# Recent compositions and performance instruments realized in Java Music Specification Language

Nick Didkovsky
didkovn@mail.rockefeller.edu
www.algomusic.com

## Abstract

*Java Music Specification Language (JMSL, Didkovsky, Burk) is a Java API for algorithmic music composition and performance. This paper presents an overview of three recent pieces realized in JMSL. The works included here demonstrate a wide range of what JMSL offers the composer and performer, including real-time interactive score generation (Zero Waste for sight reading pianist and computer), physical instrument modeling for compositional purposes (Tube Mouth Bow String for string quartet and live electronics), and the creation of intelligent, real-time performance instruments (Virtual Rhythmicon, based on the Henry Cowell's Rhythmicon).*

## 1 Introduction

JMSL has been used to create a wide variety of new computer music works. Zero Waste, for sight reading pianist and computer, challenges the performer to sight-read a score in common music notation which is generated in real-time. The Virtual Rhythmicon models an early electronic music instrument designed by Henry Cowell, and runs as an applet in a web browser. Tube Mouth Bow String, for string quartet and live electronics, uses JMSL to model the sonics of the ensemble and algorithmically generate a final notated score. JMSL is available at www.algomusic.com.

## 2 Zero Waste, for sight reading pianist and computer generated score

Zero Waste uses JMSL to generate a score in common music notation which is sight-read by the live performer. There is no computer generation of audio; the audience hears only a live grand piano and sees a large projection of the score that is being read by the performer. A piano capable of sending MIDI is required, such as the Yamaha Diskclavier, so that the computer can receive real-time performance data. As the performer sight reads, JMSL transcribes and notates. This continuous loop of performance and transcription generates a score in real-time. Zero Waste was composed by the author for pianist Kathleen Supové.

### 2.1 Description

Zero Waste begins by displaying two measures of stochastically generated music. The performer views the score on a laptop computer which is situated before her on the piano. As soon as she begins playing, JMSL begins recording her MIDI performance data. As soon as the performance of measures 1 and 2 is complete, JMSL transcribes the performance and displays it as measures 3 and 4. The performer continues sight reading, now performing measures 3 and 4. At the end of these two measures, JMSL once again displays a transcription of her performance, displayed as measures 5 and 6. This process of sight-reading and transcription continues for the twelve-minute duration of the piece.

Zero Waste amplifies the resonances of a system which is characterized by the limitations of human performance and common music notation. If the performer were perfect, and if music transcription and notation were both theoretically and practically perfect, then Zero Waste would consist of identical repetitions of the first two measures. Of course, no sight reader is perfect, and notation must strike a balance between readability and absolute accuracy, so each new pair of measures diverges and evolves a bit more from the last.

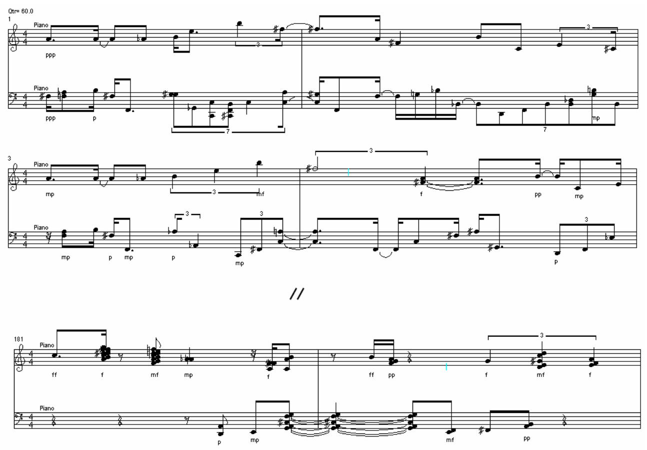


Figure 1 Zero Waste begins with two measures of algorithmically generated music (top). Measures 3 and 4 (middle) are transcriptions of the live performance of measures 1 and 2. The last two measures of the piece are displayed at the bottom of the figure.

## 2.2 Some observations

Systematic tendencies were consistently observed over multiple performances. The transcribed rhythm deviates almost immediately from the original (see figure 1). As performer Kathleen Supove described it, the expressive qualities of interpretation suddenly took on major notational and compositional consequences. Also evident immediately is that pitches which fall outside reasonable reach are lost.

One curious tendency is the gradual appearance of rests at the beginning and end of each two-measure system. This was due to the slight performance hesitation caused by not being able to read ahead to the next two measures, since they can not be displayed until the performance of the current two measures is completed. This minute pause will be notated as a rest. The performer's rush to catch up might finish the material a little early, resulting in a rest appearing at the end. Once notated, the performer is of course obliged to perform these rests; they become an essential part of the piece.

Countering this effect is the occasional intent of the performer to slow down for expressive purposes. This might result in the notes at the end of one two-measure frame spilling over into the beginning of the next two-measure frame. The next transcription will then begin with the notes left over from the previous transcription, and the head of the measure begins to fill in again.

Another pair of contrary forces became evident: the formation of chord clusters and their dissolution. Chords form when the timestamps of two notes are attracted to the same quantized time point. Once notated as a chord, the performer is obliged to continue performing the chord, so there seems no escape back to melody. However, the performer's hands are too small to perform all the notes contained in very tall chord clusters. Her performance strategy was to split these chords into multiple events. These sub-events might be transcribed as smaller chords and monophonic notes. In this way, the clumping tendency of the system has a natural counter-effect.

## 2.3 Summary

Some audience members likened Zero Waste to the game of "telephone", where a story is passed through a sequence of people, each of whom hears the story from one neighbor and retells it to the next. Like the telephone game, Zero Waste behaves as an information filter. The focus is on the process of change, and what it reveals about the dynamics of the system. JMSL's notator, transcriber, and MIDI support provided the author with an elegant real-time API with which to realize this work.

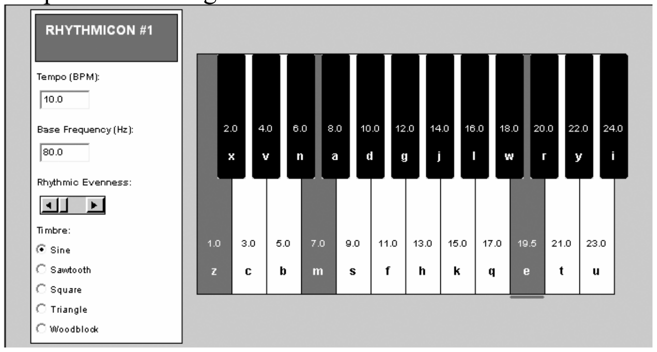
# 3 The Virtual Rhythmicon

The Rhythmicon (Cowell) is an early electronic music instrument designed by Henry Cowell and built by Leon Theremin in 1930. It has seventeen keys. Each key corresponds to a partial in the harmonic series, and plays a pitch with a steady, repeated pulse. The first key plays a low fundamental frequency at a slow tempo. The second key plays twice the frequency at twice the tempo, and so on. By holding down keys simultaneously, complex polyrhythms can be realized.

In 2003 Minnesota Public Radio commissioned the author to create a software realization of Cowell's Rhythmicon, to be deployed on the web. JMSL was chosen as an appropriate technology because of its flexible scheduler and its web capabilities. JSyn (Java Synthesizer, Burk) was chosen as the audio engine, due its ability to perform real-time CD quality audio in a web browser and its Java API. In May of 2003, The Virtual Rhythmicon was launched on MPR's Music Mavericks website ( http://www.musicmavericks.org/rhythmicon/ ).

## 3.1 Design

Essential design features in Cowell's original Rhythmicon were maintained: each key corresponds to a partial, multiple keys can be performed simultaneously, the fundamental frequency can be changed, and the overall tempo can be changed.


RHYTHMICON #1

Tempo (BPM): 10.0

Base Frequency (Hz): 80.0

Rhythmic Evenness:

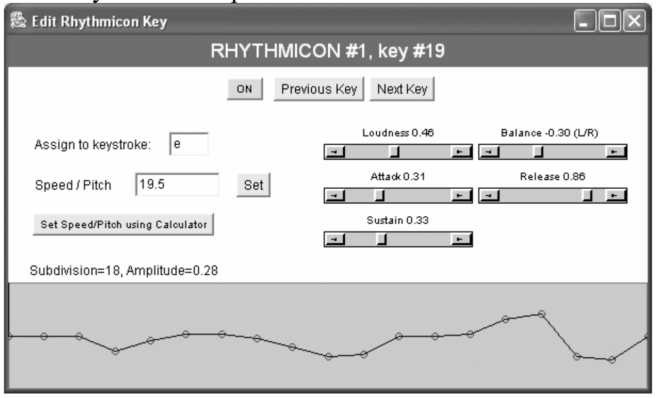
Timbre: [✓ Sine] Sawtooth Square Triangle Woodblock

Figure 2. The Virtual Rhythmicon control panel. Three keys are shown to be sounding here, corresponding to the fundamental, the seventh, and the 19.5th partials.

The Virtual Rhythmicon adds a number of innovations to the original design. The user can play up to four Rhythmicons at a time. This affords the possibility of very complex and beautiful pitch and time relationships between Rhythmicons.

The user can change the timbre of any Rhythmicon by choosing from various synthesis patches. The evenness of the rhythm can also be varied, ranging from strict regularity to very irregular. This is implemented with a Myhill Distribution (Ames), which distributes events over time using an entry delay mechanism, and provides control over the evenness of this distribution.

The Virtual Rhythmicon offers extensive control over every key. The composer can assign any partial value to a key, including non-integer partials. The overall amplitude and stereo panning of the key can also be specified and changed over time. Attack and release rates of the synthesized notes can be controlled. A scaler specifies sustain length as a fraction of a pulse's duration, corresponding roughly to notions of staccato and legato. Finally, any Virtual Rhythmicon key can be assigned to any computer key. The user can construct groupings of Rhythmicon keys to be triggered by the same key-press, creating complex behaviors and rhythmic relationships that are easily recalled in performance.



RHYTHMICON #1, key #19

Assign to keystroke: e

Speed / Pitch 19.5

Loudness 0.46

Balance -0.30 (L/R)

Attack 0.31

Release 0.86

Sustain 0.33

Subdivision=18, Amplitude=0.28

Figure 3. The Rhythmicon Key Editor allows the composer to control the attributes of a key.

## 3.2 Community composition

Minnesota Public Radio's commitment to public access and its strong sense of community required that users be able to record and upload performances to a public server. Archived performances ought to be publicly available for play-back. These client/server features were included in the Virtual Rhythmicon.

When the user starts recording, all GUI events and corresponding timestamps are captured. The "Command Design Pattern" (Gamma et al) is used to implement this feature, where user events are encapsulated as objects and stored. As opposed to simply capturing note-level musical output, Virtual Rhythmicon play-back controls the Rhythmicon itself with "ghost hands". The user sees keys turn on and off, partial values change, amplitude contours altered, etc. This also enables the user to play along with the performance, record a composite, and upload a derivative work. Uploads, downloads, and searches of Virtual Rhythmicon performances are managed by a MySQL database and PHP scripts residing on the server. The user can email a performance to a friend: the recipient receives an email message with a link to the composition.

## 3.3 Summary

JMSL provides an elegant scheduling model appropriate for the Virtual Rhythmicon. MusicJobs were used to implement the individual key pulses. Multiple MusicJobs were put into a ParallelCollection to ensure that they would start and stop together, following a master clock. JMSL's tight integration with JSyn made it straightforward to implement a unified Instrument interface with common control over envelope, pan, sustain, and amplitude, and populate it with various SynthNotes that realize different timbres.

JMSL's Java foundation allowed the author to leverage off of popular object oriented design techniques such as the Command Design Pattern. Java's ability to run in a web browser and send data to and from a server made it straightforward to deploy the Virtual Rhythmicon on the web, where thousands of users have performed it and accessed its growing public archive.

## 4 Tube Mouth Bow String, for string quartet and live electronics

Tube Mouth Bow String is an algorithmically generated composition for string quartet and live electronics. The role of the software is to model the ensemble and generate a score. There is no computer component to the performance; the live electronics are commercially available devices. Four talk-boxes are used to modulate the sound of the string quartet with vowels mouthed by the performers. Four foot pedals are used to create harmonic glissandi.

Tube Mouth Bow String was algorithmically generated, transcribed to common music notation using JMSL's Transcriber class, notated in JScore, and exported from JScore to San Andreas Press's Score for final publication. Tube Mouth Bow String was composed by the author for The Sirius String Quartet, and was supported by a grant from Meet the Composer's Commissioning Music/USA program.

## 4.1 Modeling the ensemble

Each live performer's instrument has a contact microphone whose output is connected to a harmonizer pedal. The pedal harmonizes the input signal ranging from an octave below pitch in the heel position, to an octave above in the toe position. This harmonization smoothly glisses as the pedal is moved. The output of the harmonizer pedal is connected to a talkbox which contains a small amplifier and speaker whose output is piped through a polyvinyl tube terminating in the performer's mouth. As the performer mutely mouths various vowels, the signal is filtered by the shape of the oral cavity. This filtered sound is amplified with a vocal microphone and public address system.

The composer wanted to play with these resources in a flexible way during the composition process. JMSL and JSyn were extremely useful tools here, as JSyn was able to mimic the sound of the quartet vividly, while JMSL managed high level compositional form.

In order to model the ensemble in software, three components were needed: synthesis of a bowed string, a harmonizer patch, and a vowelization filter. These were modeled in JSyn and encapsulated in higher level JMSL Instrument classes.

Bowed strings were modeled in a straightforward way using JMSL's TransposingSampleSustainingInstrument class. A convincing virtual string quartet was assembled by loading four such instruments with commercially available bowed string samples.

The harmonizer pedal was modeled as a JMSL Instrument implementing the PlayLurker interface. PlayLurkers can be notified of performance data being played by other objects. For this piece, the harmonizing instrument receives notification whenever a bowed string plays a note. It reads the pitch value contained in the data, and calculates the frequency of the harmony. The instrument sounds the harmony with a sawtooth oscillator which achieves a convincing harmonization when mixed with the original bowed string sample. This was more efficient and reliable than implementing a true pitch tracker and a pitch shifter.

A vowelization filter was designed in JSyn by James Forrest using multiple bandpass filters (Dodge). This filter was encapsulated in a JMSL instrument.

### 4.2 Notation

Each musician reads three staves: the top staff notates the vowels, the middle staff notates bowed strings, and the bottom staff notates pedal positions (see figure 4).

The full sweep of the harmonizer pedal is broken down into seven distinct positions: heel, toe, and middle position, plus four intermediate positions: two between middle and toe, and two between middle and heel. These locations were notated on seven distinct staff positions.

The difficulty in performing Tube Mouth Bow String derives from the complete independence between the three parts that each player must execute. Over time, the players became facile with this independence.

### 4.3 Comments on form

Over the course of twelve minutes, the pitches of the bowed strings follow a statistically interpolated trajectory from low harmonic complexity (unison pitch) to high harmonic complexity, then back to unison, resulting in an ending rich with cadence-like gestures. Event density of talk-box material begins with long sustained vowels and ends with dense rhythmic vowel cycles. Pedal moves are slow and sparse at beginning and end of the work, reaching peak density in the middle.

The piece specifies rhythmic correlations between pairs of players. The two violins' pedal glissandi are in rhythmic unison but gliss in contrary motion. Viola and 'cello are similarly paired. Violin 1 and viola share the same vowel rhythms; violin 2 and 'cello form the other vowel pair. These pairings provide internal coherence, which along with the large scale shaping of the piece, provides the listener with a number of different focal points of attention.

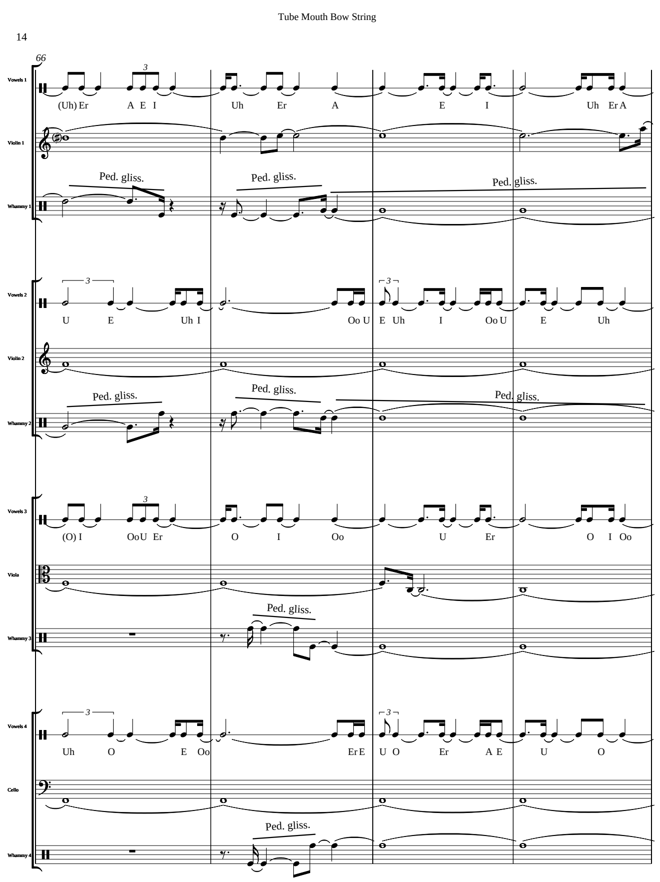


Figure 4. Vowel rhythms, bowed strings, and pedal positions are notated on separate staves.

## 5 Conclusion

The three pieces described in this paper have very different goals and aesthetic directions, including real time score generation, digital online performance, instrument modeling, and algorithmic composition. JMSL provides an API that is fluid enough to embrace the very different musical worlds defined by these pieces.

## 6 References

Ames, C., (1994). "A Catalog of Statistical Distributions: Techniques for Transforming Random, Determinate and Chaotic Sequences," *Leonardo Music Journal*, Vol. 1, No. 1, pp. 55-70

Burk, P.L., (1998). "JSyn - A Real-time Synthesis API for Java." *Proceedings of the International Computer Music Conference*. International Computer Music Association, pp. 252-255.

Cowell, H., "Preface". *Quartet Romantic*; *Quartet Euphometric*. New York: C. F. Peters, 1974

Didkovsky, N., Burk, P.L., (2001). "Java Music Specification Language, an introduction and overview" *Proceedings of the International Computer Music Conference*. International Computer Music Association, pp. 123-126.

Dodge, C., Jerse, T., (1985). *Computer Music. Synthesis, Composition, and Performance*. Schirmer Books, pp. 204-205.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1995). *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, pp.233-242