On-line Additive Synthesis by Interfacing SOM-A to Java Sound™

Aluizio Arcela & Rodolfo Bezerra Batista
University of Brasilia
Computer Science Dept.
{arcela, rbb}@cic.unb.br

Abstract

It's described a new approach for the SOM-A sound generating system which has new features, including an editing tool for assisted musical composition, a graphical interface for visualizing orthogonal timbres, an instrument analyser capable of displaying the envelope of every spectral component, a normalizer device for scaling automatically the amplitude to fit exactly a predefined range, and some realtime sound rendering facilities. Once SOM-A is now implemented as a public set of Java classes, populations of additive synthesis objects can be defined inside hyper documents, virtual reality worlds, as well as in general purpose Internet programs whether applets or local applications. In order to provide an effective means for instant audition to the user, the Java Sound API has been added to the SOM-A signal synthesis domain as its sound rendering unit.

INTRODUCING THE NEW SOM-A

Since 1986, when it was created, the SOM-A[3] language has been submited to a number of implementations, starting with that in LISP[9,10] which was entirely inspired on the topology of the algorithmic instruments defined by the time-trees[2,4]. Thereafter, C and C++ implementations[6,7] for both Unix and DOS platforms have been done, mainly because a better computing performance was needed for additive orchestras having more than one thousand harmonic units. Now, as a result of such continuous work of implementation, an approach supported by the Java language is presented here which extends significantly the SOM-A's musical action space. The most meaningful difference between the

implementation in Java to the old ones [5] is that new SOM-A, besides interpreting musical scores with additive instruments exactly in the same way, it serves as a basic API for the development of programs for sound synthesis as well as for musical composition. A SOM-A object is an encapsulation of methods related to additive synthesis, PCM sample generation, realtime MIDI message posting, audio rendering, and sound storage if desired. The data living inside an object is basically a spectral chart. Therefore, when a SOM-A object is created inside a program, one has the capability of additive synthesis for the production of sound samples on the fly, without using any sound file at all in the data transmission from the computer to the audio device, whether in a local or network application. The final audition of the PCM signal then generated is now possible in the same instance of the synthesis process thanks to a beta-version sound programming interface —called Java Sound — that was released by Sun Microsystems, Inc. a couple of months ago [12] for general purpose sound applications. When the computing power is not enough to produce the realtime PCM additive synthesis required by the chart, the Java Sound's MIDI direct player can be used as an alternate realtime audio rendering in place of the actual additive production. In Java Sound, a software-based MIDI engine is available so that, even if the user has no MIDI device connected to his computer but only D/A converters, he will get a good-quality audio signal from such an engine in despite of the unavoidable loss in the timbre information. However, if sinusoidal midibanks are defined, it is possible to group a set of concurrent messages so as to produce a satisfactory MIDI approximation for those additive instruments having less than 32 partials [8].

1.1 The Object SOM-A

JSOM-A is an API defined by a set of four packages covering all classes needed for building musical application programs where additive synthesis is required. Because SOM-A is a concise sound synthesis language [1,8] having just 5 commands and its additive synthesis capability is able to handle harmonic spaces of any size, the use of SOM-A objects inside musical and other sound systems can be very profitable, specially in on-line sound processing and musical services where a considerable economy in the required amount of transmited data will take place. The ideal situation for SOM-A is reached when its functionality can include the premise that no sound file needs to be generated but everything is produced in realtime, whether as a true sample-by-sample interpretation or

as a MIDI synthesis approximation. The idea behind a SOM-A object can be ilustrated by the equation: *score* = *sound*. That is, from the program's point of view there is no difference between the sound representation and the sound itself.

1.1.1 The Runtime System (Package I: br.unb.cic.SOM_A)

Classes	
EXE	Holds the notes of a SOM-A chart.
INS	Holds instruments used by a Som_Achart object.
JSom_A	Interface
Note	Defines a single note inside a SOM_A chart.
Som_A	Is a command-line interpreter for SOM-A charts.
Som_AChart	Is a class that holds every parameter found in a SOM-A chart.
Som_AInterpreter	Is responsible for interpreting a SOM-A chart.

1.1.2 Monitoring the Synthesis Process (Package II: br.unb.cic.SOM_A.monitor)

GUI's frames for displaying some spectral behaviors along the synthesis process, as waveform, envelopes, and shapes for orthogonal timbres.

Classes	
Orthogonal	Is an AWT component that displays an orthogonal composition of an Orthogonal SOM_A Instrument.
Envelope	Is an AWT component that displays the shape of uH envelopes.
Waveform	Is an AWT component that displays a RAW File's waveform.

1.1.3 The SOM-A Interpreter (Package III: br.unb.cic.SOM_A.interpreter)

PCM and MIDI sound generation for interpreting a SOM-A chart is defined inside this package. PCM samples constitutes the usual desired result whereas the MIDI synthesis plays an auxiliary function either in the preview for timbres and in rendering the overall signal when the sampled sound interpretatiion is not able to produce a realtime output.

Classes	
MIDIInterpreter	Is responsible for finding a MIDI approximation for the additive synthesis required by the chart.

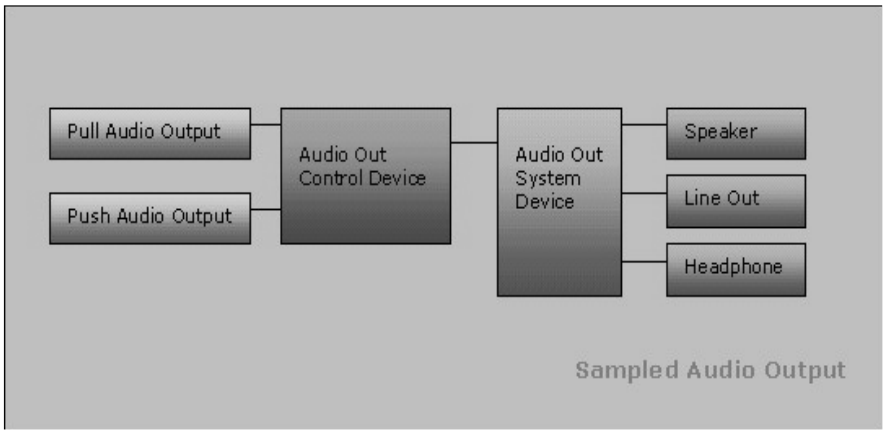
PCMInterpreter	Is responsible for rendering a Som- A chart and generating byte samples representing 16- bit stereo samples.

1.1.4 SOM-A Fundamental Devices (Package IV: br.unb.cic.SOM_A.synth)

The fundamental algorithms [5], that is, the H-unit, the instrument and its small parts, i.e., the oscillator and the envelope generator, are all implemented in this package.

Classes	
CircuitAdapter	Provides an interface for adapting circuits that cascade their clock.
Envelope	Defines an envelope generator whose value is used in the Oscillator.
H_Unit	Is the minimum musical instrument implementation.
Instrument	Is the minimum additive musical instrument implementation.
LinTable	Provides a look-up table that interpolates values between indexes.
LookUpTable	Is the root all look-up tables used inside some circuits.
Oscillator	Defines an oscillator object that generates a sin flavored waveform with defined frequency, envelope generator and sampling rate.
SinTable	Provides sin(x) function values throught a look-up table and interpolation.

ABOUT JAVA SOUND



Java Sound [12,13] is a set of classes having support for both digital audio streaming and MIDI synthesis. There are two major modules of functionality which are provided in separate packages: `javax.media.sound.sampled` used for capture, mixing, and playback of sampled audio, and `javax.media.sound.midi` for MIDI synthesis, sequencing, and event transport.

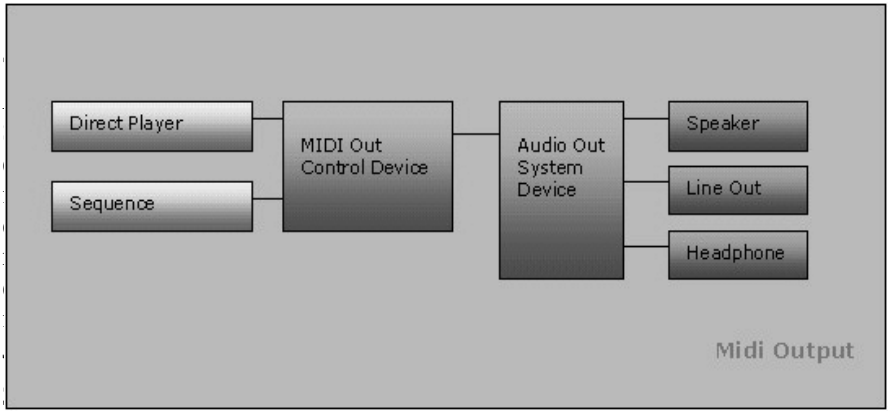
2.1 Sampled Audio

The Audio Out Control Device is a device used for rendering sampled audio. It supports the generation of audio output channels for both pull and the push data transport mechanisms. The sound to be rendered need not to be a file. Data structures inside the program which carry sound samples out can be moved to this output device at a speed according to the user- supplied sampling rate. Basically, this section of Java Sound is a brigde conecting the samples produced by the program to the D/A conveters in a very effective way.

2.2 MIDI

Discrete MIDI messages must be sent to the Direct Player channel whereas MIDI files must be sent to the Sequence channel. The MIDI Out Control Device is a true MIDI synthesizer which generates sound and provides methods for manipulating soundbanks and instruments. In addition, a synthesizer may support a set of global non- MIDI controls such as gain and pan. Finally, it provides access to a set of MIDI channels through which sound is actually produced.

MIDI channels support methods representing the common MIDI voice messages such as "note on" and "control change."



A text file having two section: one for the additive orchestra and the other for the score. Usually this file is to be read and interpreted by the SOM-A main loop in order to produce sound samples to be played directly by the audio device or stored in a digital sound file. However, this input file may be changed in many ways: (1) the user may edit the file, (2) the parameter setup (see below) has been changed by the user action on the graphical interface, (3) SOM-A runtime system detected overflow, that is, the maximum value of the overall waveform is greatest than the maximum number that can be represented by the length in bytes of the sample. Is this case, SOM-A changes de norm parameter in the VAL command.

3.2 Parser

The functional unit responsible for checking the spectral chart syntax and for translating the LISP-like syntax of SOM-A charts into a form more suitable for reading in Java. An output of numbered errors is provided in case of non well formed charts.

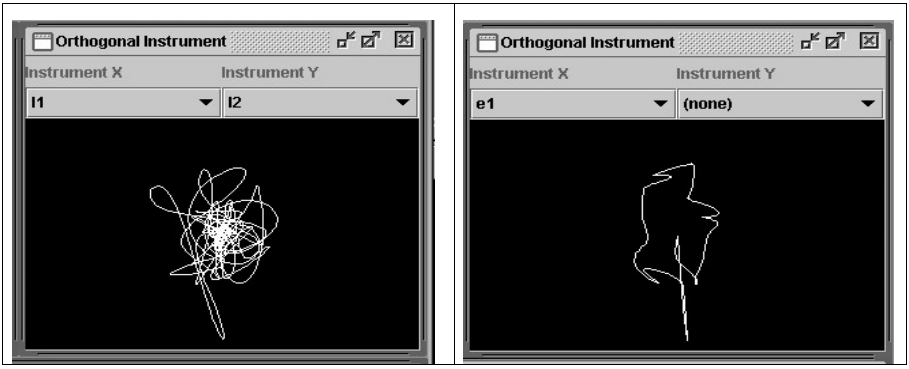
3.3 Digital Sound File

If required, as a new segment of the signal is computed it is immediatly appended to the amount of samples in the binary file. Almost all common digital sound formats are available, as aiff, wav, au, etc..

3.4 Timbre Viewer

It is a very useful device for choosing timbres by hearing their spectral quality combined to the appreciation if their visual representation. Its main application seems to be in musical

composition as a very large number of additive instruments is available in spectral charts computed by the time-trees. These computed instruments may be extracted from their native chart to figure out in another one which must be edited by the user. If necessary, the H-units can have their envelopes changed directly from a mouse action on the breakpoints of the envelope's graphical representation. Below, some orthogonal timbres selected from different charts are shown.



3.5 Instrument Analyser

For a selected instrument, all of its H-unit parameters are displayed, being the frequency order and the inicital phase angle numerically displayed while the envelope is graphically represented. At any time the instrument may be requested by the user to produce

a sound at a particular base frequency for a particular duration. Sub-instruments [1] can be also enabled to sound.

3.6 Editor

Capable of building a complete chart by copying selected instruments from different charts as well as score segments from a spectral sequence database. Very meaningful results can be obtained and many musical ideas may be implemented in a very short work time. Results are possible to be heard and reedited.

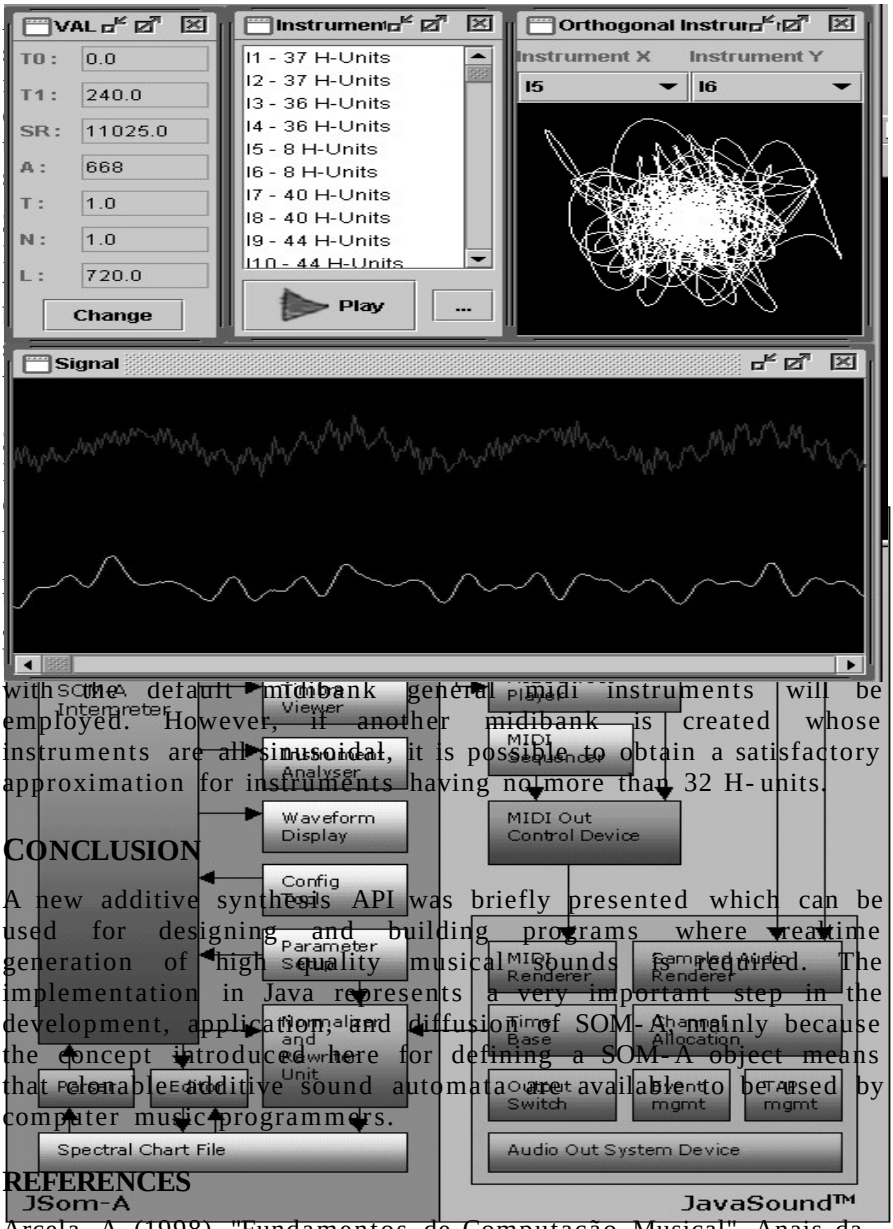
3.7 Normalizer and Rewritter Unit

Capable of adjusting automatically the norm parameter of the VAL command after running all the chart in order to find out the amount of overflow so as to determine the normalization factor. In addition, the Java Sound tempo control (see below) may become a feedback to the normalizer for automatic changing in the VAL tempo parameter.

3.8 Parameter Setup

All the VAL parameters (*VAL* t_0 t_1 F A T N L) namely, starting time, final time, tempo, transposition, norm, and envelope length can all be changed by the user directly on the screen.

3.9 Configuration Tool



with the default midibank general midi instruments will be employed. However, if another midibank is created whose instruments are all sinusoidal, it is possible to obtain a satisfactory approximation for instruments having no more than 32 H-units.

CONCLUSION

A new additive synthesis API was briefly presented which can be used for designing and building programs where realtime generation of high quality musical sounds is required. The implementation in Java represents a very important step in the development, application, and diffusion of SOM-A, mainly because the concept introduced here for defining a SOM-A object means that cloneable additive sound automata are available to be used by computer music programmers.

REFERENCES

Arcela, A, (1998). "Fundamentos de Computação Musical", Anais da VI Escola de Informática da SBC Regional Sul, Curitiba (also in http://www.cic.unb.br/tutores/fcm/fcm.html).

Arcela, A, (1996). "As árvores de tempos", On-line publishing: http://www.lpe.cic.unb.br/teoria/teoria.html, Brasília.

Arcela, A., (1994)."A linguagem SOM-A para síntese aditiva", Proceedings of the First Brazilian Symposium on Computer Music", Caxambu, MG.

Arcela, A, (1986). "Time-trees: the inner organization of intervals", Proceedings of the International Computer Music Conference, The Hague.

Castro, R.R.F., (1998). "Primeira implementação de SOM-A em Java", Technical Report, CNPq/RHAE 610.069/96-9, Projeto W3CIC, Brasília.

Castro, R.R.F., (1994). "Implementação de SOM-A em Borland C++ para o ambiente MS-Windows", Technical Report LPE-9402, University of Brasilia.

Meireles, A., Gioia, O., Castro, R., (1993). "SOM-A em C para Sun SparcStation, Technical Report LPE-9303, University of Brasilia.

Miranda, E. R., (1998). Computer Sound Synthesis for the Electronic Musician, Oxford: Focal Press.

Nogueira Filho, V., (1998). "Síntese aditiva modular -- uma máquina espectral programável", MSc. Dissertation, University of Brasilia.

Ramalho, G.L., (1991). "SOM-A em Sun Common LISP para o ambiente OpenWindows", Technical Report LPE-9105, University of Brasilia.

Sun Microsystems, Inc., (1998). JavaSound API: A Technical Overview, http://java.sun.com:8081/javaone/javaone98/sessions/T601

Sun Microsystems, Inc., (1999). JavaSound API Specification, 10 March. http://java.sun.com:8081/products/java-media/sound/forDevelopers/javasound085/index.html