technology
Music Department

Sekhar Ramakrishnan
Zentrum für Kunst und Neue Medientechnologie

Kristjan Varnik
Akademie Schloss Solitude

Max Neuhaus

Phil Burk
SoftSynth

David Birchfield
Arizona State University
Arts, Media, and Engineering

ABSTRACT

Auracle is a voice-controlled, networked sound instrument which enables users to control a synthesized instrument with their voice and to interact with each other in real time over the Internet. This paper describes the architecture of the system in detail, including the multi-level analysis of vocal input, the communication of that analysis data across the network, and the mapping of that data onto a software synthesizer.

1. INTRODUCTION

Auracle is a voice-controlled, networked sound instrument conceived by Max Neuhaus and realized collaboratively by the authors. Users interact with each other in real time over the Internet, playing synthesized instruments together in a group 'jam'. Each instrument is entirely controlled by a user's voice, taking advantage of the sophisticated vocal control which people naturally develop learning to speak. The project was designed to facilitate the kinds of communal sound dialogue which are rare in contemporary society:

> […Auracle and my earlier broadcast works are…] proposing to reinstate a kind of music which we have forgotten about and which is perhaps the original of the impulse for music in man. Not making a musical product to be listened to, but forming a dialogue, a dialogue without language, a sound dialogue. These pieces then are about taking ordinary people and somehow putting them in a situation where they can start this nonverbal dialogue. [8]

Auracle is an entity made to give the lay public this opportunity. It was designed to be accessible to people without musical training or technical expertise. We strived to create an open-ended architecture rather than a musical composition: a system which, as much as possible, responds to but does not direct the activities of its users. We also sought to build a highly transparent system, in which users could easily identify their own contributions within the ensemble, while also remaining engaged over extended periods of time.

2. ARCHITECTURE

Users launch Auracle from the project's web site, opening a graphical user interface through which they can 'jam' with other users logged in from around the world. To control their instrument, users input vocal gestures into a microphone. Their gestures are analyzed, reduced into control data, and sent to a central server. The server broadcasts that data back to all participating users within their ensemble. Each client computer receives the data and uses it to control a software synthesizer.

The client software is implemented as a Java applet incorporating the JSyn plugin [2], and real-time collaboration is handled by a server running TransJam [3]. Data logging for debugging, usage analysis, and long-term system adaptation is handled by an HTTP post (from Java) on the client side and PHP/MySQL scripts on the server side.

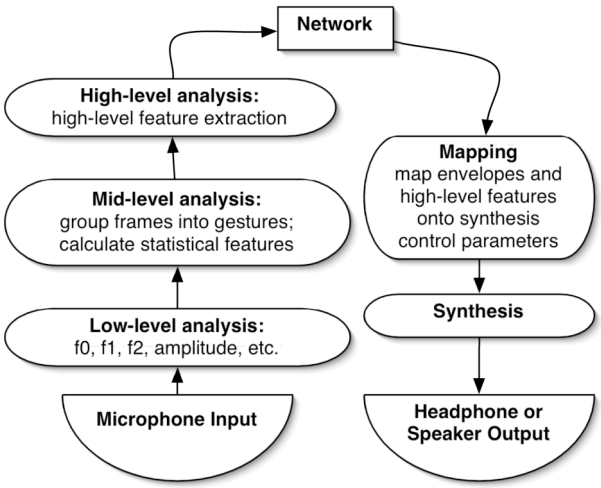

Figure 1. The overall architecture of Auracle.

2.1. Low-level Analysis

The initial low-level analysis of the voice computes basic features of the audio signal over an analysis window [10]. The incoming sound is analyzed for voicedness / unvoicedness, fundamental frequency, the first two formant frequencies with their respective formant bandwidths, and root mean square (RMS) amplitude. JSyn is used to capture the input from a microphone, but it cannot extract the vocal parameters we need, so we built this functionality ourselves in pure Java using linear prediction, feeling it would be the easiest approach to implement in pure Java with acceptable performance and accuracy.

Raw sample data from the microphone is brought from JSyn into Java. Once in Java, the data is determined to be voiced or unvoiced based on the zero-crossing count. Following Rabiner and Schafer [9], the data is downsampled to 8192 Hz and broken into 40 ms blocks, which are analyzed by LP for the following characteristics: fundamental frequency, the first and second formant frequencies, and the bandwidth of each formant. RMS amplitude values are also calculated for each block of input. The values for each block of analysis are fed into a median smoothing filter [9] to produce the low-level feature value for that analysis frame.

2.2. Mid-level Analysis

The mid-level analysis parses the incoming low-level analysis data into gestures. Since users are asked to hold down a play button while they are making a sound, it was trivial to parse vocal input based on the button's press and release.

Once a gesture is identified, a feature vector of statistical parameters is created to describe the entire gesture. The choice of features is based largely on previous studies of vocal signal analysis for emotion classification [1] [4] [11].

2.3. High-level Analysis

It is theoretically possible to directly transmit each 43-element mid-level feature vector across the network and to map that vector onto synthesis control parameters, but we found it impractical in practice to directly address this amount of data.

Instead, we perform a high-level analysis which projects the 43-dimensional mid-level feature space onto three dimensions. We were attracted to the use of Principal Components Analysis (PCA) to generate this projection, because it preserves the greatest possible amount of variance in the original data set and facilitates a self-organizing, user-driven approach.

But PCA creates a static projection; for Auracle, we wanted a dynamic approach which could perform both short-term adaptation — by changing over the course of a single user session to focus on the mid-level features varied most by that user — and long-term adaptation, in which the classifier's initial state for each session slowly changes to concentrate on the mid-level features most varied by the entire Auracle user base. To do so, we implemented PCA using Adaptive Principal Component EXtraction (APEX) model, which incorporates a neural network [5] [6] [7].

2.4. Network

Each gesture's low-level analysis envelopes, along with the high-level feature values, are sent to a central server running TransJam [3], a Java server for distributed music applications. The TransJam server provides a mechanism to create shared objects, acquire locks on those objects, and distribute notifications of changes to those objects. Each client sends its gesture data as a modification to a data object which it has locked, and the server then transmits the updated object information to all clients in the ensemble. In this manner, all client machines maintain all players' analysis data in sync. By sending only control data, Auracle maintains low latency and high audio quality using a fraction of the bandwidth required for audio streaming.

Java security restrictions and practical networking issues made direct peer-to-peer communication impossible. This necessitates a central server. However, to mitigate the probability of a performance bottleneck, Auracle's architecture is designed to minimize the work done by the server. The server is merely a conduit for data and does no processing itself. Mapping and synthesis operations are duplicated by all clients, but we preferred this solution over adding load on the server.

Auracle is designed to facilitate a conversational style of interaction, in which players respond to earlier sounds they hear instead of planning simultaneous gestures with other players. The analysis data is transmitted to the server only once a complete gesture has been detected, and data is only mapped onto synthesis control parameters when it arrives from the server, even when the data was created by the local client. And the onset of gestures from different players are occasionally shifted, and the gestures are slightly compressed in time, in order to minimize the overlap of gestures from different players. Because of this design, network latency is not generally an issue, and individual gestures need not be time-stamped to exactly synchronize their playback on all client machines.

2.5. Mapping and Synthesis

Each client receives the data from the server and passes it to a mapper, which generates envelopes and control parameters for a software synthesizer. The mapper manages an entire ensemble of synthesis instruments, each of which is controlled by the vocal gestures of a single player.

The synthesis algorithm, implemented entirely using the JSyn API [2], is a hybrid of several techniques, designed to enable the mapping of player data onto a wide range of timbres. The synthesizer is composed of three separate sections: an excitation source, a resonator, and a filter bank. The initial excitation is composed of two sources — a pulse oscillator and a frequency-modulated sine oscillator. These are mixed and sent through an extended comb filter with an averaging lowpass filter and probabilistic signal inverter included in the feedback loop. The result is sent through a bank of bandpass filters and mixed with the unfiltered sound to generate the final output.

Much of the low-level analysis data is mapped onto the synthesis algorithm in straightforward ways. The fundamental frequency envelope controls the frequency of the excitation sources and the length of the feedback delay line. The amplitude envelope controls the amplitude of the excitation source, the overall amplitude of the synthesizer, and the depth of frequency

modulation. The first and second formant envelopes are used to set the centre frequencies of the bandpass filters, and the Q on those filters are inversely proportional to the formant bandwidth envelopes. The voicedness / unvoicedness envelope of a gesture modulates the probabilistic signal inverter between noisier and purer timbres.

High-level feature data, on the other hand, is used to control timbral aspects of the synthesis which evolve from one gesture to the next but do not change within a single gesture: the ratio of pulse to sine generators in the excitation source, the probability of inverting the feedback signal, and the filter Q values. In the latter two cases, a high-level feature value defines a range within which the parameter can vary over the course of the gesture. Then low-level envelopes — voicedness / unvoicedness level and formant bandwidths, respectively — control continuous, subtle variations within that range over the course of the gesture.

We did not want sound output to stop completely when users were not making vocal gestures. So when a player's synthesizer is finished playing a gesture, it continues sounding a quiet 'after ring' until the next user gesture is received. The relationship of the vocal gesture to this after ring is less transparent than with the gesture itself; it is designed simply to be a quiet sound which constantly but subtly changes.

2.6. Graphical User Interface

The focus of Auracle is on aural interaction, so the software's graphical user interface is deliberately sparse. The main display area shows information about all users in the active ensemble of players: their usernames, their approximate locations on a world map (computed with an IP-to-location service), and a running view of the gestures they make (displayed as a series of coloured squiggles corresponding to their amplitude, fundamental frequency, and formant envelopes).

Users push and hold a large play button when they want to make a vocal gesture. Additional controls allow them to move to another ensemble, create a new ensemble, and monitor and adjust audio levels. A text chat among players within the ensemble is available in a separate popup window.

3. DISCUSSION

Auracle was officially launched to the public in October 2004 — on the Internet, at Donnaueschinger Musiktage in Germany, and during a live radio event on SWR. We have been thrilled to see how Auracle engages people ranging from non-musicians to trained singers, of many different ages and cultural backgrounds. Inevitably, some players are shy, have difficulty thinking of vocal gestures, and quit after a few minutes. But we have observed many players interacting with Auracle for over thirty minutes, enjoying the identification of their voices in the sounds they hear and the surprise in hearing those transformations and the responses of others. Over time, Auracle encourages them to create an increasing variety of vocal sounds — whether whistling, gurgling, shouting, or singing — as they strive to explore the

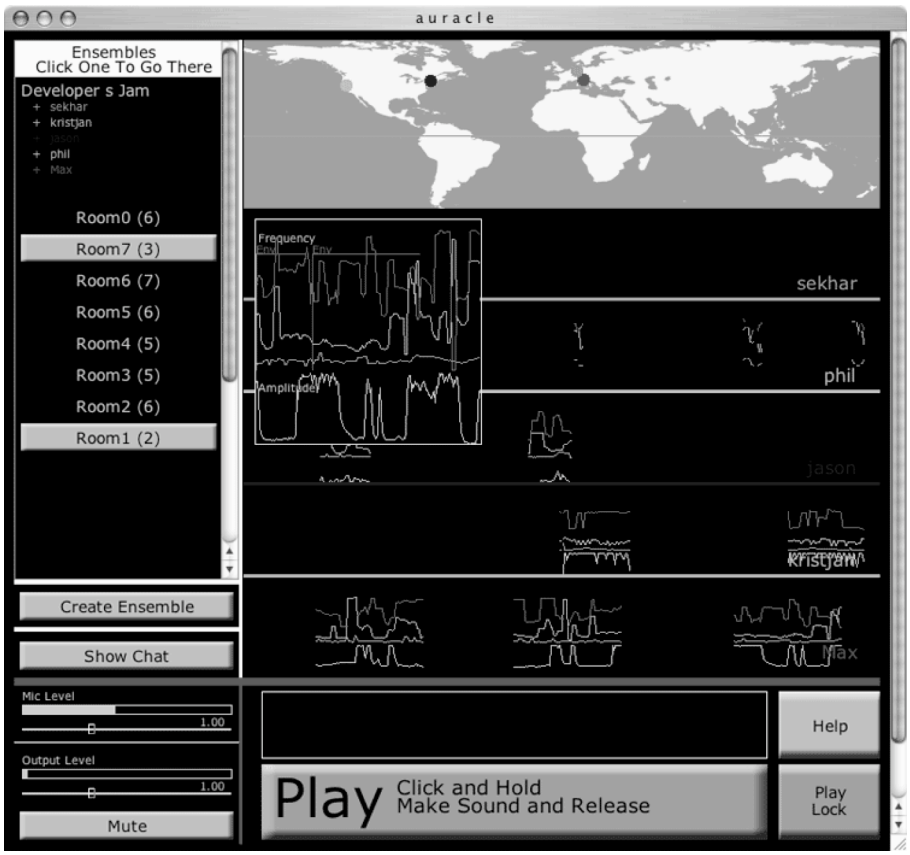

Figure 2. The Auracle graphical user interface.

boundaries of the system.

In our own regular 'developer jams' on the system, we developed strategies for collaboratively structuring our interaction over extended periods of time. Often, one of us would suddenly start making gestures which radically departed from the current sounds in frequency, density, noise content, or dynamics, and the rest of the players would gradually begin to imitate them. And sometimes, we would use Auracle's text chat functionality to plan such changes more deliberately.

During the eight months beginning October 15, 2004, there were 1494 user sessions on Auracle, with 803 usernames connecting from 717 distinct hosts. The average session length was 14.8 minutes. The majority of those users play Auracle alone. While Auracle is still engaging when played in this manner, it is most interesting when users are online at the same time and can 'jam' together.

We have experimented with a variety of strategies to help users find each other online, including scheduling regular online events and encouraging users to schedule Auracle meetings with friends, but these techniques have had limited success. Our most successful Auracle events, ironically, have taken place in the physical world, with several computers set up as kiosks on which people can try Auracle. We are continuing to present Auracle in this format at events such as this conference.

In the long term, we hope to draw enough users to Auracle so that there are always ensembles of players online. In this regard, we are focusing not only on drawing more users to the site, but also on getting more of them to log in and participate once they arrive. Many users visit the site but never successfully launch Auracle and make a sound. While we are working to improve site documentation to help them more easily test and configure their audio system, our informal polling indicates that the majority of these users simply lack computer microphones. With the growing popularity of online audio chat and telephony applications, we hope that computer microphones will become more ubiquitous in the coming years.

We are also working to make Auracle more accessible to members of the computer music community. We are preparing much of the Java source code for release under an open-source license, so that others may leverage our development work in their own projects, and we are also creating a Software Developer's Kit which would enable Java and JSyn [2] developers to contribute mapping and synthesis components to the project. By opening Auracle development to new contributors, we hope that the project will evolve in new ways and new directions which we could not have envisioned ourselves.

4. ACKNOWLEDGEMENTS

The Auracle project is a production of Max Neuhaus and Akademie Schloss Solitude (art, science, and business program) with the financial support from the Landesstiftung Baden-Würtemburg. We express our gratitude for their generous support. Auracle is available at http://auracle.org.

5. REFERENCES

[1] Banse, R., and K. Scherer. 1996. Acoustic Profiles in Vocal Emotion Expression. *Journal of Personality and Social Psychology*, 70 (3): 614-636.

[2] Burk, P. 1998. JSyn – A Real-time Synthesis API for Java. *Proceedings of the 1998 International Computer Music Conference*. Ann Arbor, MI: ICMA, 252-255.

[3] Burk, P. 2000. Jammin' on the web – a new client/server architecture for multi-user performance. *Proceedings of the 2000 International Music Conference*. Berlin, Germany: ICMA, 117-120.

[4] Cowie, R., E. Douglas-Cowie, N. Tsapatsoulis, G. Votsis, S. Kollias, W. Fellenz, and J. Taylor. 2001. Emotion Recognition in Human-Computer Interaction. *IEEE Signal Processing Magazine*, January 2001, 32-80.

[5] Diamantaras, K. and S. Y. Kung. 1996. *Principal Component Neural Networks*. New York: John Wiley and Sons, Inc.

[6] Freeman, J., C. Ramakrishnan, K. Varnik, M. Neuhaus, P. Burk, and D. Birchfield. 2004. Adaptive High-level Classification of Vocal Gestures Within a Networked Sound Environment. *Proceedings of the 2004 International Computer Music Conference*. Miami, FL, ICMA: 668-671.

[7] Kung, S., K. Diamantaras, and J. Taur. 1994. Adaptive Principal Component EXtraction (APEX) and Applications. *IEEE Transactions on Signal Processing*, 42 (5), 1202-1217.

[8] Neuhaus, M. 1994. The Broadcast Works and Audium. In *Zeitgleich*. Vienna: Triton, 1994. http://auracle.org/docs/Neuhaus_Networks.pdf

[9] Rabiner, L. and R. Schafer. 1978. *Digital Processing of Speech Signals*. Englewood Cliffs, NJ, Prentice-Hall.

[10] Ramakrishnan, C., J. Freeman, K. Varnik, D. Birchfield, P. Burk, and M. Neuhaus. 2004. The Architecture of Auracle: A Real-Time, Distributed, Collaborative Instrument. *Proceedings of the 2004 Conference on New Interfaces for Musical Expression*. Hamamatsu, ACM: 100-103.

[11] Yacoub, S., S. Simske, X. Lin, and J. Burns. 2003. Recognition of Emotions in Interactive Voice Response Systems. http://www.hpl.hp.com/techreports/2003/HPL-2003-136.html.