

The Architecture of Auracle: a Real-Time, Distributed, Collaborative Instrument

C. Ramakrishnan
Akademie Schloss Solitude
Solitude 3
D-70197 Stuttgart
Germany

cramakrishnan@acm.org

Jason Freeman
Columbia University
2960 Broadway
621 Dodge Hall
New York, NY 10027

jason@jasonfreeman.net

Kristjan Varnik
Akademie Schloss Solitude
Solitude 3
D-70197 Stuttgart
Germany

kristjan.varnik@akademie-solitude.de

ABSTRACT

Auracle is a “group instrument,” controlled by the voice, for real-time, interactive, distributed music making over the Internet. It is implemented in the JavaTM programming language using a combination of publicly available libraries (JSyn and TransJam) and custom-built components. This paper describes how the various pieces — the voice analysis, network communication, and sound synthesis — are individually built and how they are combined to form Auracle.

Keywords

Interactive Music Systems, Networking and Control, Voice and Speech Analysis, Auracle, JSyn, TransJam, Linear Prediction, Neural Networks, Voice Interface, Open Sound Control

1. INTRODUCTION

Auracle is a new collaborative, networked, voice-controlled musical instrument conceived by Max Neuhaus and realized collaboratively by the authors. Users run a Java applet through which they can “jam” with others around the world. The applet analyzes and classifies user microphone input and transmits the analysis and classification to other participants in the jam. This data drives a synthesis module, which itself evolves in response to the user input.

The history of Networked Computer Music begins with the League of Automatic Music Composers[1], which, with additional members, later evolved into the Hub. Since the late 1970s, these technologist/musicians have been developing systems for sharing state between participants in an electronic ensemble[3]. Musical groups, including Sensorband[13] and Slub[17], have incorporated similar techniques into their musical performances. In recent years, new protocols have been formalized for transmitting musical data over

the Internet protocol stack, such as CNMAT’s Open Sound Control (OSC)[25] and Phil Burk’s TransJam[5].

Development of voice-controlled synthesizers can be traced back to Homer Dudley’s Vocoder[9]. The Vocoder analyzes voice input with a bank of bandpass filters and re-synthesizes an approximation of the original signal ([22] pp. 197-198). In the digital realm, Antares Audio Technologies[2] distributes the Kantos, a synth plugin controlled by the voice. Tod Machover’s Brain Opera[14] includes the Singing Tree[20], an interface for non-musicians to control sound and visuals with their voice.

Auracle combines network music making and voice control with the goal of engaging a broad public in “playing” with sound. This gives it a kinship with Tod Machover’s projects, such as the aforementioned Brain Opera and the Toy Symphony[15], and with Chris Brown’s Eternal Network Music[4]. Barbosa characterizes such projects in the context of networked musical systems as “Shared Sonic Environments”, which he describes as “a new class of emerging applications that explore the Internet’s distributed and shared nature [and] are addressed to broad audiences.” ([1] p. 58)

Despite the connections to these trends in computer music, Auracle’s ancestry is more closely related to analog music systems. Beginning in 1966, Max Neuhaus started exploring Shared Sonic Environments by fashioning networks from telephones and radio stations and altering vocal input with analog processing modules[18][8]. Several variations were realized between 1966 and 1977, when he began contemplating future directions for this type of work. Auracle is an evolution of Neuhaus’ Shared Sonic Environments into the digital realm.

2. ARCHITECTURAL OVERVIEW

The Auracle architecture is comprised broadly of six distinct components (Figure 1), realized through two commercial packages, JSyn and TransJam, and custom-built modules. Initial processing of vocal input is handled by the input analysis component. The results of the input analysis frames are combined and classified by the analysis coalescing component. Transmission of state from each client to its peers is the task of the network communication component. The state of each client in the system is received and

Architecture

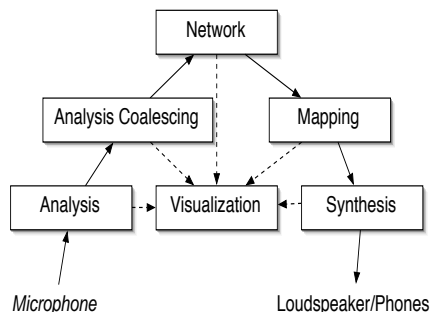


Figure 1: Diagram of Architecture

merged together to form a single model by the mapper component. This single model is in turn used to control sound production. Through it all, information describing the internal state of the Auracle system is presented to the user through a visualization component.

This paper discusses how each of these components are built and how they connect to one another. Emphasis is given to implementation decisions, code-performance issues, and architectural goals, such as designing for flexibility. In this article we do not discuss in depth the issues of human-computer interaction (sonic or visual), machine learning, sound design or aesthetics. We leave these topics for future publications.

3. INPUT ANALYSIS

Input analysis computes basic features of the signal over an analysis window. We analyze the incoming sound for voicedness/unvoicedness, fundamental frequency, the first two formant frequencies with their respective formant bandwidths, and root mean square (RMS) amplitude. (This analysis is predicated on the assumption that the incoming sound is vocal. We are not guaranteed the user is making vocal sounds, but we treat all input as if it were vocal.)

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. We limited our own DSP implementation to pure Java to avoid packaging and deploying JNI libraries for each targeted platform. We considered techniques based on linear prediction (LP), cepstrum (used in the Singing Tree [19]), FFT, and zero-crossing counts. Of these, we chose linear prediction. We felt it would be the easiest to implement in pure Java with acceptable performance and accuracy.

Raw sample data from the mic 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[21], the data is downsampled to 8192 kHz 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 ([21] pp. 158-161) to produce the feature value for that analysis frame.

Performance of the LP code was a major concern of ours. So, in this case, we violated Knuth’s maxim and prematurely optimized. The LP code is implemented in a slightly peculiar, non-object-oriented style. The goal was to minimize virtual and interface method lookup, and more importantly, to minimize object creation. Though such issues are often disregarded when writing Java, it should not be surprising that removing memory allocations in time-critical loops proved crucial to tuning this code. In the end, we were able to implement the signal analysis in pure Java with satisfactory performance.

4. ANALYSIS COALESCING

Analysis coalescing extracts higher-level features from the result of the input analysis stage. It segments analysis frames into gestures, computes features of those gestures, and classifies them.

The gesture segmentation serves multiple purposes: to chunk the input into meaningful units for the classifier and to improve the playability and efficiency of Auracle. We define a gesture as the data envelope from one block of silences more than 200 ms in length to another.

The inspiration for our classifier comes from the analysis of vocal signals for emotion. Though we are not interested in emotional classifications per se, the concrete results in this field make for a good start for any vocal input classification. In particular, we used the results of Klaus Scherer[24], Cowie et. al.[6], and Yacoub et. al.[23] as our foundation. These results guided both our selection of low-level features to extract (fundamental frequency, first and second formants, etc.) and the implementation of our feature classifier. The classifier performs Principal Components Analysis (PCA), implemented by a neural network, on features computed over the gesture[10].

Auracle transmits data to the network on a per-gesture basis. Though it would be more responsive were data transmitted more frequently, we had two problems with this. First, the network traffic was too great. Second, it was difficult for users to distinguish between their input and the response, as the system was responding while the input was in progress. Transmitting gestures reduces network traffic and gives users an opportunity to “have their say” before the response comes back.

5. NETWORK COMMUNICATION

As a user produces a gesture, data describing it is transmitted to every other participant. Network communication among the participants is provided by TransJam, a library and protocol for implementing distributed music applications. The TransJam server provides a way to create shared objects, acquire locks on those objects, and distribute notifications of changes to those objects.

A difficulty we encountered with TransJam is that the protocol is text-based. This makes transmitting floating point

numbers expensive. In Java, a floating point number occupies 4 bytes (or 8 bytes for double precision numbers)[12], but when represented as a string, it takes up as many bytes as there are digits (plus, up to two more for a negative sign and a decimal point). To save space, we truncate to five significant digits and encode the number as an integer.

Another source of concern is that there is a central sever. 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. Work may be duplicated by the clients, but we preferred that to adding load on the server. Our benchmarking shows that we can support 100 simultaneous users, each sending one gesture per second, with an average CPU load of 35% on our Apple Xserve (G4 1.33 GHz, 512 MB RAM).

6. PARTICIPANT MAPPING AND SOUND SYNTHESIS

Participant mapping merges the state of all clients into a single model that is used to drive synthesis. An example would be using the amplitude, fundamental, and formant envelopes of a gesture to control parameters in a physical model. The sound is synthesized by a JSyn patch.

To enable easier experimentation with sound synthesis techniques, we built the capability to transmit control data from Auracle over OSC. Though the final version of any patch we build will be implemented in JSyn, it is helpful to experiment with patches in Max/MSP[7] and SuperCollider[16].

7. VISUALIZATION

Auracle’s user interface is quite different from a typical interactive application. Auracle’s primary means of interaction is not the keyboard and mouse, but rather the voice. Thus, the GUI for Auracle is not a typical WIMP (Windows, Icons, Menus and Pointers) UI, focused on buttons, input widgets, etc. Rather than accepting input, the GUI’s focus is providing feedback about the internal activity of the system and aiding the user in correlating his/her input with the resulting output.

The centerpiece of the UI is the large view that occupies the majority of the left of the screen. It illustrates the last five gestures for every user participating in an ensemble (in the screenshot, there is only one user). The bottom third displays the RMS amplitude for the gesture; the top two-thirds display the track of the fundamental frequency and first two formants over the gesture. The right side of the screen presents information relating to the local user. The top right view plots the amplitude, fundamental frequency, and formants one and two as the user provides input into the microphone. When a gesture is detected, it is delineated by a line extending over its duration. The right middle view maps changes in the formants against changes in the fundamental frequency. The bottom right view provides information about the internals of the system, such as CPU utilization and the number of bytes of data sent and received.

8. ARCHITECTURAL CONSIDERATIONS

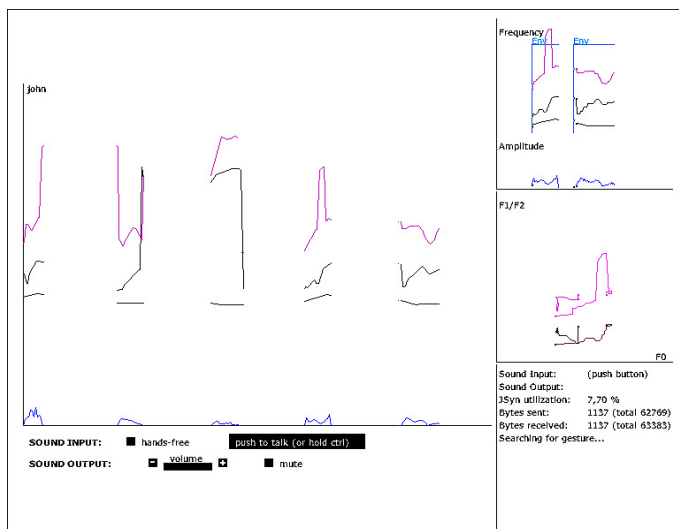


Figure 2: The Auracle User Interface (colors inverted)

We have tried to keep the different categories of components as loosely coupled as feasible, so we can easily plug in alternative implementations for any component. To this end, we employ the Observer Pattern[11], Interfaces, and occasionally Java Reflection and hand-input metadata. Objects in one component category rarely reference directly objects in another. Instead, they either register themselves as observers of the other object, or they reference only the interface. For example, the analysis component does not directly communicate to the user interface or the analysis coalescing components. Instead, those components register themselves as observers of the analysis. This lets us build up layers of analysis processing and visualization, without having to alter existing classes.

Another benefit of the indirection is that the desired implementations of each object category can be specified in a property file. When Auracle is launched, there is an initial boot-strapping period, during which it reads a configuration file to determine which specific classes to use and instantiates them through Java Reflection. We have also built a GUI tool that can reconfigure Auracle at runtime. This lets us experiment with a variety of different implementations of analysis, visualization, mapping and synthesis.

9. CONCLUSIONS AND FUTURE WORK

Auracle is a new instrument in the early phases of its development. The instrument is intended to be enjoyed by a non-specialized audience; it analyzes vocal sounds and transmits them over the Internet to control a synthesizer. We have a foundation which performs voice analysis and feature extraction and efficiently transmits data over a network. This foundation has been tested in numerous transatlantic jams involving up to six people. Our next challenge is to design a satisfying user experience with interesting and controllable sounds.

One of the goals we set for Auracle is that the instrument be “transparent”. That is, the user should be able to per-

ceive her/his effect on the overall sound output, even in a group situation. Another goal for Auracle is that its sound evolves, both in the time scale of a jam, and from week to week. Auracle produces quite a bit of data to describe and classify user input: much of our current and future work is centered on harnessing this data to generate an engaging user experience that meets these goals. Part of this effort is focused on sound design and part of it is focused on data visualization. We expect to have more results to report in the future.

10. ACKNOWLEDGMENTS

Thanks to Stephen T. Pope, who commented on earlier revisions of this paper. The Auracle Project is a production of Akademie Schloss Solitude with financial support from the Landesstiftung Baden-Württemberg. We express our gratitude for their generous support.

11. ADDITIONAL AUTHORS

Additional authors: David Birchfield (Arizona State Univ., email: dbirchfield@asu.edu), Phil Burk (SoftSynth, contact: <http://www.softsynth.com>), Max Neuhaus (email: neuhaus@compuserve.com)

12. REFERENCES

- [1] Álvaro Barbosa. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, 13:53–59, 2003.
- [2] Antares audio technologies: Antares kantos, 2004. <http://www.antarestech.com/products/kantos.html>.
- [3] J. Bischoff, R. Gold, and J. Horton. Microcomputer network music. In C. Roads and J. Strawn, editors, *Foundations of Computer Music*. MIT Press, Cambridge, Massachusetts, 1985.
- [4] C. Brown. Eternal network music, 1999. http://www.mills.edu/LIFE/CCM/Eternal_Network_Music.html.
- [5] P. Burk. Jammin' on the web - a new client/server architecture for multi-user musical performance. In *Proceedings of the ICMC*, 2000.
- [6] R. Cowie, E. Douglas-Cowie, N. Tsapatsoulis, G. Votsis, S. Kollias, W. Fellenz, and J. G. Taylor. Emotion recognition and human-computer interaction. *IEEE Signal Processing Magazine*, January 2001:32–80, 2001.
- [7] Cycling '74: Max/msp, 2004. <http://www.cycling74.com/>.
- [8] E. Decker and P. Weibel. *Vom Verschwinden der Ferne. Telekommunikation und Kunst*. Dumont, Köln, 1990.
- [9] H. Dudley. Remaking speech. *Journal of the Acoustical Society of America*, 11(2):167–177, 1939.
- [10] J. Freeman, C. Ramakrishnan, K. Varnik, M. Neuhaus, P. Burk, and D. Birchfield. Adaptive high-level classification of vocal gestures within an interactive aural architecture. Submitted to ICMC, 2004.
- [11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.
- [12] B. Joy, G. Steele, J. Gosling, and G. Bracha. *Java™ Language Specification*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2nd edition, 2000.
- [13] Z. Karkowski, A. Tanaka, and E. van der Heide. Sensorband, 2004. <http://www.sensorband.com/>.
- [14] T. Machover. Brain opera, 2000. <http://brainop.media.mit.edu/>.
- [15] T. Machover. Toy symphony, 2002. <http://www.toysymphony.net/>.
- [16] J. McCartney. Supercollider: A new real-time synthesis language. In *Proceedings of the ICMC*, 1996.
- [17] A. McLean and A. Ward. Slub, 2004. <http://www.slub.org/>.
- [18] M. Neuhaus. The broadcast works and audium, 1994. <http://kunstradio.at/ZEITGLEICH/CATALOG/ENGLISH/neuhaus2-e.html>.
- [19] W. Oliver. The singing tree: A novel interactive musical interface, 1997. http://feynman.stanford.edu/people/Oliver_www/singhtml/main.html.
- [20] W. Oliver, J. Yu, and E. Metois. The singing tree: Design of an interactive musical interface. In *Proceedings of the Symposium on Designing Interactive Systems*, pages 261–264, 1997.
- [21] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.
- [22] C. Roads. *The Computer Music Tutorial*. MIT Press, Cambridge, Massachusetts, 1996.
- [23] X. L. S. Yacoub, S. Simske and J. Burns. Recognition of emotions in interactive voice response systems, 2003. HPL-2003-136. <http://www.hpl.hp.com/techreports/2003/HPL-2003-136.html>.
- [24] K. R. Scherer. Vocal correlates of emotional arousal. In H. Wagner and A. Manstead, editors, *Handbook of Social Psychophysiology*, chapter 7. Wiley, Chichester, New York, 1989.
- [25] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the ICMC*, 1997.