

**Realização**  
Departamento de Ciência da Computação  
Universidade Federal de Minas Gerais

**Promoção**  
Sociedade Brasileira de Computação

**Patrocínio**  
CNPq  
FAPEMIG  
FINEP

**Apoio**  
CAPES  
FAPERGS  
FAPESP  
TELEMIG  
UFMG

XIV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

CA869.3.C  
S612a  
1º

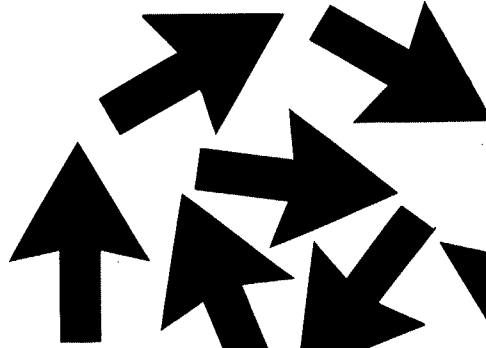
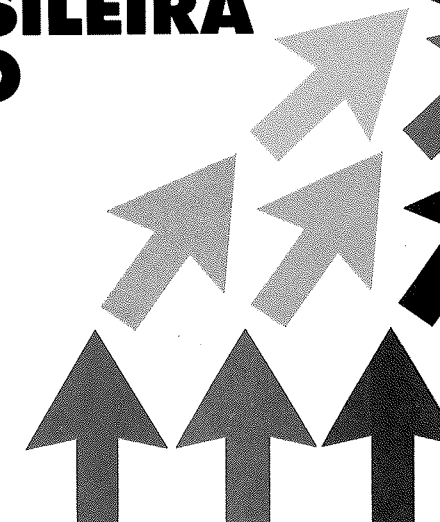


# XIV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

## ANAIS

I Simpósio Brasileiro de  
Computação e Música

Caxambu - MG  
Brasil  
1994



17988

DEDALUS - Acervo - IME



31000051659

**I SIMPÓSIO BRASILEIRO  
DE COMPUTAÇÃO E MÚSICA** 1290324

Caxambu — MG

01 a 05 de agosto de 1994

**ANAIS**

Editado por  
Maurício Alves Loureiro

Realização  
Escola de Música  
Departamento de Ciência da Computação  
Universidade Federal de Minas Gerais

Promoção  
Sociedade Brasileira de Computação

V\_ 17  
lo  
se  
ia  
i-  
ia  
e  
e  
)  
a  
)  
a  
i  
o  
e

Cópias adicionais:

Universidade Federal de Minas Gerais  
Escola de Música  
30.130-005 Belo Horizonte, Brasil  
Telefone: (031) 222 2357  
e-mail: mauricio@dcc.ufmg.br

UNIVERSIDADE DE SÃO PAULO INSTITUTO DE MATEMÁTICA E ESTATÍSTICA BIBLIOTECA	
DATA 11.09.03	N.º DE CHAMADA 018693C
N.º DE TOMBO 51229	REGISTRADO POR: Mauricio

I Simpósio Brasileiro de Computação e Música (1.; 1994; Caxambu)

Anais do I Simpósio Brasileiro de Computação e Música, Caxambu, de 01 a 05 de agosto de 1994. Editado por Maurício Alves Loureiro. UFMG. 1994.

I. Loureiro, Maurício A., coord. II. Sociedade Brasileira de Computação.

## Prefácio

É com prazer que apresentamos os anais dos eventos científicos que fazem parte do XIV Congresso da Sociedade Brasileira de Computação. O Congresso de 1994 foi constituído de dez grandes eventos, além de oito palestras de conferencistas convidados de interesse geral e várias palestras convidadas de interesse específico dentro dos simpósios.

O aspecto político teve grande destaque através do XXIV Seminário de Computação na Universidade (SECOMU). O SECOMU foi constituído de cinco painéis e da criação de grupos de trabalho que apresentaram recomendações para a definição da política tecnológica brasileira na nossa área.

Na área científica tivemos a apresentação de 46 trabalhos no XXI Seminário Integrado de Software e Hardware (SEMISH), 24 trabalhos no VI Simpósio Brasileiro de Arquitetura de Computadores (SBAC), 24 trabalhos no I Simpósio Brasileiro de Redes Neurais (SBRN), 34 trabalhos e 48 composições musicais no I Simpósio Brasileiro de Computação e Música (SBC&M), além dos trabalhos selecionados no VII Concurso de Teses e Dissertações (CTD) e XIII Concurso de Trabalhos de Iniciação Científica (CTIC).

Na área acadêmica tivemos 14 cursos na XIII Jornada de Atualização em Informática (JAI), e na área de ensino de Ciência da Computação o II Workshop de Educação em Informática (WEI) promoveu o debate sobre avaliação e currículo dos cursos de computação e regulamentação da profissão, além de apresentar um volume contendo o Currículo de Referência da SBC e a descrição de 20 cursos de computação no país.

O XIV Congresso foi realizado pelo Departamento de Ciência da Computação da Universidade Federal de Minas Gerais e promovido pela Sociedade Brasileira de Computação. Gostaria de agradecer a todas as pessoas envolvidas na organização do Congresso, especialmente as pessoas do nosso Departamento, aos coordenadores de cada um dos dez eventos, as comissões de programa e de organização, aos avaliadores de artigos e aos estudantes de graduação e de pós-graduação dos nossos Cursos. Agradeço também o suporte financeiro do CNPq, FAPEMIG e FINEP e o apoio da CAPES, FAPERGS, FAPESP, TELEMIG e UFMG.

Belo Horizonte, junho de 1994

Nívio Ziviani  
Coordenador do XIV Congresso da SBC

## Apresentação

A produção brasileira em Computação e Música (Computer Music) teve início na década de 70 graças a um pequeno número de esforços isolados, contando sempre com recursos muito limitados. Hoje, o setor conta com a contribuição de um número considerável de pesquisadores e músicos que têm trabalhado arduamente não só na produção de arte e pesquisa, mas também na implementação e reconhecimento da Computação e Música na comunidade acadêmica.

No ano de 1993 foi criado o NUCOM — Núcleo Brasileiro de Computação e Música — com o objetivo de promover o desenvolvimento do setor no país, viabilizando o intercâmbio e o acesso à informação entre pesquisadores e artistas brasileiros dedicados à Computação e Música. Com o I Simpósio Brasileiro de Computação e Música, o NUCOM vem não só promover o encontro de artistas e pesquisadores brasileiros e estrangeiros, mas também uma mostra da produção brasileira e latino-americana dirigida à comunidade internacional de Computação e Música.

Foram selecionados 34 artigos e 48 composições por uma comissão de programa formada por pesquisadores e músicos brasileiros, alguns com residência fora do país, contando também com a participação de 2 membros externos à comunidade brasileira. As composições foram executadas em 6 concertos, para os quais o Simpósio contou com a participação especial do Grupo de Música Contemporânea da UFMG.

Os organizadores do I Simpósio Brasileiro de Computação e Música gostariam de expressar uma profunda gratidão às instituições que viabilizaram este evento: Sociedade Brasileira de Computação, Escola de Música da UFMG, Pró-reitorias Acadêmicas da UFMG, CNPq (Conselho Nacional de Pesquisa), FAPEMIG (Fundação de Amparo à Pesquisa do Estado de Minas Gerais), Fundação Rockefeller, CCRMA (*Center for Computer Research in Music and Acoustics, University of Stanford, California*), CRCA (*Center for Research in Computing and the Arts, University of California, San Diego*) e LIPM (*Laboratório de Investigación e Producción Musical, Buenos Aires*).

Belo Horizonte, julho de 1994

Maurício Alves Loureiro  
Coordenador do I SBC&M

## I SBC&M

### *Coordenador Técnico*

Maurício Alves Loureiro, Escola de Música/UFMG

### *Comissão de Artigos*

Aluizio Arcela, CIC/UnB

Eduardo Reck Miranda, Edinburg University, Escócia

Geber Ramalho, Université Paris VI, França

Jamary de Oliveira, Departamento de Música/UFBA

Wilson de Pádua de Paula Filho, DCC/UFMG

### *Comissão de Concertos*

Conrado Silva, Departamento de Música/UnB

Francisco Kröpfl, LIPM

Maurício Alves Loureiro, Escola de Música/UFMG

Robert Willey, CRCA

## XIV Congresso da SBC

### *Coordenador Geral*

Nivio Ziviani, DCC/UFMG

### *Sub-Coordenador*

Mário Fernando Montenegro Campos, DCC/UFMG

### *Secretária*

Elissandra Barbosa, DCC/UFMG

### *Comissão Organizadora*

Antonio Otávio Fernandes, DCC/UFMG

Geraldo Robson Mateus, DCC/UFMG

José Nagib Cotrim Árabe, DCC/UFMG

Oswaldo Farhat de Carvalho, DCC/UFMG

Rafael Guilherme Rodrigues da Silva, DCC/UFMG

Virgílio Augusto Fernandes Almeida, DCC/UFMG

### *Apoio*

Celso de Souza Lima, DCC/UFMG

Márcio Drumond Araújo, DCC/UFMG

Maria Aparecida Scaldaferrri Lages, DCC/UFMG

Maurício Antônio de Castro Lima, DCC/UFMG

Wagner Toledo Corrêa, DCC/UFMG

### *Assessoria de Comunicação*

Cláudia Graça da Fonseca, DCS/FAFICH/UFMG

## Sociedade Brasileira de Computação

### *Diretoria*

Ricardo Reis — Presidente  
Paulo Roberto Freire Cunha — Vice-Presidente  
Edson C.B. Carvalho Filho — Vice-Presidente Adjunto  
Cirano Iochpe — Secretário Geral  
Flávio Rech Wagner — Secretário Geral Adjunto  
Cláudia Maria Bauzer Medeiros — 1ª Secretária  
Miguel Jonathan — 1º Secretário Adjunto  
Roberto da Silva Bigonha — 2º Secretário  
Clarindo Isaías P. Silva e Pádua — 2º Secretário Adjunto  
Therezinha Souza da Costa — Tesoureira  
Emmanuel Lopes Passos — Tesoureiro Adjunto

### *Conselho — Membros Titulares*

Cláudia Motta  
Clésio Saraiva dos Santos  
Daltro Nunes  
Luiz de Castro Martins  
Marcos Borges  
Nivio Ziviani  
Pedro Manoel da Silveira  
Philippe Navaux  
Siang Wun Song  
Silvio Romero Lemos Meira

### *Conselho — Membros Suplentes*

Alberto Henrique Frade Laender  
Cláudio Kirner  
Daniel Schwabe  
Ricardo Anido  
Sérgio Schneider

## Sumário

<b>Sistemas e Linguagens para Síntese de Som, Processamento de Sinais e Transformação de Som</b>	<b>1</b>
1 Modelling the Excitation Function to Improve Quality in LPC's Resynthesis <i>Celso Aguiar</i>	3
2 Síntese Aditiva de Tons Musicais através da Transformada Karhunen-Loève <i>João Fernando Marar, Edson dos Santos Moreira</i>	9
3 The Synthesis of Complex Sonic Events by Functional Iterations <i>Agostino Di Scipio, Ignazio Prignano</i>	15
4 Modular Programming of Instruments in cmusic <i>Carlos Cerana</i>	21
5 On the Improvement of the Real-Time Performance of Two Fundamental Frequency Recognition Algorithms <i>Andrew Choi</i>	27
6 A Linguagem SOM-A para Síntese Aditiva <i>Aluizio Arcela</i>	33
7 Processador de Efeitos em Sinais Digitais de Áudio <i>Márcio da Costa Pereira Brandão, Carlos Augusto Jorge Loureiro, Túlio da Costa Zannon</i>	39
8 FracWave: Non-linear Dynamics as Timbral Constructs <i>Jônatas Manzolli</i>	45
9 An Overview of Criteria for Evaluating Synthesis and Processing Techniques <i>David A. Jaffe</i>	53
10 The Music Kit on a PC <i>David A. Jaffe, Julius O. Smith, Nick Porcarro</i>	63



<b>Sistemas de Notação Musical</b>	<b>71</b>
11 NotaCor — Impressão de Partituras em Cores <i>Alex de Oliveira Meireles</i>	73
12 Representação Angular para Notação Musical <i>Edilson Eulalio Cabral</i>	79
13 Positional Rhythmic Notation: an Implication for a Positional Theory of Rhythm <i>M. R. Moraes</i>	83
<b>Sistemas e Linguagens para Composição</b>	<b>89</b>
14 A Visual Programming Environment for Constraint Based Musical Composition <i>Camilo Rueda</i>	91
15 Incremental Evaluation in a Musical Hierarchy <i>M. Desainte-Catherine, K. Barbar, A. Beurivé</i>	99
16 CAMM — Automatic Composer of Musical Melodies <i>Eloi Fernando Fritsch, Rosa Maria Viccari</i>	107
17 Interfaces Musicais — um Problema Antigo <i>Domingos Aparecido Bueno da Silva</i>	121
18 Modelos Matemáticos e Composição Assistida por Computador, Sistemas Estocásticos e Sistemas Caóticos <i>Mikhail Malt</i>	125
19 Synthesizing Music with Sampled Sound <i>YeeOn Lo, Dan Hitt</i>	133
20 Um Ambiente de Auxílio a Composição Musical <i>Alexandre Jonatan Bertoli Martins, André Luiz Costa Ballista, Marcelo Soares Pimenta</i>	139
21 Orquestrador MIDI Sinfônico <i>Osman Giuseppe Gioia</i>	147
22 Editor de Preceitos Intervalares <i>Ricardo Ribeiro de Faria Castro</i>	155

23 The <u>SmOke</u> Music Representation, Description Language, and Interchange Format <i>Stephen Travis Pope</i>	161
<b>Análise Musical, Educação</b>	<b>167</b>
24 Learning Counterpoint Rules for Analysis and Generation <i>Eduardo Morales, Roberto Morales-Manzanas</i>	169
25 A System for Aiding Discovery in Musical Analysis <i>Edilson Fernalda, Carlos Alan Peres da Silva, Luciênio de Macêdo Teixeira, Hélio de Menezes Silva</i>	177
26 An Integral Education Project: Musical Production <i>Ricardo Dal Farra</i>	185
<b>Inteligência Artificial, Psicoacústica e Modelos Cognitivos</b>	<b>187</b>
27 A Connectionist Model for Chord Classification <i>Fabio Ghignatti Beckenkamp, Paulo Martins Engel</i>	189
28 The MusEs System: an Environment for Experimenting with Knowledge Representation Techniques in Tonal Harmony <i>Francois Pachet</i>	195
29 ARTIST: an AI-based Tool for the Design of Intelligent Assistants for Sound Synthesis <i>Eduardo Reck Miranda</i>	203
30 Representing Musicians' Actions for Simulating Improvisation in Jazz <i>Geber Ramalho</i>	217
<b>Desempenho, Interface com o Usuário e Projeto de Instrumentos</b>	<b>223</b>
31 A Phenomenological Study of Timbral Extension in Interactive Performance <i>Anna Sofie Christiansen</i>	225
32 Um Novo Músico Chamado "Usuário" <i>Fernando Lazzetta</i>	231

<b>33</b> A New Technology for Musical Sound Synthesis and Control <i>Richard Hodges</i>	<b>237</b>
<b>34</b> Virtual Musical Instruments: Accessing the Sound Synthesis Universe as a Performer <i>Axel Mulder</i>	<b>243</b>
<b>Índice por Autor</b>	<b>251</b>

## Sistemas e Linguagens para Síntese de Som, Processamento de Sinais e Transformação de Som

## Modelling the Excitation Function to Improve Quality in LPC's Resynthesis.

CELSO AGUIAR

*CCRMA - Center for Computer Research in Music and Acoustics.  
Stanford University, Stanford, CA 94305-8180 USA.  
aguiar@ccrma.stanford.edu*

### ABSTRACT

LPC (Linear Predictive Coding) is a well known technic for speech analysis-synthesis. The analysis consists in finding a time-based series of n-pole IIR filters whose coefficients better adapt to the formants of a speech signal. These computations produce a residual signal which is the exact complement to the information kept in the coefficients, if we wish to recover the original. The model of the human vocal tract mechanism assumed by LPC presupposes that speech can be reduced to a succession of voiced or unvoiced sounds. Thus, an excitation function composed of pulse or noise substitutes the residual in the resynthesis. This assumption is adequate for some tasks but, in a musical context, the artifacts introduced can yield unsatisfactory results. This article proposes a simple alteration in the model to improve the quality of LPC's resynthesis. Some of the conventional problems like "buzzy" quality and loss of coloration are partially corrected.

### INTRODUCTION

The problem of quality in the resynthesis with linear prediction technics has already been noticed in the Computer Music literature (Lansky, 1989) but not sufficiently addressed. Some of the critical problems impeding the accomplishment of higher quality results are the general "buzzy" quality of the sounds (due to the use of a band-limited pulse signal to model the voiced components in the speech), alterations in the coloration of the original speech and loss of energy in the region of the fundamental.

These problems altogether are enough to remind us that the technic was not initially created to generate high quality results as may be needed in Computer Music. Although quality and intelligibility were also an issue in its first applications, LPC was basically created as an analysis technic for data reduction in speech transmission. Under these conditions we must conclude that it would be necessary for the technic to undergo some kind of adaptation or improvement if it is to be applied in its full potential for our purposes.

In this paper, the model employed by LPC to reproduce the human vocal-tract mechanism is described as well as are discussed the reasons for the improvements here introduced. Due to the sufficient literature in the field, only a brief description of the LPC technic is done. Finally, we present some sound examples that demonstrate the ideas discussed.

### THE LPC MODEL OF SPEECH REPRODUCTION

The main idea behind linear predictive coding is that a sample of speech can be approximated as a linear combination of past speech samples. An inverse filter must be produced so that it matches the formant regions of the speech under analysis. The coefficients for that filter are determined by minimizing the square difference between the speech samples and the linearly predicted ones. The process of LPC analysis is nothing less than the tentative application of this filter to the speech samples. These computations produce what is known as the

residual sound in this field of research.

The model used by linear prediction is the same as used conventionally in most of the models for the digital representation of speech (Schaffer & Rabiner, 1975). It is fundamentally based on the human vocal-tract mechanism, which can be thought as an acoustic tube terminated at one end by the vocal cords and at the other by the buccal cavity and lips. The shape of this tube is determined by the position of its components like lips, tongue, jaw and vellum.

The sound sources generated by the air pressure coming from the lungs can excite the vocal tract in several ways. These can be divided in two basic types of excitation: quasi-periodic pulses which generate the voiced sounds, and a noisy source which produces the unvoiced sounds of speech. Both types of excitation sources are modelled by LPC and are represented by the two boxes in the diagram below (Fig. 1).

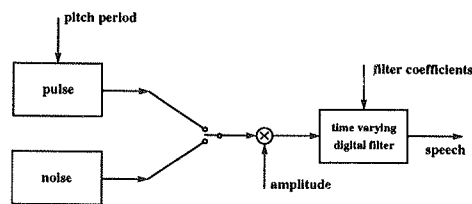


Fig. 1 - LPC's digital processing model.

The main characteristic of that residual sound from LPC analysis described above is to be a close approximation to the vocal tract input signal (Cann, 1985). If the formants are being extracted efficiently by the filtering performed during the analysis, the residual should look like a combination of noise and a series of pulses. LPC does not provide a way to gradually mix these two sources when modelling the excitation. The boundary between voiced and unvoiced speech is precisely one of the places where artifacts are bound to appear in LPC resynthesis.

The two sources cited above create a wide band excitation of the vocal tract that acts as a linear time-varying filter (Fig. 1). This filter imposes its characteristics on the frequency spectra of the excitation sources. The vocal tract is thus characterized by its natural resonances and is represented in the LPC model by the time varying digital filter whose coefficients should match the formants present in the analyzed signal.

#### IMPROVING THE LPC MODEL

The experiments and conclusions reached in this research stem from a *Voice Class* course in the Spring of '93 at CCRMA. Our initial project was just to improve the quality of some programs developed by Professor Perry R. Cook to perform LPC analysis and resynthesis according to the autocorrelation method. The programs were improved, a new pitch tracking routine was included but, when experimenting with sounds sampled at 44 kHz, we noticed that the quality of the resulting sounds would not improve as normally happens when dealing with sampled sounds. On the contrary, the coefficients extracted from these 44 kHz sounds would do a worse job than the ones extracted from the same files handled at 22 kHz. The same phenomena would manifest in other LPC programs like CMix's routines for performing the covariance method. A good explanation for this fact is that the formants present in the speech will not change their position with the change in sampling rate. When analysing at a higher sampling rate, we would find ourselves trying to do the job of extracting the same formants, only that now we had more samples to handle and compute, as if trying to do the job in a less efficient manner.

Another interesting observation that can be made about the LPC process is that if the residual is the precise complement to the information kept by the coefficients then, the more our excitation function looks like the residual the closer we will be to recovering completely the original signal. The problem with the direct use of the residual in LPC is that it can only be used as it is, and only to recover the original. In other words, if we wish to stretch our sound in time and try to do this by stretching the residual and feeding it back as the excitation

function in the LPC algorithm, the artifacts introduced in the process of stretching the residual will propagate into the resulting sound and the result is the same as stretching the original sound with the same method applied to the stretching of the residual.

So, if the difference between excitation function and residual is what keeps us from recovering completely the original, or at least coming close to do that, the residual would probably be able to contribute with an extra amount of formant information that is not present in the conventional way of extracting the coefficients. With this in mind, we included another stage in the LPC model in order to diminish the discrepancies between the excitation function and the residual. The idea is to perform in the residual the same analysis done to the original sound. The extra information extracted from the residual is then stored in the coefficients that result from the second LPC analysis (Fig. 2).

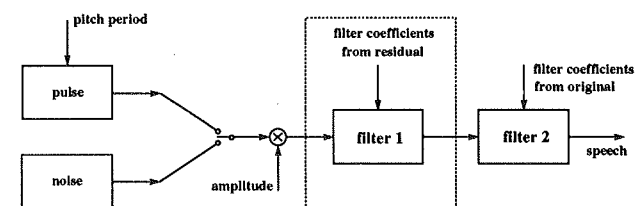


Fig. 2 - Improved LPC model including another filtering stage (residual coefficients) in cascade.

Other parameters used in the resynthesis process, like pitch, power and the information on the voiced or unvoiced content of the speech are extracted in the same way as before, from the original. This change in the algorithm together with the adjustments performed in the frequency function resulting from pitch tracking (see section below) have proved to be capable of recovering almost completely the characteristic of the original sound. As we are not directly using the residual in the resynthesis step, but rather its formant information expressed by its coefficients, the same possibilities that are present in the conventional LPC model are still present in this modified model like time stretching or changing pitch independently from duration. It can be argued that this change in the model will consume an extra amount of computing time (the double time to be more exact). It is clear that the application of this improvement can be delayed until the end of the process of obtaining the sounds in a composition, after all the choices have been made and as a means of enhancing the result.

In our experiments we have also found out that increasing the number of stages and keeping constant the number of poles (extracting the coefficients of the residual of the residual) will not yield any better results. This should be happening probably because all the useful formant information has already been completely extracted when we performed the analysis of the residual. Another possibility that we have not directly investigated in our research would be to do an increase in the number of stages while analysing the sound with a smaller number of poles.

#### IMPROVING THE QUALITY OF PITCH TRACKING

Another very important aspect of LPC analysis-resynthesis procedures that should also be stressed, is the quality of the pitch tracking realized in the original sound. The quality of the sounds resulting from the resynthesis depends entirely on the quality of the pitch tracking initially done on the sound.

The pitch detector developed by Professor Perry Cook and used in our programs is a lag-domain type of pitch detector. It uses AMDF technic and refinements to determine the pitch of a quasi periodic sound. In this pitch detector a delay line size is determined according to lower and higher pitch limits specified. For the female voice used in our experiments the limits of 90 Hz and 525 Hz have been sufficient. In case of a male speaker the lower limit can be adjusted to 50 Hz. The pitch tracking is also realized frame by frame. In our tests a frame and hop size of 600 and 200 samples respectively (the same values for the equivalent parameters used in the LPC analysis) for the female speech sampled at 22050 Hz, have proven to be the most effective.

Once the pitch tracking is done, a Frequency X Time function like the one in Fig. 3 emerges as a result of the algorithm. At this point it is very important to be able to adjust this curve for resynthesis. This adjustment is mostly necessary due to imperfections in the result emerging from the pitch tracking algorithm. These imperfections come in the form of glitches that can be easily detected by a small deglitching routine or avoided for different sections of the Frequency X Time curve by a redefinition of the frame and hop sizes in the pitch tracking input.

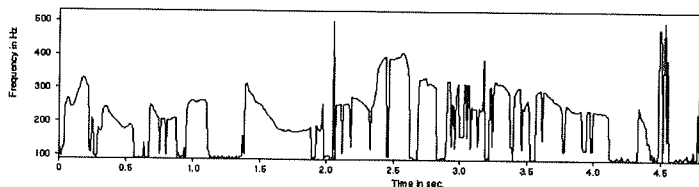


Fig. 3 - Frequency function obtained from the pitch tracking algorithm.

The worst artifacts introduced in the resynthesis by these imperfections occur in the boundaries between voiced and unvoiced frames. In case the pitch tracker fails to produce a good estimate of the pitch for a voiced frame, which is very conceivable due to the proximity to a noisy section of the speech, the resynthesis will yield a false frequency for the pulsed excitation in that frame, producing a very annoying effect. The best option in cases like this is to just eliminate these points and interpolate between the remaining ones. A smoother curve like the one displayed in Fig. 4 should be sufficient to produce a more faithful result.

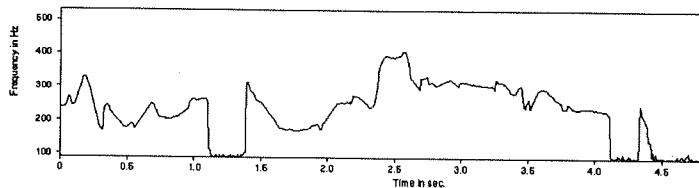


Fig. 4 - Adjusted version of frequency function used to improve resynthesis

Apparently, any compositional elaboration to be introduced in the frequency dimension of a speech phrase with LPC, should probably be based on this adjusted version of the curve rather than in the inaccurate version coming out of the pitch tracker.

#### SOUND EXAMPLES

With the purpose of demonstrating the ideas discussed in this paper, an example phrase of female speech was analyzed and resynthesized in several different conditions. An audio tape containing these sound examples can be ordered from the author.

1. Original speech phrase used: "I burst out in scorn, at the reprehensible poverty of our sex". Female voice sampled at 22050 Hz. Duration of 4.79 sec.
2. Residual produced by the LPC analysis of sound example 1. Autocorrelation method using the

standard LPC model was employed. Number of poles was 32, block size 600 and hop size 200. Amplitude has been boosted up for demonstration purposes only.

3. Resynthesis of the phrase using the residual as excitation.
4. Resynthesis of the phrase using a pulse-noise combination as excitation. No adjustments were made in the frequency function obtained from pitch tracking (Fig. 3).
5. Resynthesis of the phrase using a pulse-noise combination as excitation. Adjustments were made in the frequency function according to what is displayed in Fig. 4.
6. Improved resynthesis of the original according to our refined model proposed. The LPC analysis program was modified so that it also extracts the LPC coefficients from the residual sound. Number of poles for the analysis of the original was 32, number of poles for the analysis of the residual was also 32, block size 600 and hop size 200. Resynthesis is further improved by including the adjustments made in the frequency function obtained from pitch tracking.
7. Excitation function used to produce sound example 5.
8. Excitation function used to produce sound example 6.
9. Resynthesis via covariance analysis method using CMix's LPC programs. Parameters were set according to Lansky's advice: number of poles of 24, block size of 200, hop size of 100. Adjusted version of frequency function was employed.
10. Time stretched version of sound example 9 (3 to 1 stretch).
11. Time stretched version of sound example 6 (3 to 1 stretch).

#### REFERENCES

- Cann, R. (1985). "An Analysis/Synthesis Tutorial." In *Foundations of Computer Music*, ed. Curtis Roads and John Strawn. Cambridge, Mass.: MIT Press, 114-44.
- Lansky, P. (1989). "Compositional Applications of Linear Predictive Coding." In *Current Directions in Computer Music Research*, ed. Max V. Mathews and John Pierce. Cambridge, Mass.: MIT Press, 5-8.
- Schaffer, R. & Rabiner, L. R. (1975). "Digital Representations of Speech Signals." *Proceedings of the IEEE*, 63(4), 662-77.

#### ACKNOWLEDGMENTS

My acknowledgements go to Professor Perry R. Cook for his challenge and support during the *Voice Class* course in the Spring of '93 at CCRMA, Stanford. I'd also like to thank Mr. Paul Lansky for confirming that I should write this paper, and composer Marco Trevisani for providing the speech samples used during this research. The paper was written while under a Master of Arts fellowship from the Brazilian Agency CAPES, to which I am most indebted.

## Síntese Aditiva de Tons Musicais através da Transformada Karhunen-Loève

JOÃO FERNANDO MARAR<sup>1</sup>  
EDSON DOS SANTOS MOREIRA<sup>2</sup>

<sup>1</sup>UNESP—Universidade Estadual Paulista  
Faculdade de Ciências - Departamento de Computação  
Bauru - São Paulo - Brasil  
jfm@di.ufpe.br

<sup>2</sup>USP—Universidade de São Paulo  
ICMSC—Instituto de Ciências Matemáticas de São Carlos  
Departamento de Computação  
São Carlos - São Paulo - Brasil  
edsmorei@icmcs.sc.usp.br

**Abstract:**

Signals can be sampled, stored and played back by digital computers. Furthermore, signals can be analyzed in such a way that compressed forms of their representation can be extracted, reducing the amount of memory needed to store them. The Fourier transform have been traditionally used in this way. This paper deals with the use of the Karhunen-Loève transform as an alternative method of representing a particular and dynamic signal, the musical tones, allowing bigger savings of computing resources, as compared to the Fourier transform.

**1 Introdução**

Nos últimos anos temos acompanhado uma rápida evolução na tecnologia de construção de computadores. Como consequência, deste desenvolvimento, associado à competitividade, o custo de produção e comercialização dos equipamentos vem barateando dia após dia, forçando a indústria, periodicamente, a substituir seus equipamentos disponíveis no mercado, anexando melhorias ou simplesmente desenvolvendo novos projetos que atraiam usuários (consumidores). Neste contexto inserem-se os sintetizadores, os quais aparecem como parte da evolução do movimento musical, que passa a buscar novos recursos, para produzir formas diferentes de sons, das que já são geradas pelos instrumentos musicais tradicionais.

Instrumentos musicais são fontes geradoras de estímulos sonoros, equivalentemente, geradoras de sinais. Através da amostragem de um sinal sonoro, obtém-se sua representação discreta no domínio do tempo. A análise destes sinais discretos, tem por objetivo extrair características para posterior reprodução. Uma transformação para o domínio da frequência, auxilia a análise do sinal.

A transformação de sinais entre os domínios do tempo e frequência, geralmente, não se configura como um problema de grandes proporções e várias ferramentas estão disponíveis. Para isto utilizamos a

transformada de Fourier, transformada de Laplace, transformada Z, entre outras [WIN 78, DeF 88, PRO 88]. Entretanto, para uma determinada classe de sinais, algumas particularidades existem, o que dificulta a reconstrução precisa do sinal através das informações espectrais. Tal é o caso de sinais que apresentam um quadro de componentes que variam com o tempo de duração do sinal. Estas variações podem envolver as componentes de frequência, suas amplitudes e fases. A reconstrução do sinal a partir das suas características espectrais exige o emprego de um tratamento dinâmico, resultando em grande demanda de potência aritmética do processador e/ou na quantidade de memória requerida. É o que se verifica na síntese de tons musicais [MOO 77].

A reprodução fiel de um instrumento musical implica em se levar em consideração todas as variações em amplitude, quantidade e fase dos harmônicos que ocorrem durante o tempo de duração de um tom. Neste sentido, várias técnicas existem e hoje são utilizadas, dentre as quais, destacam-se a distorção de fase (*Casio*), frequência modulada (*Yamaha*), métodos aditivos, subtrativos e métodos *sampling* [MAS 87].

O esforço computacional exigido por métodos de síntese de tons musicais, podem torná-los proibitivos para execução em tempo real, utilizando-se

computadores convencionais. Trabalhos têm sido desenvolvidos na tentativa de simplificar a geração de tons musicais, através da utilização de funções não senoidais [HUT 75, STA 88]. Estes métodos exigem uma cuidadosa análise, para que se evite a degradação na qualidade do som produzido, bem como para manter a simplicidade para o controle das funções parâmetros, amplitude e frequência, para construção de novos tons [MOO 77].

Este artigo aborda a síntese aditiva, bem como salienta alguns conceitos básicos, fundamentais para a aplicação de computadores em síntese de tons musicais e, finalmente, descreve um método de síntese aditiva, utilizando um conjunto de funções básicas não senoidais. Neste artigo, utilizaremos as funções básicas provenientes da transformada Karhunen-Loève (K-L). O método K-L, também conhecido por análise das principais componentes do sinal [CHE 91] é baseado na criação da matriz de covariância dos dados de entrada (sinais). Os auto-vetores, principais componentes, são extraídos desta matriz e ordenados de acordo com a magnitude dos auto-valores. O primeiro auto-vetor está associado com o maior autovalor, e assim sucessivamente. Resultados práticos demonstram os ganhos computacionais através de comparação com outros métodos [CAM 71, STA 88], possibilitando sua aplicação em tempo real.

## 2 Características Físicas do Som

O movimento vibratório de uma massa em contato com o ar produz som. Isto corresponde a variações na densidade e na pressão do ar ao redor da massa.

Em termos de tons musicais, os movimentos vibratórios podem ser causados por instrumentos acústicos ou por meio de ondas eletricamente excitadas através de um alto falante. Todo som possui três propriedades características:

- **Volume:**

É descrito pela amplitude do som. Um fato bastante interessante, analisando tons musicais, é que a amplitude, bem como os outros dois parâmetros, varia com o tempo. A maneira pela qual a amplitude varia é descrita pelo envelope do tom musical. Usualmente, o envelope pode ser dividido em três fases distintas: ataque, sustentação e decaimento.

Na figura 1, a direita, podemos observar que nem todos os instrumentos possuem estas três fases. Por exemplo, os instrumentos de percussão geram tons totalmente não periódicos, de modo a não possibilitarem uma estimativa de sua frequência fundamental; a esquerda, ilustra o comportamento de um tom de clarinete que apresenta nitidamente seu envelope.

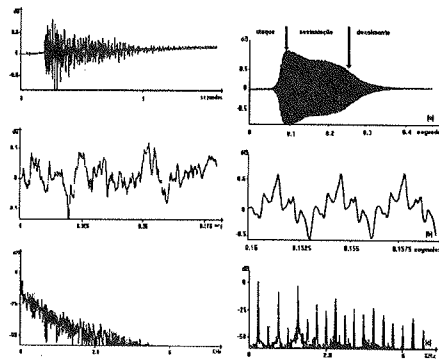


Figura 1: Características de um bumbo de bateria (direita) e características de uma clarinete (esquerda), a figura (a) representa a amostragem do sinal inteiro, (b) representa um pequeno segmento do sinal ampliado no eixo do tempo, (c) representa o espectro do sinal [MOO 77].

- **Altura Tonal:**

É descrita pela frequência fundamental. Na realidade, o som é formado por uma mistura de frequências, sendo que a menor delas é chamada de fundamental e uma combinação de outras frequências maiores chamadas de harmônicos, sobretons ou parciais.

- **Timbre:**

É descrito como a qualidade distintiva de sons de mesma altura e intensidade, e que resulta da quantidade maior ou menor dos harmônicos coexistentes com a frequência fundamental.

## 3 Síntese Aditiva de Tons Musicais

A principal característica de síntese aditiva, em tons musicais, é a representação do sinal através de um conjunto de funções ortogonais. Esta técnica quando baseada na representação truncada de Fourier, é um dos métodos mais conhecidos em geração de tons, devido à sua fidelidade. Entretanto, esta representação não é utilizada em aplicações de tempo real, devido à grande demanda de potência aritmética exigida do processador. Este método tem sua performance ainda mais degradada, principalmente se for esperado que um mesmo sintetizador gere tons de diversos instrumentos. Em geral, um tom musical,  $x(\bullet)$ , é

aproximado pela equação 1:

$$x(n) \cong \sum_{i=1}^K A_i(n) \phi_i(\theta_i(n)) \quad (1)$$

onde  $\{\phi_i\}$  representa o conjunto de funções periódicas e ortogonais (funções básicas),  $A_i(\bullet)$  representa a função amplitude (envelope) e  $\theta_i(\bullet)$  representa a função fase. A função frequência,  $F(\bullet)_i$ , é determinada através da derivada de  $\theta_i(\bullet)$  no tempo.

O processo de análise, extrairá os parâmetros necessários para o cálculo da equação 1, que representa a síntese. A figura 2 ilustra a extração dos parâmetros e a composição destes para representação de um tom musical.

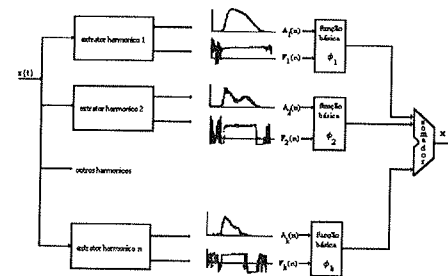


Figura 2: Extração de parâmetros e reconstrução de um tom musical através da síntese aditiva.

A análise não caracteriza um problema de processamento em tempo real, mas a síntese é caracterizada como tal. Em [WAW 89], aborda-se meios para contornar alguns problemas encontrados em síntese de tons musicais em tempo real, sendo que o maior deles reside na grande quantidade de dados envolvidos. Por exemplo, suponhamos que nossa amostragem do sinal seja feita a uma taxa de aquisição de 40 KHz. Neste caso, necessitamos armazenar 40 mil valores por segundo. Pela equação 1, podemos ter uma ideia do "tamanho" das funções amplitude e frequência que são de mesma ordem de  $x(\bullet)$ , que é a amostra do tom completo. Note que, estas funções são necessárias para cada harmônico extraído.

O esforço computacional baseado em aplicações que utilizam este método de síntese provem de duas origens: na geração das funções amplitude e frequência para cada harmônico e na multiplicação das funções amplitude com as funções básicas.

Teoricamente, qualquer conjunto de funções periódicas que satisfaça as condições de ortogonalidade [AHM 75], pode ser utilizado para geração de tons

musicais, baseado em síntese aditiva. A escolha deste conjunto está intimamente ligada aos objetivos propostos pela síntese. Desejamos sintetizar tons musicais em tempo real. Para isso, a redução do número de funções básicas será uma condição necessária e inevitável. A figura 3 ilustra um diagrama mínimo para a síntese aditiva. Embora seja bastante natural

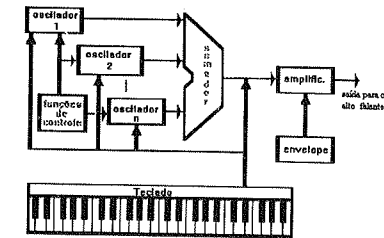


Figura 3: Representação em blocos para um sintetizador baseado em síntese aditiva

a escolha de funções básicas senoidais, deve-se levar em conta a demanda computacional exigida por este conjunto. Dependendo do tom a ser gerado, serão necessárias mais de 30 senóides para minimizar as diferenças perceptíveis entre o tom sintetizado e o natural [GRE 77]. Várias pesquisas utilizando funções básicas não senoidais tem sido realizadas. As funções Walsh [AHM 75] têm apresentado muitas vantagens em relação às funções senoidais, devido à facilidade de geração destas funções básicas em computadores digitais. Infelizmente, são necessárias mais de 16 funções Walsh para minimizar as diferenças perceptuais [HUT 75].

Na seção seguinte, apresentaremos a transformada Karhunen-Löve com o objetivo de reduzir o número funções básicas necessárias.

## 4 Representação de Sinais Utilizando Transformadas Karhunen-Loève

Uma importante aplicação de transformadas ortogonais, em processamento de sinais digitais, é a compressão de dados [AND 70, AHM 75, WOM 77, STA 88, CHE 91], isto é, a representação de um sinal de maneira mais eficiente. As transformadas ortogonais podem ser divididas em duas classes, segundo as funções básicas utilizadas. Desta forma, temos a classe baseada em funções não senoidais e a de funções senoidais, tendo esta última como única representante a transformada de Fourier.

Faremos agora, uma breve introdução a expansão

K-L. Seja  $\{X\}$  um conjunto de vetores, obtidos por amostragem, de uma classe de sinais aleatórios; podendo ser tons musicais. Um representante de  $\{X\}$  é dado por  $x_j = (x_{j,1}, x_{j,2}, \dots, x_{j,K})$ . A amostra  $x_j$  pode ser aproximada por 2:

$$x_j = y_{j,1}\phi_1 + y_{j,2}\phi_2 + \dots + y_{j,N}\phi_N = \sum_{i=1}^N y_{j,i}\phi_i \quad N < K \quad (2)$$

$$y_{j,i} = x_j^t \phi_i \quad i = 1, 2, \dots, N \quad (3)$$

onde  $K$  é o número total de componentes da amostra e  $N$  é o número de componentes utilizadas na aproximação.

Por definição, o mínimo erro quadrado,  $\varepsilon$ , é dado pela expressão 4:

$$\varepsilon = \left( \sum_{i=1}^K y_{j,i}\phi_i - \sum_{i=1}^N y_{j,i}\phi_i \right)^2 = \sum_{i=N+1}^K \phi_i^t R_X \phi_i \quad (4)$$

onde  $R_X$  é a matriz de covariância do conjunto  $\{X\}$ . Dada por  $R_X = \frac{1}{V} \sum_{j=1}^V (x_j - \bar{X})(x_j - \bar{X})^t$ , onde  $V$  representa o número total de elementos do conjunto  $\{X\}$  e  $\bar{X}$  é o vetor médio do referido conjunto. Quando  $\{\phi_i\}$  constituem a base ortogonal de Karhunen-Loève, os elementos  $\phi_i$  são determinados a partir dos autovetores de  $R_X$ , de acordo com a equação 5:

$$R_X \phi_i = \lambda_i \phi_i \quad (5)$$

O erro de truncamento da equação 4 é dado através da equação 6

$$\text{Min}(\varepsilon) = \sum_{i=N+1}^K \lambda_i \quad (6)$$

Isto significa que, se utilizarmos apenas  $N$  autovetores para a representação de funções, o erro de truncamento será mínimo, sendo dado pela equação 6. A equação 2, escrita em termos dos auto-vetores da matriz de covariância, é denominada expansão Karhunen-Loève. A correspondente transformação ortogonal inversa, na equação 3, é chamada transformada Karhunen-Loève (K-L).

## 5 Aplicação de Transformada Karhunen-Loève em Tons Musicais

Pela maneira que expomos a transformada K-L, pode nos sugerir uma aplicação direta do tom musical completo, conforme ilustra a figura 1. Porém, se observarmos atentamente a equação 5, notaremos que a dimensão da matriz de covariância está associada a duração do tom musical e este, por sua vez, à taxa de

amostragem, a qual o tom foi submetido. Em geral, tons musicais são amostrados à 40 KHz. Portanto, uma aplicação direta da transformada K-L seria impossível, devido à dimensão da matriz de covariância. Contudo, esta preocupação está descartada, pois, por definição, a síntese aditiva requer apenas que o conjunto de funções básicas seja periódico, e não de mesmo comprimento do tom original.

Neste instante, faremos algumas adaptações à transformada K-L para tons musicais, mais especificamente na computação da matriz de covariância, para uma grande quantidade e variedade de tons de instrumento musicais.

Para cada tom musical amostrado, selecionamos um conjunto de funções-amostras para o cálculo da matriz de covariância, *i.e.*, seja  $\{x_\xi\}$  um conjunto de funções-amostras de  $x_\xi(\bullet)$ , um membro deste conjunto é dado por  $x_\xi^t = (x_{\xi,1}, x_{\xi,2}, \dots, x_{\xi,p})$ , assumiremos que tais conjuntos de amostras possuem o vetor médio identicamente nulo, facilitando o cálculo da matriz de covariância. Desta forma, podemos expressar a matriz de covariância, através da seguinte equação:

$$R_X = \frac{1}{V} \sum_{\xi=1}^{k_p} x_\xi x_\xi^t \quad (7)$$

Como já sabemos, as funções básicas K-L são determinadas através da solução dos auto-vetores, pela equação 5.

Um resultado interessante, obtido através do método K-L, é que o espectro de frequências de um auto-vetor pode ser escrito como combinação linear das transformadas de Fourier das funções-amostras, o qual é dado utilizando as equações 5 e 7:

$$\mathcal{F}(\lambda_i \phi_i) = \mathcal{F}\left(\sum_{\xi} x_\xi x_\xi^t \phi_i\right)$$

$$\mathcal{F}(\lambda_i \phi_i) = \sum_k \sum_{\xi} x_\xi(k) x_\xi^t \phi_i e^{-j\omega k t}$$

$$\mathcal{F}(\lambda_i \phi_i) = \sum_{\xi} x_\xi^t \phi_i \left( \sum_k x_\xi(k) e^{-j\omega k t} \right)$$

$$\mathcal{F}(\lambda_i \phi_i) = \sum_{\xi} x_\xi^t \phi_i \mathcal{F}(x_\xi) \quad (8)$$

Baseado no resultado obtido pela equação 8, impomos as seguintes restrições para a aplicação desta técnica:

- 1. As funções-amostras devem ter um período fundamental, por exemplo  $k_p$  amostras. Isto determinará o período fundamental dos auto-vetores.

- 2. As funções-amostras devem ser periódicas e terem banda limitada. Assim, os auto-vetores terão estas propriedades.

Com estas restrições, agruparemos as amostras dos tons musicais em classes semelhantes, para a computação das funções básicas. Após esta computação, determinaremos as funções amplitude e frequência, pela técnica de funções lineares por partes.

## 6 Procedimentos para a utilização da Transformada K-L em Síntese Aditiva de Tons Musicais

Na prática, nosso objetivo será alcançado dividindo-se o problema em 5 procedimentos, os detalhes sobre os procedimentos podem ser encontrados em [MAR 92]. Abaixo, relacionamos apenas os tópicos principais:

**Normalização das Funções-amostras:** procedimento utilizado para o tratamento dos dados conforme as restrições (1) e (2), de maneira a garantir a validade da equação 8.

**Alinhamento de Fase das Funções-amostras:** Procedimento utilizado para auxiliar a concentração de energia nos primeiros auto-vetores, consequentemente nas funções básicas K-L, possibilitando uma redução significativa deste conjunto para a síntese [CHR 79]. Por exemplo, se cada função-amostra possui um ponto de máximo, o qual contenha uma grande fração de energia do sinal, estas funções serão alinhadas de maneira a que estes máximos ocorram no mesmo tempo para todas as funções-amostras.

**Classificação das Funções-amostras:** procedimento utilizado para auxiliar a concentração de energia nas funções básicas K-L, através de classificação dos instrumentos musicais [STA 88].

**Determinação das Funções Básicas K-L:** procedimento utilizado para a determinação, através da equação 5, das funções básicas para cada uma das classes encontradas.

**Determinação das Funções Amplitude e Frequência:** procedimento utilizado para a determinação dos parâmetros variantes no tempo. Uma maneira eficiente para a geração das funções amplitude e frequência é dada através de funções lineares por partes (piecewise linear) [MOO 77, GRE 77, STA 88, SER 90].

A figura 4 ilustra a aplicação deste método em síntese de tons musicais.

## 7 Conclusões

Todos os sinais sonoros existentes na natureza são complexos, analógicos e contínuos no tempo. Através

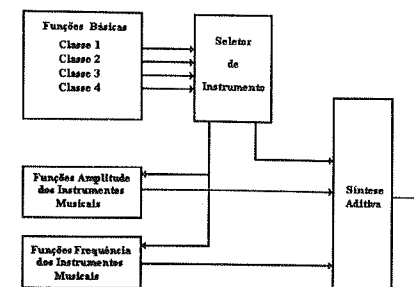


Figura 4: Ilustra o processo de síntese aditiva baseada em transformada Karhunen-Loève para 4 classes de instrumentos musicais. Utilizamos um seletor para enviar apenas as funções básicas necessárias para cada instrumento e suas respectivas funções amplitude e frequência

de uma cuidadosa amostragem, podemos obter uma sequência de valores discretos no tempo para a representação fiel destes sinais. Desta forma, os computadores podem ser utilizados para processar ou reproduzir este tipo de informação.

Sons de alta qualidade tem sido gerados por técnicas de síntese aditiva. Uma das vantagens desta técnica é a flexibilidade fornecida no controle das funções básicas que são somadas para a reconstrução do tom. Entretanto, a carga computacional exigida na implementação desta técnica, quando baseada na representação truncada de Fourier, limita seu uso em aplicações de tempo real, devido a grande quantidade de funções básicas necessárias. Este quadro se complica ainda mais, quando se deseja que um mesmo sintetizador reproduza tons de vários instrumentos.

Dentro deste contexto, estudamos maneiras alternativas de representação de sinais utilizando transformadas ortogonais baseadas em funções não senoidais, dentre as quais destacamos a transformada Karhunen-Loève. Esta transformada possibilita a representação de sinais sonoros a partir de um número reduzido de funções básicas. A redução das funções básicas é conseguida através de uma análise conjunta de amostras de todos os instrumentos desejados.

A utilização desta técnica tem mostrado sua eficiência. Na análise realizada por Stapleton [STA 88], foram realizadas, a classificação e a síntese para vários instrumentos. A tabela 1 ilustra o resultado da classificação para 12 instrumentos musicais.



Classe	Instrumento	F. básicas
1	Diapásão	1,2
	Flauta	1,2,3,4
	Guitarra	1,2,3
	Marimba	1,2,3,5
	Violino	1,2,3,4
2	Contrabaixo	1,2,3,5
	Trompete	1,2,3
	Trompa Francesa	1,2
3	Trombone	1,2
	Clarinete	1,2,3
	Sax Tenor	1,2
	Sax Alto	1,2

Tabela 1: Ilustra um exemplo de classificação de instrumentos musicais e as respectivas funções básicas K-L.

Como podemos observar, a tabela 1, apresenta os instrumentos divididos em classes, as funções básicas utilizadas por cada instrumento. Um exemplo da eficiência da técnica é observado quando da síntese do clarinete figura 1: uma representação razoável utilizando Fourier exigiria em torno de 19 funções básicas [MOO 77], enquanto que por Karhunen-Loève, seriam necessárias apenas 3, como visto na tabela 1.

Este método de representação de sinais pode ser utilizado com sucesso em outras aplicações onde as informações componentes dos sinais variam rapidamente. Tal é o caso de tons não harmônicos.

#### Referências

- [AHM 75] N. Ahmed, K.R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer-Verlag, New York, 1.975.
- [AND 70] H. C. Andrews, J. Kane, *Kronecker Matrices, Computer Implementation, and Generalized Spectra*, Journal of the Association for Computing Machinery, Vol 17, Nro 2, April 1.970, p 260-268.
- [CAM 71] S.J. Campanella, G.S. Robinson, *A Comparison of Orthogonal Transformations for Digital Speech Processing*, IEEE Transactions on Communication Technology, Vol.Com-19, Nro 6, December 1.971, p 1045-1049.
- [CHE 91] C.S. Chen, K.S. Huo, *Karhunen-Loève Method for Data Compression and Speech Synthesis*, IEE Proceedings-I, Vol. 138, Nro. 5, October 1.991, p 377-380.
- [CHR 79] R.A. Christen, A.D. Hirschman, *Automatic Phase Alignment for the Karhunen - Loève Expansion*, IEEE Transactions on Biomedical Engineering, vol.bme-26,Nro 2, February 1.979, p 94-99.

[DeF 88] D.J. DeFatta, J.G. Lucas, W.S. Hodgkiss, *Digital Signal Processing: A System Design Approach*, John Wiley & Sons, New York, 1.988

[GRE 77] J.M. GREY, J.A. Moorer, *Perceptual evaluations of Synthesized Musical Instrument Tones*, J. Acoust. Soc. Am., Vol. 62, Nro 2, August 1.977, p 454-462.

[HUT 75] B.A. Hutchins, *Applications of a real-time Hadamard Transform Network to sound synthesis*, Journal of Audio Engineering Society, Vol 23, Nro 7, September 1975, p 558-562.

[MAR 92] J. F. Marar *Utilização da Transformada Karhunen-Loève em Síntese de Tons Musicais*, Dissertação de Mestrado - USP-São Carlos, 1.992.

[MAS 87] H. Massey, A. Noyes, D. Shklair, *A Synthesist's Guide to Acoustic Instruments*, Amsco Publications, New York, 1.987.

[MOO 77] J.A. Moorer, *Signal Processing Aspects of Computer Music: A Survey*, Proceedings of IEEE, Vol 65, Nro 8, August 1.977, p 1108-1137.

[PRO 88] J.G. Proakis, D.G. Manolakis, *Introduction to Digital Signal Processing*, Macmillan Publishing Company, New York, 1.988.

[SER 90] M.H. Serra, D. Rubine, R. Dannenberg, *Analysis and Synthesis of Tones by Spectral Interpolation*, J. Audio Eng. Soc., Vol. 38, Nro. 3, March 1.990, p 111-128.

[STA 88] J.C. Stapleton, S.C. Bass, *Synthesis of Musical Tones Based on the Karhunen-Loève Transform*, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 36, Nro. 3, March 1.988, p 305-319.

[WAW 89] J. Wawrzynek, *VLSI Models for Sound Synthesis*, Current Directions in Computer Music Research, The MIT Press, Massachusetts, 1.989, p 113-148

[WIN 78] S. Winograd, *On Computing the Discrete Fourier Transform*, Math. Comp. 32, 1.978.

[WOM 77] M.E. Womble, J.S. Halliday, S.K. Mitter, M.C. Lancaster, J.H. Triebwasser, *Data Compression for Storing and Transmitting*, Proceedings of the IEEE, Vol 65, Nro 5, May 1.977, p 702-706.

#### Agradecimentos

Ao Departamento de Computação da UNESP-Bauru, pelo afastamento concedido para a realização do programa de doutoramento no DI/UFPE.

À Capes-PICD e CNPq pelo apoio financeiro. Ao Departamento de Informática da UFPE pela utilização dos recursos computacionais.

Ao Dr. Edson Costa de Barros Carvalho Filho (D.I./UFPE) pelo grande incentivo para realização deste artigo.

## The Synthesis of Complex Sonic Events by Functional Iterations

Agostino Di Scipio & Ignazio Prignano

Laboratorio Musica & Sonologia  
Dipartimento di Matematica Pura ed Applicata, Univ. di L'Aquila (Italy)  
e-mail [lms@vxscaq.aquila.infn.it](mailto:lms@vxscaq.aquila.infn.it)

### Abstract

This paper introduces a non-standard sound synthesis method that the authors, with terminology drawn from the literature on chaos theory, call *synthesis by functional iterations*. It describes the general formalism and the application of a difference equation model - the "sin map". Sounds generated with this method can show dynamical properties ranging from very "active" behavior (spectrally rich transient phenomena, turbulence and noise) to relatively "inactive" behavior (smooth, if not almost flat curves), and unpredictable transitions in between.

### Motivations

In computer music research, sound synthesis is a major topic. In short, one can easily recognize two distinct approaches to the design and implementation of sound synthesis methods; here, we refer to them as the "standard" and the "non-standard" approach.

Most research work is usually undertaken in the first kind of approach. It entails the study of one or more acoustic models of some theoretical coherence and the implementation of algorithms capable of reproducing them on the computer. In general, this is the case with most well-known synthesis methods, whether based on linear strategies - additive and subtractive synthesis, plus the related analysis-resynthesis methods - or non-linear transformations of one or more signals - like waveshaping, FM, RM, AM (De Poli, 1981). This is the case for physical modelling, too - an area of major concern in current research (De Poli et al., 1991).

Less has been done in the non-standard approach, although many composers have developed and utilized a variety of methods (e.g. G.M.Koenig, I.Xenakis, and H.Brün). In this perspective, there is no pre-existing acoustic model, while the synthesis process is of the composer's own invention, aiming at the deepest integration between sound synthesis and the compositional process.

If appropriately understood, non-standard methods represent an approach of *microstructural time modelling of sound*, a perspective of sonic design also proper to instances of asynchronous granular synthesis/processing and other techniques based on microstructural representation of sound in the discrete-time domain. There is a bias towards the blurring of the neat distinction between *sound* and *structure* (Truax, 1990a), between the composer's *models of sonic materials* and his/her *models of large scale musical design* (Di Scipio, 1993).

### A fresh perspective of non-standard synthesis

Any particular instance of microstructural time modelling can be seen as the operationalization of a definite music-theoretical problem: how the local organization of myriads of low-level elementary units may bring forth higher-level, global properties of sonic and musical structure, according to a personal strategy of the composer's own invention.

This point, however, has been largely underestimated in previous work in non-standard synthesis. Indeed, the challenge lies precisely here: much like musical form can be understood as the epiphenomenon of a dynamical process captured in a model of musical design (i.e.: at some macro-temporal scale), in computer music the properties of the sonic structure - whose local Gestalt is usually called *timbre* - should themselves be understood as epiphenomena of micro-temporal *compositional* processes, unrelated to acoustic models but capable of modelling a phenomenon of morphological

emergence (Di Scipio, 1994). The problem raises about what may ever be the relevant features in such kind of models.

A possible answer comes from the mathematical modelling of complex systems. In chaos theory, it is shown that simple iterated difference equation systems can capture the details of rich, dynamical phenomena showing peculiar qualitative emergent properties of macrostructure (May, 1977; Collet & Eckmann, 1980). In the following we describe what can be called *synthesis by functional iteration* - drawing from terminology introduced by Mitchell Feigenbaum in an early article on nonlinear systems (1980). The effort consists in exploiting the notion of *iteration* as a source of self-organization in the dynamical behavior of sound.

**Theoretical basis of sound synthesis by functional iterations**

The formal frame of our mathematical model can be described as follows: we shall call

- A ⊂ ℝ the set of "initial values" for our iterations;
- G ⊂ ℝ<sup>m</sup> the set of parameters of the particular map(s) considered;
- B ⊂ ℝ the set of samples of the sound signal finally generated,

and consider the following cartesian product:

$$A \times G \subset \mathbb{R} \times \mathbb{R}^m.$$

Let F be a map defined as

$$F: A \times G \rightarrow B$$

$$(x, \{a_i\}) \rightarrow F(x; \{a_i\}) \quad (\{a_i\} \equiv a_1, a_2, \dots, a_m)$$

which can be considered as a parameter-dependent function which maps from A to B with a<sub>i</sub> as varying parameters. By fixing a set of m real parameters (a point of G) we have:

$$f: A \rightarrow B$$

$$x \rightarrow f(x)$$

$$f(x) \equiv F(x; a_1 \dots a_m).$$

By considering g<sub>i</sub> a sequence of points from G, we get a sequence of maps f<sub>i</sub>. Then, if

$$B \subset A$$

we can construct the iteration of the function f - thereby introducing the operation called *functional iteration* - by repeatedly applying f to itself n times:

$$f^n(x) \equiv f(f(\dots f(x) \dots)) \equiv (f \circ f \circ \dots \circ f)(x)$$

Given a sequence of initial values x<sub>0,i</sub> ∈ A and a sequence of parameter sets g<sub>i</sub> ∈ G and corresponding functions f<sub>i</sub>, a discrete time series can be computed where the i-th sample is the i-th n-iterate of x<sub>0,i</sub>:

$$\begin{aligned} x_{n,i-1} &= f_{i-1}^n(x_{0,i-1}) = f_{i-1}(f_{i-1}(\dots (f_{i-1}(x_{0,i-1})) \dots)) \\ x_{n,i} &= f_i^n(x_{0,i}) = f_i(f_i(\dots (f_i(x_{0,i})) \dots)) \\ x_{n,i+1} &= f_{i+1}^n(x_{0,i+1}) = f_{i+1}(f_{i+1}(\dots (f_{i+1}(x_{0,i+1})) \dots)) \end{aligned}$$

In order to create time series with complex behavior, we must choose a nonlinear f. In the following section, we introduce and study the main properties of one such function. However, it is clear that the relevant point here is the *process* of functional iteration, more than the function we work with: "Yet, precisely because the same operation is reapplied [...] self-consistent patterns might emerge where the consistency is determined by the key notion of iteration and not by the particular function performing the iterates" (Feigenbaum, 1980).

**The "sin map" model**

We have chosen to study the application of the "monoparametric" map (m = 1) defined by

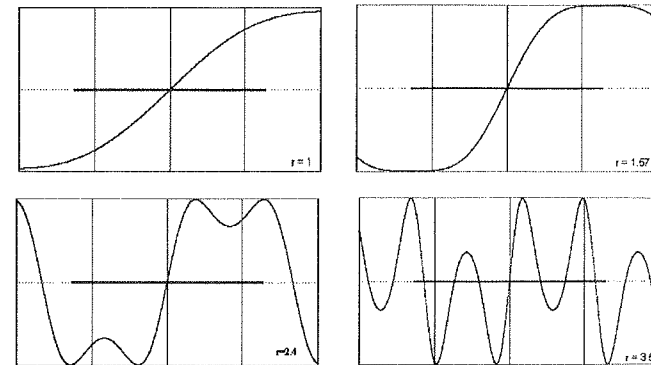
$$F: [-\pi/2, \pi/2] \times [0,4] \rightarrow (-1,1)$$

$$(x, r) \rightarrow \sin(rx)$$

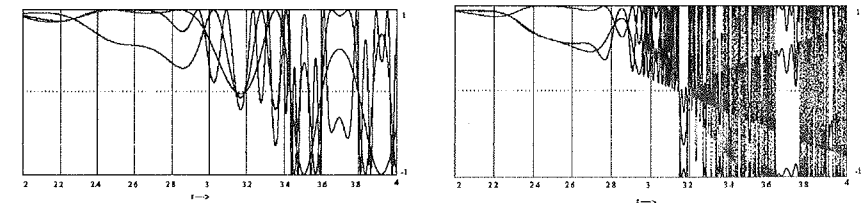
whose explicit iterated form is written as

$$x_{k,i} = \sin(r_i x_{k-1,i})$$

We set A = [-π/2, π/2] because of the fact that, given the periodicity of the sin function, a larger space would only return trajectories already achievable starting from within [-π/2, π/2], with the exception of x<sub>0</sub>, of course. This is because the 1st iterate would anyway fall in the interval [-1,1], completely covered by sin(rx) for x<sub>0</sub> in [-π/2, π/2] and r => 1. Moreover, we set G = [0,4] because any larger value in r would only provide dynamical behaviors already achievable with r just below 4. Following are four examples showing the graphs of sin(rx) with increasing values of r.



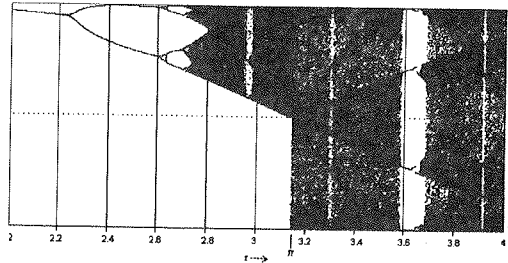
The behavior of the iterated process can be observed by calculating the n-iterate of some x<sub>0</sub> for linearly increasing r. In the leftmost figure, the 5th, 6th and 7th iterates are shown as r moves from 2 to 4; in the rightmost, the 8th, 9th and 10th iterates are plotted. In all cases we set x<sub>0</sub> = 0.1.



Observe that the higher is the iterate and the more dynamical is the evolution of the signal traced by successive n-iterates. To our knowledge, the oscillating trajectories traced by such sequences of n-iterates have not been explicitly addressed by any scientific study. This means that our applications is in need of further theoretical insight into mathematical details which may be unknown as yet. (This is the topic of a forthcoming study).

When we consider a larger number of iterations, the initial datum x<sub>0</sub> is forgotten as the process goes on (one cannot say, by any analytical means, where in the interval [-π/2, π/2] the process started from). Moreover, transient phases to any attractor disappear; indeed, the calculation actually traces the

bifurcation diagram which characterizes the dynamical behavior of the model. Following is a plot of the 100th iterate of  $x_0 = 0.1$  for  $r$  going from 2 to 4.



### A simple implementation

As typical to non-standard methods, the process of synthesis by functional iterations is only conceivable and explorable by implementing a specific algorithmic structure on the computer. The basic procedure is relatively simple and straightforward. However, notice that, since  $n$  is not defined before the synthesis starts (it's up to the user the decision of what iterate is to generate the sound signal) we have a loop of variable length within the computation of each single sample; therefore, for real-time applications we must be able to adjust the computation in order to maintain a stable sampling rate.

An other point worth of being mentioned is that for  $r$  in the interval  $[0, \pi]$  the signal is only positive. We suggest that a conditional control be adopted in order to activate a normalization of the signal in the range  $[-1, 1]$  as far as  $r < \pi$  and to switch to no normalization when  $r > \pi$ . (In most cases this does not produce any noticeable discontinuity, given that  $r = \pi$  is itself a somewhat chaotic area in the bifurcation diagram).

The following is a very simple C program implementing functional iteration synthesis with time-varying  $r$  and constant  $x_0$ . (The normalization problem is not taken into account, here).

```
#include <math.h>
#include <stdio.h>
int cot=32767; /* max amp */
int dur=22050; /* number of output samples */
double nlos(double x, double r, int n); /*prototype*/
void audio(void);
void main()
{
    int k, outsam, n, i_time=0;
    double icr, r_start, r_end, x0, r;
    printf("input start r, end r, start x, considered iterate: \n");
    scanf("%lf %lf %lf %d", &r_start, &r_end, &x0, &n);
    icr=(r_end-r_start)/dur;
    for(r=r_start; r < r_end; r += icr) {
        i_time++;
        outsam=nlos(x0, r, n)*cot;
        printf("sample %d time %d current r %f \n", outsam, i_time, r);
        audio(); /* send to audio output(file) */
    }
}

double nlos(double x, double r, int n) {
    int k;
    for(k=0; k<n; k++) {
        x=sin(r*x);
    } /* end for */
    return(x);
}

void audio(void)
{
    /* audio output code */
}
```

Small modifications in this short piece of code would make it work as an opcode of CSOUND. Actually, the algorithm can be itself rendered with existing CSOUND opcodes, but of course this would make the synthesis slower.

### Exploring the phase space of the "sin map" model

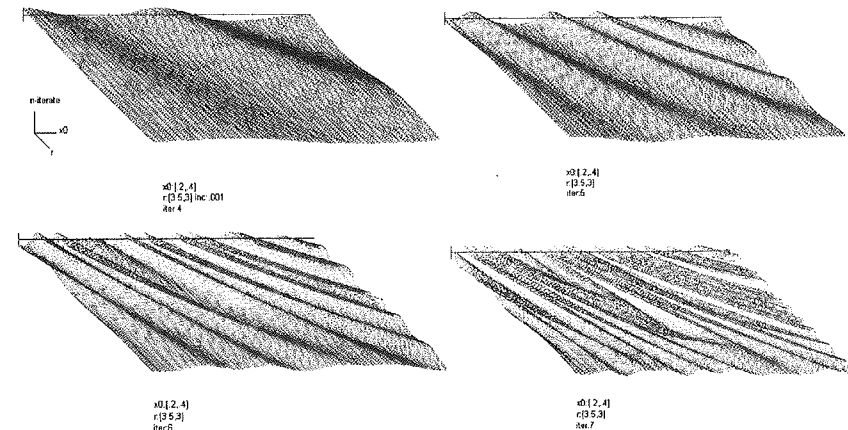
As observable in the bifurcation diagram, moving  $r$  in its space  $[0, 4]$  determines the *kind* of global evolution of the generated signal, ranging from very smooth curves (e.g.  $r = 2$ ) to a more active and complex behavior (e.g.  $r = 4$ ), passing through many transitory phases corresponding to periodical cycles. A most important aspect of signals thus synthesized is an high dependency on the particular region of the phase space  $[-\pi/2, \pi/2] \times [0, 4]$  being visited. As suggested above, each orbit in the space corresponds to a signal of particular properties. In the synthesis process, we can

- change  $r$  while keeping  $x_0$  constant; this is what the C program above does, but we could also
- change  $x_0$  while keeping  $r$  constant; and
- change both  $r$  and  $x_0$  according to some defined driving function.

Moreover, in all these cases we can choose an arbitrary  $n$ -iterate to generate the signal - although, in practice, useful values of  $n$  are bound up to few possibilities (see discussion below). Simply speaking, the three control strategies result respectively in

- rapid changes of global structural properties in the sound signal (highly dynamical spectra);
- innumerable signals of the very same properties (as captured by  $r$ );
- a complex mixture of these two.

Notice, finally, that the higher is the iterate and the more "active" the sound synthesized. Looking at the following 3D graphs, one may grasp some idea of how the sound signal changes as we move in the phase space. Here the  $n$ -iterates of  $f(r, x) = \sin(rx)$  are plotted over a particular region in the phase space  $[-\pi/2, \pi/2] \times [0, 4]$ , namely in the region  $[3, 3.5] \times [2, 4]$ . Each single graph relates to a different iterate, i.e.  $n = 4$ ,  $n = 5$ ,  $n = 6$  and  $n = 7$  (we used such small values not to complicate the graphical rendition too much).



Typically, the morphological properties of sounds generated with functional iteration synthesis are in a flux of continual variation through time, but include both local and global correlation; especially when very "active" ( $r > 3$ ,  $n > 8$ ), they are perceived as textural sonic phenomena, rich of sudden transitions, turbulence and, eventually, broad-band noise. If we use closed orbits in the phase space (by cycling or "modulating" either  $r$  or  $x_0$ ) periodic signals can be obtained. Thus, the sounding results can range from pseudo-random (but locally and globally correlated) sound signals to sounds of harmonic spectra.

Before ending this section, we should recall that *sensitive dependency on the initial conditions* is the essential feature of chaotic systems. A measure that would enable us to characterize such feature in our model - hence the correlation degree within a single signal and among signals - can be computed in terms of the *Lyapunov coefficients* reflecting the exponentially growing distance of trajectories that start

with near but distinct  $x_0$ . In short, such a measure would provide a means to gain control over the predictability of the system behavior, thus making possible the user's meaningful choices of  $n$ . Indeed, in many cases higher iterates will certainly not result into sounds of more interesting structural properties - though they will certainly result in a longer computation time. Sometimes a large  $n$  may even destroy in the signal any observable relation with the signal's generating orbit in the phase space.

#### Some conclusions

Relevant theoretical details and the empirical use of synthesis by functional iteration must be both investigated in more depth. This is necessary, although a most peculiar aspect of such approach to sound synthesis lies exactly in the possibility of an explorative, nonlinear style of sonic design. Indeed, it is dubious that further analytical knowledge would make the model of better use in a linear and completely deterministic approach. Functional iteration synthesis provide *indeterministic* models of sonic material: the composer must learn his/her strategy by interacting with a source of structured information activated at the level of the microstructure of music, within and through the sound.

We think that the concept of *iteration*, being a concrete source of unpredictable but self-organized, consistent behavior, can capture large-scale design features which are particularly useful when one works at the microstructural level of sound. Methods of chaos theory have been already proposed and used as models of syntactical articulation of music (Pressing, 1988) and as powerful control structures of granular synthesis techniques (Di Scipio, 1990; Truax, 1990b). Our study proposes the extension of this approach to the level of the sample itself, by operationalizing a model of sonological emergence which projects the compositional process down to the micro-time scale in the musical structure. In so doing, it also implies a blurring of the conventional distinction between *sound* and *structure*, since with this kind of model the composer can generate entire fabrics of sound events and extended sonic textures that can hardly be perceived and conceived as separate partial components of the musical structure.

#### References

- Collet, P. & Eckmann, J.P (1980) *Iterated Maps on the Interval as Dynamical Systems*. Boston: Birkhauser
- De Poli, G., Piccialli, A. and Roads, C. eds (1991) *Representations of Musical Signals*. Cambridge Ma.: MIT Press
- De Poli, G. (1981) Tecniche Numeriche di Sintesi della Musica. *Quaderni del LIMB*, v.1, 12-45
- Di Scipio, A. (1990) Composition by Exploration of Nonlinear Dynamical Systems. *Proceedings ICMC90*, SanFrancisco: ICMA, 324-327
- Di Scipio, A. (1994) Micro-time Sonic Design and the Formation of Timbre. *Contemporary Music Review* 10(2). In press.
- Di Scipio, A. (1993) Models of Material and of Musical Design Become Inseparable. A Study in Composition-theory. *Proceedings International Conference on Cognitive Musicology*, Jyväskylä: Jyväskylä Yliopisto, 300-316
- Feigenbaum, M (1980) Universal Behavior in Nonlinear Systems. *Los Alamos Science*, 1, 4-27
- May, R. (1977) Simple Mathematical Models with Very Complicated Dynamics. *Nature*, 261, 459-67
- Pressing, J. (1988) Nonlinear Maps as Generators of Musical Design. *Computer Music Journal*, 12(2), 35-46
- Truax, B. (1990a) Composing with Real-time Granular Sound. *Perspectives of New Music*, 28(2), 120-134
- Truax, B. (1990b) Chaotic Nonlinear Systems and Digital Synthesis. An Exploratory Study. *Proceedings ICMC90*, SanFrancisco: ICMA, 100-103

## Modular programming of instruments in cmusic

CARLOS CERANA

Laboratorio de Investigación y Producción Musical (LIPM)

Junín 1930- 1113 Buenos Aires - República Argentina

tel/fax: 54-1-804-0877

email: rcerana@arcriba.edu.ar

#### Abstract

This paper explains a system developed at the Laboratorio de Investigación y Producción Musical (LIPM), for modular programming of instruments in cmusic. The system allows users to build up their own patches using ready-made pieces of code, in a self-explanatory process. Every musician trained in hardware synthesizers programming can easily develop a complex instrument following simple rules, what turns it useful both for composition and for teaching software synthesis. The system allows coherent connections between envelope generators, devices for pitch and amplitude modulation, and for additive, subtractive, FM and waveshaping synthesis. As it is made exclusively of cmusic operators, users can add their own modules to follow their particular needs.

#### What it is

AMI (Aid of Modular Instruments) is an attempt to develop a user-friendly interface for instrument programming in cmusic.

The system allows the user to design complex instruments by adding simple modules. These modules are easily understood by every musician trained in hardware synthesizers programming, and their names are self-explanatory.

#### What it does

In its present state, the system allows coherent connections between envelope generators, devices for pitch and amplitude modulation, for additive, subtractive, FM and waveshaping synthesis, and for spatialization of sound.

#### User interface

Users build up their own patches by assembling modules, that are ready-made pieces of code. The system guarantees the coherence of connections between modules. By following certain simple rules it is easy to develop a complex instrument, because the constitutive parts are already debugged.

The resultant instrument is later invoked from the score by means of a macro.

### What it looks like

A patch designed with AMI is a macro, that is interpreted by the C preprocessor before compiling the cmusic score. The modules that compose the instrument are invoked by other macros, nested in the main one.

An instrument definition in AMI should have the following parts:

1) Header: here is the #define directive, the name of the instrument and its arguments (the parameters that are left open to be controlled from the score).

2) A module (WHEN), that sets the action time for all of them.

3) A module for defining the use of global control for the amplitude.

4) Units for controlling the different aspects of sound (frequency, intensity, spectrum), and their evolution upon time. Some of these modules are specific for a particular synthesis system, others are common to all of them.

5) A module for the audio output.

Let's see an example:

```
#define FMPAIR(beginning,duration,intensity,note)\      <-- header
  WHEN(beginning,duration)\                          <-- action time
  NOGLOBALAMP\                                       <-- global control of amplitude
  NOTREMOLO\                                         <-- tremolo unit
  NOENVPITCH\                                        <-- pitch envelope
  NOVIBRATO\                                         <-- vibrato unit
  NOMODULATOR1\                                     <-- modulation unit (FM)
  ENVMODULATION3S(0,.5,1,.7,.2,.8)\                <-- modulation envelope
  MODULATOR1(3,note,1.7,1)\                         <-- modulation unit (FM)
  ENVAMPLITUDE3S(.3,1,.8,1)\                       <-- amplitude envelope
  CARRIER1(intensity,note,1)\                      <-- carrier (FM)
  POSITION(3,5)\                                       <-- audio output
```

The definition of the instrument FMPAIR tells us that it is not affected by global control of the amplitude, nor uses tremolo, vibrato or pitch envelope units. Nevertheless, all these features must be declared and its place held by *ad hoc* modules (NO-modules).

Other modules show us that the instrument is an FM pair. The MODULATOR1 has a modulation index of 3, and a frequency relation of 1.7 with the carrier. The waveshape of both operators is a sine, set by the value 1 as their last parameters. ENVMODULATION3S and ENVAMPLITUDE3S are the envelope generators for modulator and carrier. They both have three segments of straight-line transitions (that is the meaning of the 3S). The first and last values for the modulation envelope are its relative levels at the beginning and at the end of sound; amplitude envelopes begins and ends in zero (for what it is not explicitly said, and they have fewer arguments).

POSITION is an output module that uses the cmusic SPACE unit to spatialize the sound as it were coming from a fixed source.

The values for *beginning*, *duration*, *intensity* and *note* remain undefined, and are later controlled note by note from the score:

```
FMPAIR(0,4,-16dB,C(1))
FMPAIR(4,2,-12dB,A(0))
```

All cmusic pre and post operators can be used with the system (Moore,1990).

It's important to notice the usage of the NOMODULATOR1 module. It says that there is no previous modulation unit affecting the MODULATOR1 (so the instrument is only an FM pair).

Another more complex example:

```
MORECOMPLEXFM(beginning,intensity,note,veltremolo)\ <-- header
  WHEN(beginning,4)\                                <-- action time
  NOGLOBALAMP\                                       <-- global control of amplitude
  NOVIBRATO\                                         <-- vibrato unit
  NOMODULATOR1\                                     <-- modulation unit (FM)
  ENVPITCH2C(-20Hz,4,.5,0Hz,-5,5Hz)\                <-- pitch envelope
  TREMOLO(veltremolo,8)\                             <-- tremolo unit
  ENVMODULATION2S(.9,.5,.2,.8)\                     <-- modulation envelope
  MODULATOR1(1.5,note,.73,1)\                       <-- modulation unit (FM)
  NOTREMOLO\                                         <-- tremolo unit
  ENVMODULATION3S(.2,4,1,.6,.3,.6)\                 <-- modulation envelope
  MODULATOR1(3,note,1.7,1)\                         <-- modulation unit (FM)
  ENVAMPLITUDE3S(.3,1,.4,.3)\                       <-- amplitude envelope
  CARRIER1(intensity,note,1)\                      <-- carrier (FM)
  MONO\                                              <-- audio output
```

The instrument MORECOMPLEXFM uses cascade FM synthesis. The output of the module MODULATOR1 is taken to feed it again, as the module is invoked twice. It is important to notice that in spite of the fact that the module is the same, its parameters are different, and so are the modulation envelopes that affect it.

Other important feature of the system is shown by the usage of pitch and amplitude modifiers. ENVPITCH2C (pitch envelope generator of two curved segments) affects the two modulators and the carrier. TREMOLO (a very simple unit for amplitude modulation) affects only the first modulator, because its action is turned off by NOTREMOLO before the appearance of the second modulator and the carrier.

### How it works

The central concept of AMI is the idea of small, incomplete instruments, whose outputs are blocks going nowhere. The values passed to those blocks are taken by the following partial instruments, which have open inputs coming from nowhere. After processing the data, the output is again "hanged in the air" and taken by the next operator, and so on until the last module of the instrument.

Playing a note in an instrument designed with AMI means to play a chord with all the modules that compound it, but only the one with the audio output actually "sounds".

The whole system relies on the coherence of naming the input-output blocks. To prevent the unwanted appearance of remaining data in a block (that could be left by other instrument playing at the same time), I introduced the NO-modules. Their sole mission is to clear a particular block, to be sure that its value is neutral to the processing.

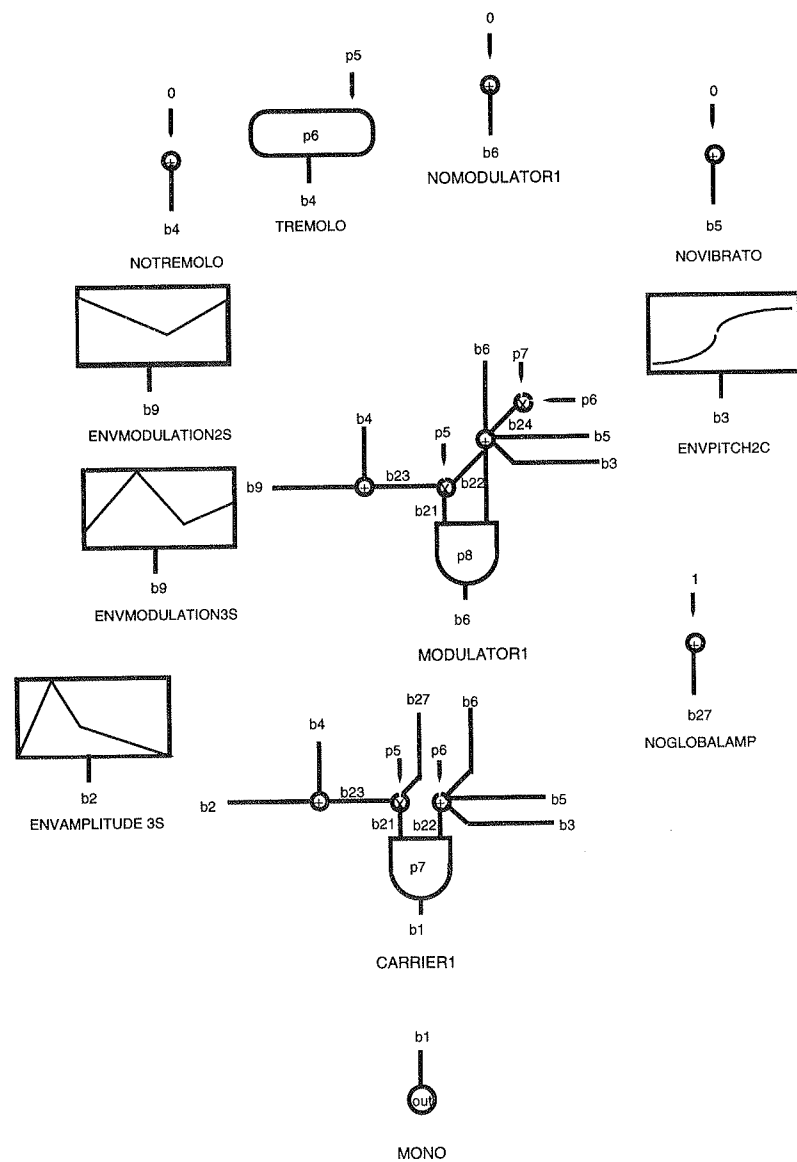
### Elements of the system

The modules are macros that play a note in one of the incomplete instruments mentioned above, which themselves are defined using cmusic syntax.

The following two examples will enlighten the subject:

#### Example 1

```
ins 0 carrier 1 ;
  adn b22 p6 b3 b5 b6 ;
  adn b23 b2 b4 ;
  mult b21 p5 b23 b27 ;
  osc b1 b21 b22 p7 d ;
end ;
```



```
#define CARRIER1(intensity,note,waveshape)\
note p2 carrier1 p4 intensity note waveshape ;
```

The macro CARRIER1 invokes the instrument carrier1, and plays a note in it at the moment *beginning* (parameter 2), lasting for *duration* (parameter 4). These two parameters are supposed to be set by the macro WHEN (that's why it must appear as the first module in the definition of the patch).

Example 2

```
ins 0 nil4;\
  adn b4 0 ;\
  end ;
```

```
#define NOTREMOLO\
note p2 nil4 p4 ;
```

The nil4 instrument cleans the block 4, by feeding an additive operator with a zero. Then it is invoked by the macro NOTREMOLO to play a note.

Structure of modules used in the MORECOMPLEXFM instrument are displayed in the figure. The TREMOLO unit is not analyzed here because it itself is composed by modules, and its complete discussion lies beyond the scope of the present paper.

Further developments

The system could easily be expanded to follow everyone's particular needs. As it is made exclusively of cmusic operators, it is possible for users to add their own modules. The only need is to preserve the coherence in naming the input-output blocks, as it was done among the existing modules.

There is also a project for developing a graphic interface for the system, that could result in a significant improvement.

Conclusions

AMI for cmusic allows a comfortable approach to software synthesis, that makes it useful for educational purposes, as well as for composition. The system is easy to deal with for musicians experienced in hardware synthesizers, since its modules remind the operators of such devices.

The primary concern of the system is friendliness and clarity, rather than computational efficiency. I consider that for anybody at the first stages of contact with computer music languages, the waste of time in debugging is much more important than the one in compiling.

References

Cerana, Carlos (1994). *AMI: Ayuda mediante Módulos de Instrumentos para cmusic*. Buenos Aires: LIPM.  
 Moore, F. Richard (1990). *Elements of Computer Music*. Englewood Cliffs, NJ: Prentice-Hall.

### Acknowledgments

The development of the system described in this paper was initiated as a result of a residence program between CRCA (University of California, San Diego), CCRMA (Stanford University) and LIPM, sponsored by the Rockefeller Foundation.

I wish to thank Tim Labor, who patiently tutored my first steps in cmusic, and F. Richard Moore, for their clever and opportune advice and encouragement.

## On the Improvement of the Real-Time Performance of Two Fundamental Frequency Recognition Algorithms

ANDREW CHOI

*Department of Computer Science  
University of Hong Kong  
Pokfulam Road, Hong Kong*

### Abstract

Many existing fundamental frequency recognition (FFR) algorithms return reliable results when the analysis window is sufficiently wide. In some applications, however, the response time, i.e., the sum of the width of the analysis window and the computation time for the FFR algorithm, must be made as short as possible. This paper studies the effect of window width on the accuracy of two FFR algorithms and describes a new algorithm with improved accuracy for narrow analysis windows. The new algorithm uses dynamic programming to match harmonics to peaks in the constant- $Q$  transform of the signal. A modification to another FFR algorithm that enhances its performance in real time is also considered.

### Introduction

A pitched musical sound is composed chiefly of harmonic components whose frequencies are integral multiples of a fundamental frequency. The problem of fundamental frequency recognition (FFR) is encountered in the automatic analysis of these signals, such as in pitch-to-MIDI systems that enable acoustic instruments to be used as controllers of digital synthesizers.

FFR algorithms that operate in the frequency domain perform spectral analysis on the signal by segments and apply a pattern matching technique to the spectrum to determine each segment's fundamental frequency. Amuedo (1985), for example, identifies sinusoidal components in a signal by the peaks in the power spectrum and examines how the hypothesis for each component to be the fundamental frequency is reinforced by the other components. Pearson and Wilson (1990) consider a multiresolution approach for the spectral analysis step. Doval and Rodet (1991a, 1991b) apply a maximum likelihood analysis to determine the fundamental frequency also using peaks in the power spectrum. Brown (1992) computes the cross-correlation of the constant- $Q$  transform of a segment of the signal with a fixed comb pattern. The calculation of the constant- $Q$  transform and a fast algorithm for approximating it are considered in (Brown 1991) and (Brown and Puckette 1992), respectively.

An alternative approach for designing FFR algorithms is based on computing an autocorrelation between the waveform and a delayed version of itself and determining the fundamental frequency by maximizing the degree of their similarity. Ney (1982) uses time-warping to account for small variations in the signal waveform. The estimated period is the amount of shift that results in the best match of a segment of the signal with a future segment. Lane (1990) adapts the center frequency of a bandpass filter to match the fundamental frequency of the signal using a convergence algorithm. Cook et al. (1993) use a least mean square adaptive algorithm to determine the coefficients of a filter that predicts a segment of a signal from an earlier segment. The phase of the filter is computed from these coefficients, which is then used to estimate the period. Another technique, described in (Brown and Puckette 1993), first determines a coarse estimate of the fundamental frequency using a frequency-domain algorithm. The phase change of the component closest in frequency to the coarse estimate between two segments of the signal separated by one sample is then used to estimate the fundamental frequency accurately.

Accuracy is an important measure of performance of an FFR algorithm. In applications where synthesizers with continuous pitch are controlled, the resolution at which the FFR algorithm can distinguish

frequencies is also an important measure. An error of a few percent in frequency is perceivable in many musical sounds. Frequency-domain FFR algorithms have lower recognition resolution inherently since their spectral analysis step subdivides the frequency range into bins. The disadvantage of autocorrelation algorithms, however, is that a good initial estimate of the fundamental frequency is required for them to converge. Hybrid approaches combine the strengths of both types of algorithms (Kuhn 1990; Brown and Puckette 1993).

When these algorithms are applied to real-time pitch-to-MIDI controllers, another important performance measure is the response time of the system, which is equal to the sum of the width of the analysis window and the computation time for the FFR algorithm. Ideally this response time should be so short that it is not perceived by the performer. Economic and engineering constraints have resulted in commercial systems whose response times are over 50 milliseconds for notes with an average pitch, and even longer for low-pitch notes (Cook et al. 1993). This paper studies the effect of window width on the accuracy of the frequency-domain algorithm in (Brown 1992) and the autocorrelation algorithm in (Cook et al. 1993) and describes techniques for improving their accuracy.

### The Constant-Q Transform/cross-correlation Algorithm

The FFR algorithm introduced by Brown (1992) operates in two steps: computation of the constant-Q transform of a segment of the signal, and cross-correlation of the constant-Q transform with a fixed comb pattern that has the logarithmic-scale spacing of the harmonics. The constant-Q transform of a sequence  $x[i]$  is defined by

$$X[f] = 1/N_f \sum_{i=0}^{N_f-1} W(N_f, i) x[i] e^{-j2\pi Qi/N_f},$$

where  $Q = 1/(\sqrt[2]{2} - 1)$  is the quality factor of the transform,  $N_f = S/(2^{f/d} f_{min})$  is the number of samples of the signal that need to be analyzed for frequency bin  $f$ , and  $W$  is the Hamming window whose width has been adjusted by  $N_f$ , given by  $W(N_f, i) = \alpha - (1 - \alpha)\cos(2\pi i/N_f)$ ,  $\alpha = 25/46$ . For real-time operation, and for smaller values of  $f$ ,  $N_f$  may in fact be greater than the number of samples in the signal being analyzed. In these cases,  $X[f]$  is computed using only the available samples. The values  $d$ , the number of bins to subdivide each octave,  $S$ , the sampling rate, and  $f_{min}$ , the center frequency of the bin with lowest frequency, are parameters of the algorithm.

The constant-Q transform extracts the frequency components of the signal  $x[i]$  in logarithmic frequency spacing, where bin  $f$  corresponds to the component with center frequency  $2^{f/d} f_{min}$ . Since the harmonics of a signal with fundamental frequency  $f_0$  have frequencies  $f_0, 2f_0, 3f_0, \dots$ , and so on, the spacing between harmonics in the constant-Q transform  $X[f]$  is fixed and independent of the value of  $f_0$ . Thus to correlate the harmonics, a cross-correlation (i.e., convolution) is computed between  $X[f]$  and the pattern

$$\underbrace{\underbrace{\underbrace{1, 0, \dots, 0, 1, 0, \dots, 0, 1, 0, \dots, 0, 1, 0, \dots}_{d \log_2 3}}_{d \log_2 4} \dots}_{d}$$

The number of harmonics to use in the pattern is also a parameter of the algorithm. The center frequency of the bin with the highest cross-correlation value is then returned by the algorithm as the estimate of the fundamental frequency of the signal. To illustrate this algorithm, the constant-Q transform of a 30-millisecond initial segment of a C4 (261.6Hz) note sampled from an electric guitar, the comb pattern, and the cross-correlation are shown in figure 1.

A set of experiments was conducted to study the real-time performance of this algorithm. We implemented and tested this algorithm with the 10-, 15-, 20-, and 30-millisecond initial segments of a set of notes sampled from an electric guitar taken from the range G2 (98.0 Hz) to C6 (1046.5 Hz). The results are shown in table 1. The algorithm correctly recognizes the 30-millisecond initial segments of all notes and fails to recognize most of the initial segments of notes at or below C3 which have length 20 milliseconds or below. It also fails for 10-millisecond initial segments of notes at or below C4. This

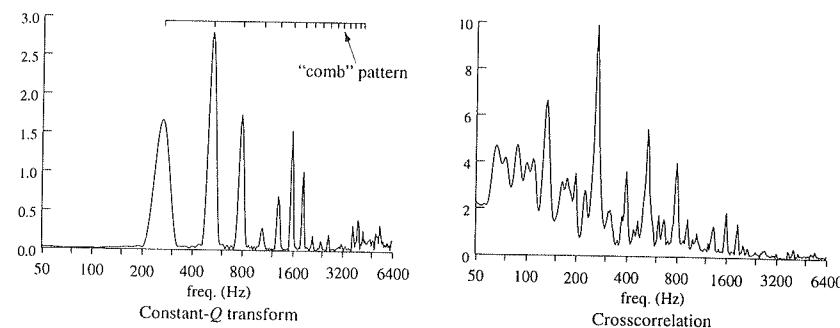


Figure 1: Constant-Q transform and cross-correlation for a 30ms segment of a C4 note.

	G2 (98.0)	C3 (130.8)	G3 (196.0)	C4 (261.6)	G4 (392.0)	C5 (523.3)	G5 (784.0)	C6 (1046.5)
10ms	<b>50.0</b>	<b>59.5</b>	<b>80.5</b>	105.9	400.0	526.3	788.5	1052.6
15ms	101.5	<b>53.7</b>	197.1	263.1	394.3	526.3	788.5	1052.6
20ms	<b>50.0</b>	<b>52.2</b>	197.1	263.1	394.3	526.3	788.5	1052.6
30ms	100.0	131.6	197.1	263.1	394.3	526.3	788.5	1052.6

Table 1: Results of constant-Q transform/cross-correlation algorithm (actual frequencies (in Hz) are shown in parentheses beneath note names; incorrect results are shown in bold type).

failure is a result of the inability of the constant-Q transform to distinguish neighboring frequencies when the analyzed signal is short and has a low fundamental frequency. The frequency contents of a bin spill over into neighboring bins causing the cross-correlation step to fail. Such a situation is shown in figure 2 for a 10-millisecond initial segment of a C4 note.

### A New Constant-Q Transform/Dynamic Programming Algorithm

The new algorithm is motivated by noticing that although peaks in the constant-Q transforms of problematic cases have broader side lobes, their relative positions remain quite stable. This suggests that higher accuracy can be achieved by replacing the cross-correlation stage by a peak detector followed by an algorithm that matches the peaks to harmonics. In this sense, the new algorithm is a generalization of the one in (Amuedo 1985). Since some detected peaks may be extraneous and peaks corresponding to some harmonics may be missing, a "time-warping" algorithm is devised to match the peaks to the harmonics in a manner that minimizes a total error measure.

Peaks are first identified in the constant-Q transform of the signal segment. To prevent excessive extraneous peaks, ones with small amplitudes are ignored. Examples of such extraneous peaks appear between 600Hz and 700Hz in figure 1. The algorithm identifies and uses the  $p$  peaks with the lowest frequencies. Let these peaks have frequencies  $f_1, f_2, \dots, f_p$  and amplitudes  $a_1, a_2, \dots, a_p$ , respectively. Also let  $h$  be the number of harmonics considered. The values  $p$  and  $h$  are parameters of the algorithm, chosen to be 10 and 8, respectively, in the experiments described below. Since some peaks as well as some harmonics should be skipped, a matching of peaks to harmonics is represented by a sequence of pairs  $(i_1, j_1), (i_2, j_2), \dots, (i_n, j_n)$ , where  $1 \leq i_1 < i_2 < \dots < i_n \leq p$ ,  $1 \leq j_1 < j_2 < \dots < j_n \leq h$ , and  $(i_k, j_k) = (i_{k-1} + 1, j_{k-1} + 1), (i_{k-1} + 1, j_{k-1} + 2),$  or  $(i_{k-1} + 2, j_{k-1} + 1)$ . The last condition ensures that only a single peak or harmonic is skipped at a time. The boundary conditions are  $(i_1, j_1) = (1, 1), (1, 2),$  or  $(2, 1)$ , and  $(i_n, j_n) = (p, h), (p-1, h),$  or  $(p, h-1)$ . The problem is then one of finding, among all possible such sequences of pairs, a sequence of pairs that minimizes the error measure  $E = \sum_{k=1}^n e(i_k, j_k)$ , where  $e(i_k, j_k)$  is the error of matching the  $i_k$ -th peak to the  $j_k$ -th harmonic.



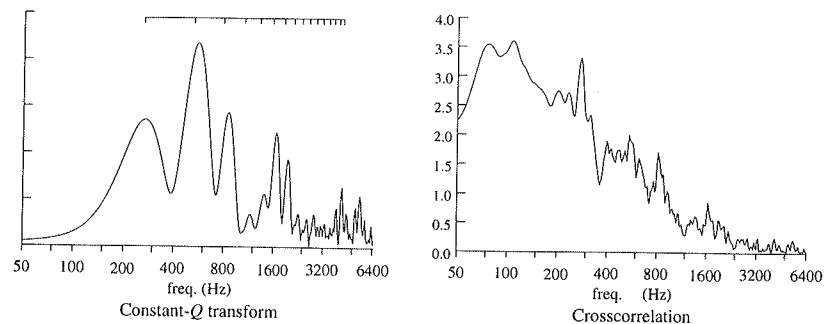


Figure 2: Constant- $Q$  transform and cross-correlation for a 10ms segment of a C4 note.

To formulate the problem so that it can be solved by dynamic programming, let  $e(i_1, j_1) = 0$  and let  $e(i_k, j_k)$  depend only on the first  $k$  pairs of a sequence, i.e.,  $(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)$ . After some experimentation, we arrive at the following definition of the error function  $e$ . The matching of the  $i$ -th peak to the  $j$ -th harmonic, represented by the pair  $(i, j)$ , suggests  $f_i/j$  as an estimate of the fundamental frequency. Let  $\hat{f}_{k-1}$  be the weighted average of the estimates of the fundamental frequency generated by the first  $k-1$  pairs, where the weights are the amplitudes of the corresponding peaks. The assumption is that peaks with larger amplitudes should have a greater effect on the final estimate of the fundamental frequency. Thus define  $\hat{f}_{k-1} = \sum_{l=1}^{k-1} (a_{il} f_{il} / j_l) / \sum_{l=1}^{k-1} a_{il}$ . The error function is then defined as  $e(i_k, j_k) = (f_{ik} / j_k - \hat{f}_{k-1})^2$ .

The recurrence formulas for implementing a dynamic programming algorithm to determine the sequence of pairs with minimum error are derived as follows. Let

$E_{i,j}$  be the error of a sequence of pairs that optimally matches the first  $i$  harmonics to the first  $j$  peaks,

$F_{i,j}$  be the fundamental frequency estimated by this sequence, and

$A_{i,j}$  be the accumulated sum of amplitudes of peaks used in this sequence.

Then,  $E_{i,j}$  is computed by the equation

$$E_{i,j} = \min\{E_{i-1,j-1} + (f_i/j - F_{i-1,j-1})^2, E_{i-2,j-1} + (f_i/j - F_{i-2,j-1})^2, E_{i-1,j-2} + (f_i/j - F_{i-1,j-2})^2\}.$$

The equations

$$F_{i,j} = \begin{cases} F_{i-1,j-1} A_{i-1,j-1} / (A_{i-1,j-1} + a_i) + (f_i/j) a_i / (A_{i-1,j-1} + a_i) \\ F_{i-2,j-1} A_{i-2,j-1} / (A_{i-2,j-1} + a_i) + (f_i/j) a_i / (A_{i-2,j-1} + a_i) \\ F_{i-1,j-2} A_{i-1,j-2} / (A_{i-1,j-2} + a_i) + (f_i/j) a_i / (A_{i-1,j-2} + a_i) \end{cases} \quad \text{and} \\ A_{i,j} = \begin{cases} A_{i-1,j-1} + a_i \\ A_{i-2,j-1} + a_i \\ A_{i-1,j-2} + a_i \end{cases}$$

are used to update  $F_{i,j}$  and  $A_{i,j}$ , respectively. Whether the first, second, or third expressions in these two equations are used depends on which term within the braces in the equation for  $E_{i,j}$  has the minimum value. Initially, let  $E_{1,1} = E_{1,2} = E_{2,1} = 0$  and  $E_{i,1} = E_{1,j} = \infty$  for  $i > 2$  and  $j > 2$ . The values of the tables for  $E_{i,j}$ ,  $F_{i,j}$ , and  $A_{i,j}$  can be updated in either column order or row order. References to values outside the range of the indices are assumed to return arbitrarily large values. The final estimate of the fundamental frequency returned by the algorithm is  $F_{p,h}$ ,  $F_{p-1,h}$ , or  $F_{p,h-1}$  depending on which of  $E_{p,h}$ ,  $E_{p-1,h}$ , and  $E_{p,h-1}$  has the smallest value.

	G2 (98.0)	C3 (130.8)	G3 (196.0)	C4 (261.6)	G4 (392.0)	C5 (523.3)	G5 (784.0)	C6 (1046.5)
10ms	<b>207.3</b>	<b>243.2</b>	201.2	269.1	400.6	527.2	793.2	1052.0
15ms	101.5	134.7	199.3	248.8	396.5	527.0	788.1	1052.0
20ms	102.2	133.5	197.2	263.0	395.1	525.5	788.1	1052.6
30ms	100.0	131.9	196.7	263.0	393.8	525.5	788.5	1052.6

Table 2: Results of constant- $Q$  transform/dynamic programming algorithm (actual frequencies in parentheses; incorrect results in bold type).

The new algorithm was tested on the same set of initial segments of notes, and its results are shown in table 2. It performs better than the algorithm based on cross-correlation, and correctly recognizes the 15-, 20-, and 30-millisecond initial segments of all notes. It also correctly recognizes all 10-millisecond initial segments of notes at or above G3.

### Real-Time Considerations for the Periodic Predictor Pitch Tracker

Given an initial estimate of the fundamental period, the periodic predictor pitch tracker (PPPT) of Cook et al. (1993) computes a set of predictor coefficients for the signal using an iterative least mean square (LMS) algorithm and uses them to refine the estimate of the fundamental period. Let the signal be given by the sequence  $x_0, x_1, \dots$  and the initial estimate of the fundamental period be  $P$ . Let there be  $2M+1$  predictor coefficients  $c_{-M}, c_{-M+1}, \dots, c_M$ . The predictor predicts the  $i$ -th sample from the  $2M+1$  samples centered around the  $(i-P)$ -th sample using the equation  $\hat{x}_i = \sum_{j=-M}^M c_j x_{i-P+j}$ . The error of this prediction is  $\epsilon_i = x_i - \hat{x}_i$ . For a given signal, an approximation of the set of predictor coefficients that minimize the mean square error over the prediction of the  $N$  consecutive sample values  $\hat{x}_{M+P}, \hat{x}_{M+P+1}, \dots, \hat{x}_{M+P+N}$  can be obtained by iterating the LMS update equations

$$c'_j = c_j + \alpha / (2M+1) \bar{x}^2 x_{i-P+j} \epsilon_i,$$

for  $j = -M, -M+1, \dots, M$ , over the  $N$  predictions  $i = M+P, M+P+1, \dots, M+P+N$ . The parameter  $\alpha$  is any positive number less than 1 and  $\bar{x}^2 = 1/R \sum_{i=0}^{R-1} x_i^2$ ,  $R = M+N+P+1$ , is the signal power. To perform this operation, the length of the signal must be at least  $R$ . Having obtained the predictor coefficients, a more accurate fundamental period estimate is given by

$$P' = P(1 - \theta/2\pi),$$

where

$$\theta = \arctan \left( \frac{\sum_{j=-M}^M c_j \sin(\omega j)}{\sum_{j=-M}^M c_j \cos(\omega j)} \right)$$

and  $\omega = 2\pi/P$ .

Cook et al. (1993) suggest using the PPPT in real time by supplying samples to it continuously, i.e., in the above formulation, letting  $N = \infty$ . The fundamental period estimate will then converge to an accurate value a certain time after the beginning of a note. They report the average of this latency to be 30.1 milliseconds for notes between F5 (698.5Hz) and G6 (1568.0Hz), which are tested in their experiments. We implemented and tested the PPPT on the same set of initial segments of notes used in the previous experiments. However, it does not converge to accurate frequency estimates during the duration of most segments of notes, especially for short, low-pitch ones. The algorithm is then modified to choose the largest possible value of  $N$  for a signal segment of a given length and iterate a number of times over that segment, allowing sufficient time for the predictor coefficients to converge. Table 3 shows the recognition results of the modified algorithm. In these experiments,  $M$  is chosen to be 2, and the initial estimate of the fundamental period is taken to be that of a semitone higher than the note to be recognized. This latter assumption can be satisfied if the PPPT is used as a postprocessing step of the dynamic programming FFR algorithm described in the previous section. The algorithm is set to iterate 50 times over a signal segment. The modified algorithm is found to correctly converge to the fundamental frequencies for all initial segments of notes except in two cases. It cannot be used for

	G2 (98.0)	C3 (130.8)	G3 (196.0)	C4 (261.6)	G4 (392.0)	C5 (523.3)	G5 (784.0)	C6 (1046.5)
10ms	-	<b>139.1</b>	193.9	258.5	392.0	525.0	787.5	1052.1
15ms	99.0	131.0	195.1	261.2	394.5	524.1	786.4	1052.3
20ms	98.9	131.4	195.1	261.8	393.7	523.5	785.6	1052.3
30ms	98.9	131.4	195.2	261.8	393.7	523.9	785.6	1052.1

Table 3: Results of PPPT algorithm (incorrect results for 10ms C3; not enough samples to run 10ms G2 case).

the 10-millisecond segment of G2 because the fundamental period of that note is 227.0 samples and the number of samples in the segment is 223 (at a sampling rate of 22255Hz). The algorithm also converges to an incorrect fundamental frequency for the 10-millisecond segment of C3. Furthermore, note that the frequency estimates generated by this algorithm are closer to the actual frequencies than the two frequency-domain algorithms.

### Summary

This paper reports experiments that show how the accuracy of the FFR described in Brown (1992) is affected by different window widths. It then proposes a new FFR algorithm with higher accuracy for narrow analysis windows. It also describes a modification to the PPPT algorithm of Cook et al. for improved operation in real time.

### References

- Amuedo, J. (1985). Periodicity estimation by hypothesis-directed search. In *Proc. of ICASSP '85*, (Tampa, Florida, May 1985), 395-398.
- Brown, J.C. (1991). Calculation of a constant Q spectral transform. *J. Acoust. Soc. Am.* v. 89, no. 1, (Jan. 1991), pp. 425-434.
- Brown, J.C. (1992). Musical fundamental frequency tracking using a pattern recognition method. *J. Acoust. Soc. Am.* v. 92, no. 3, (Sep. 1992), pp. 1394-1402.
- Brown, J.C. and Puckette, M.S. (1992). An efficient algorithm for the calculation of a constant Q transform. *J. Acoust. Soc. Am.* v. 92, no. 5, (Nov. 1992), pp. 2698-2701.
- Brown, J.C. and Puckette, M.S. (1993). A high resolution fundamental frequency determination based on phase changes of the Fourier transform. *J. Acoust. Soc. Am.* v. 94, no. 2, Pt. 1, (Aug. 1993), pp. 662-667.
- Cook, P.R., Morrill, D., and Smith, J.O. (1993). A MIDI control and performance system for brass instruments. In *Proc. of ICMC*, (Tokyo, Japan, 1993), 130-133.
- Doval, B. and Rodet, X. (1991a). Fundamental frequency estimation using a new harmonic matching method. In *Proc. of ICMC*, (Montreal, Canada, 1991), 555-558.
- Doval, B. and Rodet, X. (1991b). Estimation of fundamental frequency of musical sound signals. In *Proc. of ICASSP '91*, (Toronto, Canada, May 1991), 3657-3660.
- Kuhn, W.B. (1990). A real-time pitch recognition algorithm for music applications. *Computer Music Journal*, v. 14, no. 3, (Fall 1990), pp. 60-71.
- Lane, J.E. (1990). Pitch detection using a tunable IIR filter. *Computer Music Journal*, v. 14, no. 3, (Fall 1990), pp. 46-59.
- Ney, H. (1982). A time warping approach to fundamental period estimation. *IEEE Trans. on Systems, Man, and Cybernetics*, v. SMC-12, no. 3, (May/June 1982), pp. 383-388.
- Pearson, E.R.S. and Wilson, R.G. (1990). Musical event detection from audio signals within a multiresolution framework. In *Proc. of ICMC*, (Glasgow, 1990), 156-158.

## A Linguagem SOM-A para Síntese Aditiva

ALUIZIO ARCELA  
 Laboratório de Processamento Espectral  
 Departamento de Ciência da Computação  
 Universidade de Brasília  
 Brasília, DF — CEP 70910-900  
 e-mail: arcela@lpe1.cic.unb.br

### Resumo

Descrição formal da sintaxe, do universo semântico, dos operadores e de alguns aspectos de implementação da linguagem SOM-A. Com o interpretador desta linguagem, que é voltada exclusivamente para a síntese aditiva de sinais musicais, executam-se partituras polifônicas associadas a orquestras, para as quais se definem, uma a uma, as componentes espectrais em todos os seus parâmetros, isto é, ordem de frequência, ângulo inicial de fase, curva de envoltória, e grau de estereofonia.

### HISTÓRICO

Um primeiro esboço para a linguagem SOM-A surgiu em 1986 a partir de algumas experiências com o programa Music V (Mathews et al. 1969). Tais experiências giravam em torno da tentativa de se interpretarem composições algorítmicas baseadas nas árvores de tempos (Arcela 1986), as quais, na maioria das vezes por força do modelo, possuíam uma grande quantidade de componentes espectrais, e eram caracterizadas, do ponto de vista da escrita de eventos e mecanismos espectrais, por não haver nelas, ao menos aparentemente, qualquer possibilidade ou indício de serem interpretadas por um processo que não fosse o da síntese aditiva. Além disso, registrava-se nessas composições a exigência de uma certa projeção acústica estereofônica, segundo a qual um subconjunto bem definido do universo de componentes espectrais deveria soar em apenas um dos canais, enquanto as demais componentes soariam no outro canal, exigência esta que trazia uma ligeira alteração na arquitetura do instrumento mínimo necessário à síntese aditiva padrão, isto é, a que se registra na literatura, como em (Moorer 1977). As estruturas de dados estáticas da implementação FORTRAN do sistema Music V permitiam apenas a execução de instrumentos possuídores de relativamente poucas componentes. E como havia a intenção de se perfazer uma interpretação plena dessas estruturas musicais em todas as suas partes, sem que houvesse qualquer truncamento no conjunto de componentes, a idéia que restava era investir na definição de uma linguagem que pudesse aceitar um instrumento de qualquer tamanho, e que tivesse uma única especialidade: a síntese aditiva. E assim, com apenas 5 operadores, e com uma sintaxe semelhante a de LISP — os instrumentos e os eventos espectrais devem ser escritos rigorosamente na forma de expressões simbólicas —, surgiu a linguagem SOM-A, assumindo a simplicidade como sua característica maior.

Como não podia ser de outra forma, a primeira implementação ocorreu em LISP (Nogueira Filho 1988), que era sem dúvida a atitude mais correta e natural, levando-se em conta a natureza das estruturas de dados escolhidas para representar os elementos de SOM-A. Naquela época, o que de melhor havia para o sistema operacional MSDOS era o interpretador muLISP87 (Mulisp 1987), um ambiente satisfatório em muitos aspectos, mas oferecendo como obstáculo crucial à implementação de programas voltados para a síntese de sinais de áudio justamente a sua inevitável execução de operações aritméticas totalmente por software. Às vezes, para se interpretar um trecho de 10s, dependendo da complexidade orquestral, eram gastas entre 10 e 20 horas de processamento em intel286 a 10 MHz. Há o caso de uma peça de 1'20" contendo instrumentos de até 76 componentes espectrais para a qual foram necessários 25 dias ininterruptos de processamento! Mesmo assim, a implementação de SOM-A em LISP para pequenas máquinas assumiu uma importância capital no desenvolvimento da linguagem, pois, se não consegue executar peças de conteúdo espectral denso em um prazo razoável, ela vem cumprindo o papel de uma especificação

viva desta linguagem para síntese aditiva, a partir da qual são levadas a cabo outras implementações através de linguagens mais ágeis no tocante a operações aritméticas, como a implementação em C++ para MS-Windows (Castro 1994), que apresenta um bom desempenho — a tal peça de 25 dias é aí concluída em cerca de 2 horas em processador intel486 a 66MHz —, além de oferecer uma interface MS-Windows de boa funcionalidade em diversos itens, como o acompanhamento visual das formas de onda à medida que o sinal vai sendo produzido, a manipulação de três formatos para arquivos de amostras (wav, voc, str), e o acionamento de placas de som (atualmente, USE e SoundBlaster 16) ao final do processo ou em seguida a uma interrupção do usuário. Está em curso o transporte desta versão em C para uma máquina Sun (Meireles et. al.). Vale ainda registrar que, entre o trabalho inicial feito com muLISP e este último em Borland C, houve uma implementação cuidadosa em Sun Common LISP para ambiente UNIX (Ramalho 1991), cujo resultado apresentou um desempenho bastante superior ao que se consegue com o LISP para DOS, principalmente em termos de interface e velocidade. Havia, contudo, uma expectativa por um rendimento bem melhor, levando-se em conta que em Common LISP as variáveis numéricas podem ser tipificadas em ponto flutuante, o que torna as operações aritméticas envolvidas em SOM-A totalmente executáveis por hardware.

Em todas esses projetos foram adotadas técnicas adequadas para gerenciamento de memória virtual para que as orquestras possuidoras de um grande número de componentes não tivessem problemas de espaço na máquina a ponto de se ver impedida a execução de uma dada partitura. O emprego desta técnica segue o já mencionado preceito de implementação SOM-A, qual seja, o de não haver limites para a orquestra quanto ao maior número possível de unidades-H, nem quanto ao número de notas simultâneas endereçadas a essa orquestra, a não ser os últimos limites de espaço impostos pela máquina física que abriga uma implementação da linguagem.

### OS ELEMENTOS

Unidade-H é a denominação do mecanismo responsável pela geração de uma componente espectral no contexto da linguagem SOM-A. Especificamente, unidade-H é o algoritmo composto por um oscilador senoidal a tabela, cuja amplitude é controlada por um gerador de envoltória, e cuja entrada de frequência é multiplicada por um valor denominado ordem de frequência, conforme ilustra a Figura 1. A saída de uma unidade-H poderá ser dirigida a um canal de áudio x, ou a um canal y, dependendo do valor do parâmetro de estereofonia. Construir um instrumento musical em SOM-A significa especificar um agrupamento de várias dessas unidades em configuração de síntese aditiva, conforme a Figura 2, de modo que todas as saídas parciais serão somadas para cada canal de áudio, e haverá uma única entrada de frequência e uma única entrada de amplitude.

Constrói-se um instrumento SOM-A a partir de um número qualquer de componentes, todas elas possuindo um comportamento espectral bem definido e individualizado. Para se executar uma nota nesse instrumento, são

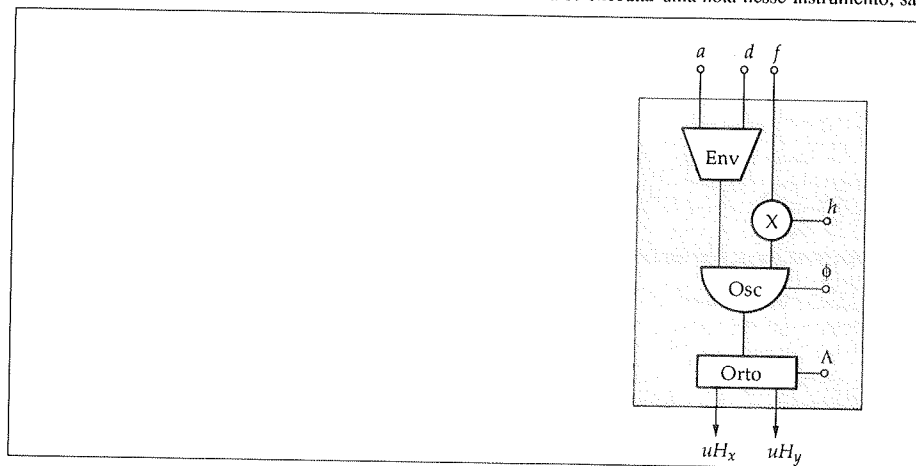


Fig. 1 A unidade-H

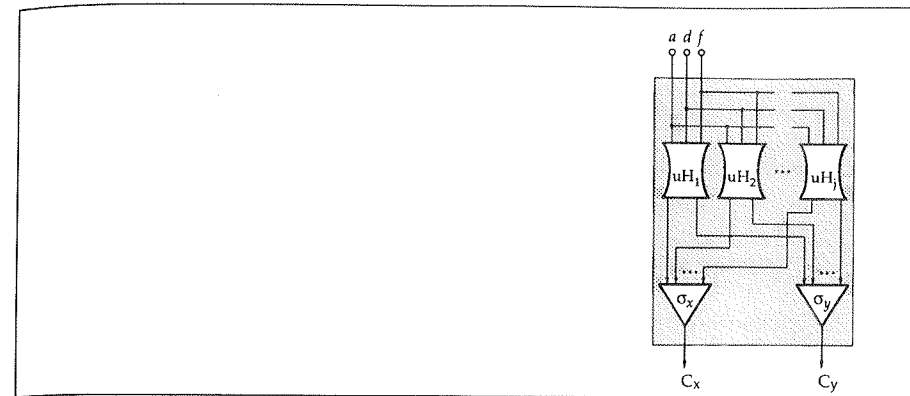


Fig. 2 Um instrumento SOM-A

utilizados na partitura comandos de notas que, na sua apresentação mais simples, fazem menção a toda a população de unidades-H do instrumento, porém, quando escrito de uma forma mais específica, solicita ao instrumento que acione seletivamente um bem definido subconjunto de unidades. Esta capacidade singular de fracionar o instrumento em grupos menores concede à linguagem a noção e, por conseguinte, o mecanismo do subinstrumento, noção esta útil em composição musical que explora a elaboração sistemática ou algorítmica de timbres. A ação do subinstrumento é eficaz especialmente na obtenção de timbres correlatos, mas também possibilita o emprego de uma certa "economia" de componentes espectrais no espaço tímbrico global de uma eventual peça.

### SINTAXE

SOM-A é um interpretador de programas espectrais que possuem pelos menos duas partes: uma orquestra feita de unidades-H, que é o primeiro bloco sintático, e uma seqüência de notas. Estes programas são denominados cartas espectrais, e constam de arquivos de caracteres representando uma seqüência de instruções capazes de criar instrumentos, atribuir valores de lapso e de taxa de amostragem, bem como executar nos instrumentos um aglomerado de notas, sejam elas simultâneas ou não, segundo uma ordenação cronológica e uma normalização de amplitudes realizadas previamente.

#### 1. Operador VAL

É sempre o primeiro comando de uma carta, como um cabeçalho, devendo ser aplicado sob a forma:

(VAL  $t_1 t_2 F_a A T N L$ )

Sua função é atribuir valores ao lapso de tempo  $t_1 t_2$  em segundos, à taxa de amostragem  $F_a$  em cps, aos 3 modificadores  $A T N$  de interpretação de notas, e ao número de pontos máximo  $L$  para as envoltórias. Estes parâmetros são assim definidos:

- A *Andamento*: valor que multiplica (com exceção dos tempos  $t_1 t_2$  estabelecidos por VAL) todos os tempos e durações, quais sejam, (1) a vigência dos instrumentos; (2) os parâmetros  $t_a t_b$  do operador EXE; (3) o instante inicial e (4) a duração ( $d$ ) da nota.
- T *Transposição*: valor que multiplica as frequências ( $f$ ) das notas, sem, contudo alterar as ordens de frequência dos instrumentos.
- N *Norma*: valor que multiplica as amplitudes ( $a$ ) das notas.
- L *Limite de envoltória*: valor limite para as abscissas das envoltórias.

Exemplo: a instrução (VAL 0 10 12000) solicita que a carta seja interpretada de 0–10 s a uma taxa de amostragem

de 12000 cps. Como os modificadores não estão explicitados, todos eles assumem o valor 1, enquanto que o limite de envoltória valerá 511 por default.

## 2. Operador INS

Define um instrumento na íntegra, através de uma expressão da forma:

(INS <vigência> <nome-ins> <unidades-H>)

para o qual:

<vigência> é o instante a partir do qual o instrumento <nome> deixa de existir.

<nome-ins> é uma cadeia de caracteres possuindo obrigatoriamente um literal como primeiro caractere.

<unidades-H> é uma lista de sublistas (( $h_1 \phi_1 e_1 \Lambda_1$ ) ( $h_2 \phi_2 e_2 \Lambda_2$ ) ... ( $h_k \phi_k e_k \Lambda_k$ )) que determinam os valores das partes funcionais de cada unidade-H do instrumento, quais sejam:

$h_i$  ordem de frequência da i-ésima unidade: número real positivo.

$\phi_i$  ângulo de fase: valor expresso em graus.

$e_i$  envoltória: Lista de pares de coordenadas (ordenada  $\times$  abscissa) sendo a ordenada um valor real compatível com os limites numéricos da implementação, e a abscissa um número inteiro compreendido entre zero e  $L$ . Estes pares possibilitam a especificação de uma forma geométrica através de até  $L + 1$  amostras de uma curva que modulará em amplitude a respectiva componente espectral e que se ajustará à duração da nota.

$\Lambda_i$  grau de estereofonia. Valor entre 0 e 1 que corresponde ao balanço de saída da componente com relação aos canais estereofônicos  $x$ ,  $y$ . Se for 0, a saída estará totalmente em  $x$ .

## 3. Operador EXE

A aplicação deste operador na forma abaixo submete uma seqüência de eventos espectrais <<notas>> entre dois instantes aos respectivos instrumentos da orquestra:

(EXE  $t_a t_b$ )

<<notas>>

(STP)

Em que <<notas>> é uma seqüência de notas <nota<sub>1</sub>> <nota<sub>2</sub>> ... <nota<sub>n</sub>>. Diversas seções (EXE  $t_a t_b$ ) <<notas>> (STP) podem compor uma carta, uma vez que o interpretador SOM-A entende o operador EXE da seguinte maneira: "execute todas as notas da seqüência abaixo até que o operador (STP) ocorra, ou o instante de execução  $t$  seja maior ou igual a  $t_b$ ". Na situação mais comum,  $t_a$  é zero e  $t_b$  é a duração da carta. Ainda que se possa utilizar um mesmo instrumento para diferentes vozes superpostas ou não, uma das imposições do interpretador EXE é que as notas estejam ordenadas segundo o seu segundo parâmetro, isto é, devem aparecer na carta em ordem cronológica.

<nota> é uma ordem de execução de um certo instrumento previamente definido na orquestra, de modo que ela será tocada a partir de um certo instante, por um certo tempo, a uma dada frequência, a uma dada intensidade, sendo escrita como uma lista de 5 elementos:

<nota> = (<instrumento>  
<instante inicial>  
<duração>  
<frequência em cps>  
<amplitude>)

O instrumento solicitado pela nota pode ser escrito como um átomo ou como uma lista, isto é:  
<instrumento> = <nome-ins> | <lista definidora de um sub-instrumento>

No caso de ser um átomo, a nota estará exigindo a ação plena do instrumento. De outro modo, isto é, se o instrumento for representado por uma lista, a nota estará clamando por um subconjunto das unidades-H de um determinado

instrumento, o qual passará a ser considerado como um macro-instrumento. A seleção de unidades-H originária desse macro-instrumento é o que se denomina subinstrumento, que é assim definido:

<subinstrumento> = (<nome do macro-instrumento>  
<(n<sub>1</sub> n<sub>2</sub> ... n<sub>j</sub>)>  
<vigência do subinstrumento>)

sendo o primeiro elemento é um átomo, representando um macro-instrumento, o segundo é uma lista de números representando unidades-H selecionadas desse macro-instrumento. Cada número representando a posição da unidade-H na seqüência <unidades-H>. O terceiro é opcional. Trata-se da vigência do subinstrumento criado com a nota. Quando um subinstrumento é criado para atender apenas a uma nota, não há necessidade de se especificar a vigência; SOM-A se encarregará de calcular este valor.

## 4. Operador STP

V. operador EXE.

## 5. Operador FIM

A aplicação do operador (FIM) encerra a interpretação da carta.

### DUAS CARTAS

Na primeira carta abaixo, a primeira nota solicita um subinstrumento derivado de I1, de tal maneira que apenas a sua primeira componente será ativada (10x10=100 Hz), soando inteiramente no canal  $x$ . Embora esta primeira nota dure apenas 0.2 s, a vigência do subinstrumento foi fixada em 0.8 em razão da quarta nota vir a utilizar este mesmo subinstrumento entre os instantes 0.6 e 0.8 s. A segunda nota solicita a segunda componente totalmente no canal  $y$ . A terceira faz soar 80% da terceira componente no canal  $x$  e 20% no canal  $y$ . A quarta é semelhante à primeira. A quinta nota solicita um subinstrumento com duas unidades-H de I1, justamente a primeira e a segunda; e a penúltima nota solicita todas as unidades-H de I1, o que corresponde a se utilizar I1 por um todo, sendo o resultado equivalente à ação da sétima e última nota.

```
(VAL 0 1.6 11025 1 1 1 925)
(INS 1.6 I1
 (10 0 ((0 0) (30000 385) (4000 343) (0 925)) 0)
 (12 90 ((0 0) (7500 385) (0 925)) 1)
 (15 180 ((0 0) (3500 385) (0 925)) 0.2)
 (EXE 0 1.6)
 ((11 (1) 0.8) 0.0 0.2 10 1)
 ((11 (2) 0.4) 0.2 0.2 10 1)
 ((11 (3)) 0.4 0.2 10 1)
 ((11 (1)) 0.6 0.2 10 1)
 ((11 (1 2)) 0.8 0.2 10 0.6)
 ((11 (1 2 3)) 1.0 0.2 10 0.4)
 (I1 1.4 0.2 10 0.25)
 (STP)
 (FIM)
```

A carta abaixo é um extrato de 1s da composição algorítmica T3OLD.CAR, ilustrando (1) a ortoestereofonia estrita, (2) a ocorrência de valores negativos nas envoltórias, e (3) valores não inteiros — sendo alguns menores do que a unidade — para as ordens de frequência.

```
(VAL 0 1.0 44100 0.0222222 1 1.25 720)
(INS 24 A21
 (1 40.08 ((0 0) (-92 7) (-5134 14) (908 21) (0 720)) 0)
 (0.7033 40.55 ((0 0) (2166 9) (-2875 18) (0 720)) 0)
 (0.5717 39.91 ((0 0) (1845 10) (0 720)) 0)
 (0.9335 41.10 ((0 0) (-3696 8) (0 720)) 0)
 )
```

```

(INS 24 A22
(1 43.06 ((0 0) (-1771 12) (-5355 24) (-3304 36) (0 720)) 1)
(1.3289 43.79 ((0 0) (833 11) (-2751 23) (0 720)) 1)
(0.0605 136.43 ((0 0) (4322 14) (0 720)) 1)
(0.245 43.57 ((0 0) (-3707 13) (0 720)) 1)
)
(INS 6 A23
(1 44.79 ((0 0) (5224 9) (0 720)) 0)
(1.4084 44.27 ((0 0) (6886 7) (0 720)) 0)
(1.5638 45.01 ((0 0) (4274 6) (0 720)) 0)
)
(INS 6 A24
(1 45.36 ((0 0) (2950 12) (0 720)) 1)
(0.4581 44.55 ((0 0) (-10723 13) (0 720)) 1)
(2.4536 44.80 ((0 0) (2710 10) (0 720)) 1)
)
(INS 41 A25
(1 29.92 ((0 0) (-778 46) (-428 91) (20 137) (0 720)) 0)
(1.532 40.61 ((0 0) (-575 10) (-225 20) (223 31) (0 720)) 0)
(1.9605 55.89 ((0 0) (-738 1) (-388 2) (60 3) (0 720)) 0)
)
(INS 41 A26
(1 135.47 ((0 0) (1317 93) (740 186) (264 279) (0 720)) 1)
(0.2397 139.32 ((0 0) (1202 111) (625 223) (150 334) (0 720)) 1)
(0.7683 40.93 ((0 0) (972 93) (395 185) (-80 278) (0 720)) 1)
)
(EXE 0 41)
((A24 (2 3)) 0      6      2166.678    0.8322)
((A23 (2 3)) 0      6      7127.652    0.8322)
((A22 (4)) 6       6      2826.039    0.8845)
((A21 (4)) 6       6      8340.816    0.8845)
((A22 (3)) 12      6      2826.039    0.8845)
((A21 (3)) 12      6      8340.816    0.8845)
((A22 (2)) 18      6      2826.039    0.8845)
((A21 (2)) 18      6      8340.816    0.8845)
(A26      23      18      3982.623    0.8666)
(A25      23      18      7822.052    0.8666)
(STP)
(FIM)

```

#### REFERÊNCIAS

- ARCELA, A. 1986. "Time-trees: the inner organization of intervals." *Proceedings of the International Computer Music Conference*, Haia.
- CASTRO, R.R.F. 1994. Implementação de SOM-A em Borland C++ para o ambiente MS-Windows, *Relatório Técnico LPE-9402*, Universidade de Brasília.
- MATHEWS, M.V. et. al. 1969. *The technology of computer music*. Cambridge, Mass.: The MIT Press.
- MEIRELES, A., GIOIA, O.G. e CASTRO, R.R.F. 1993. SOM-A em C para SUN SparcStation, *Relatório Técnico LPE-9303*, Universidade de Brasília.
- MOORER, J.A. 1977. "Signal processing aspects of computer music—A survey." *Proceedings of the IEEE* 65(8): 1108-1137.
- NOGUEIRA FILHO, V. 1988. Síntese aditiva modular — uma máquina espectral programável. *Dissertação de mestrado*, Universidade de Brasília.
- RAMALHO, G.L. 1991. SOM-A em Sun Common LISP para o ambiente Open Windows. *Relatório Técnico LPE-9105*, Universidade de Brasília.
- MULISP 1987. *muLISP-87, LISP Language Programming Environment*. SoftWarehouse Inc., Honolulu, Hawaii.

## Processador de Efeitos em Sinais Digitais de Áudio

MÁRCIO DA COSTA PEREIRA BRANDÃO  
 CARLOS AUGUSTO JORGE LOUREIRO  
 TÚLIO DA COSTA ZANNON  
*Laboratório de Processamento Espectral*  
*Departamento de Ciência da Computação*  
*Universidade de Brasília - Brasília, DF*  
 CEP 70910-900 BRASIL

#### RESUMO

Um sistema para o processamento de sinais digitais de áudio por meio de estruturas geradas através da interligação de blocos básicos é aqui descrito. Podem ser utilizados diversos tipos de blocos básicos tais como somadores, multiplicadores, osciladores e blocos de retardo. As estruturas estão descritas por arquivos de configuração que contém a forma de interligação, juntamente com os parâmetros de funcionamento destes blocos. O objetivo principal deste trabalho é a construção de estruturas que correspondam a sistemas de ambientação de áudio, tais como filtros, reverberadores e câmaras de eco. Como é possível arbitrar-se as interligações entre os blocos, as configurações disponíveis não são limitadas como no caso da maioria dos processadores de efeitos implementados em 'hardware'. Além disto, o sistema foi idealizado de tal forma que seja fácil a criação de novos blocos através da simples incorporação de novas funções.

#### 1. Introdução

O objetivo deste trabalho é descrever o desenvolvimento de um processador de efeitos para os sinais digitais de áudio produzidos no Laboratório de Processamento Espectral (LPE) da Universidade de Brasília. Estes sinais são gerados pelo programa de síntese ortoestereofônica SOM-A (Arcela, 1989), em conjunto com ferramentas de composição algorítmica que se fundamentam na Teoria da Árvore de Tempos (Arcela, 1984, 1986, 1991).

Através de configurações arbitrárias de interligações entre suas unidades básicas, sinais de áudio armazenados em arquivos são transformados pelo sistema, possibilitando a obtenção dos mais variados tipos de efeitos nos arquivos de áudio de saída. Como o sistema foi idealizado visando o processamento por software, sem o apoio de DSP's, não é possível a sua operação em tempo-real (Lobão, Martinelli, 1992). Esta limitação em parte é compensada pela possibilidade de sua utilização em diferentes plataformas que não disponham de recursos adicionais de hardware.

#### 2. Estrutura Interna

A estrutura interna do processador de efeitos é composta por buffers de dados, buffers de operações e blocos básicos, como mostra a Figura 1. O sistema foi implementado em torno de uma estrutura totalmente baseada no uso de buffers para que seja possível a utilização de configurações arbitrárias para as interligações entre os blocos, além de possibilitar expansões futuras com facilidade.

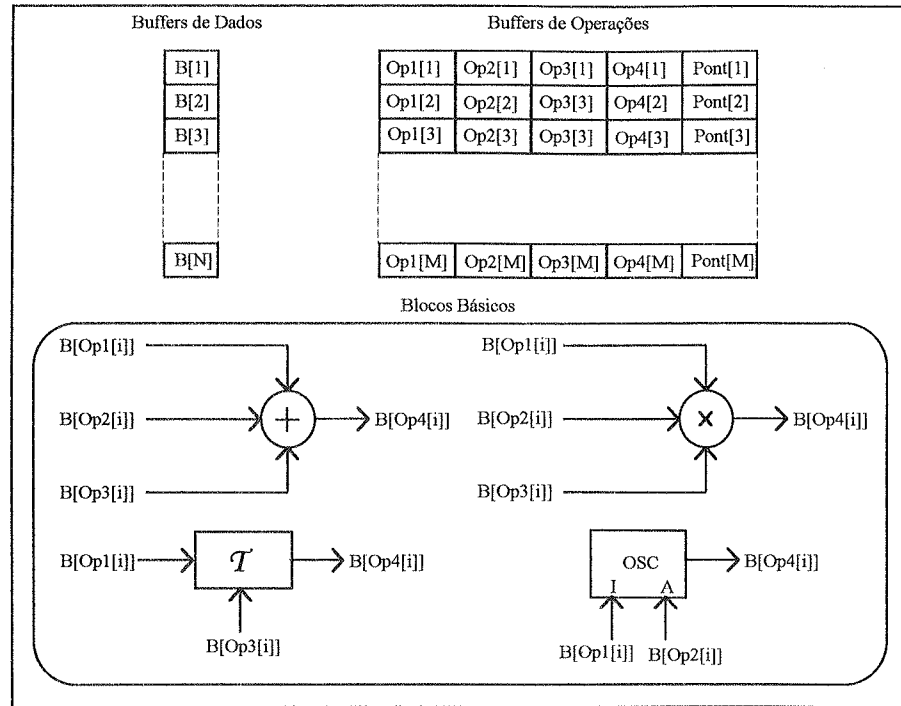


Figura 1 - Estrutura Interna do Processador de Efeitos

Na Figura 1 podemos observar que os buffers de operações armazenam dados pertinentes aos blocos utilizados na estrutura descrita pelo arquivo de configuração, sendo que os valores de entrada e saída de cada bloco correspondem a buffers de dados.

**Buffers de Dados**

Os buffers de dados são utilizados para armazenar dados de entrada, valores intermediários de operações e dados de saída, e foram agrupados em um vetor de elementos reais. Todas as operações do sistema recebem como parâmetros índices para este vetor.

**Buffers de Operações**

Os buffers de operações contém parâmetros para as funções, o índice do buffer de dados que receberá o resultado da operação, além de um ponteiro para a função a ser executada. Os buffers foram agrupados em um vetor onde cada elemento apresenta quatro campos do tipo inteiro (Op<sub>1</sub>, Op<sub>2</sub>, Op<sub>3</sub> e Op<sub>4</sub>), e um campo do tipo ponteiro, que aponta para uma das funções que implementam os blocos básicos. Esta maneira de se efetuar as chamadas de funções permite que novas operações sejam facilmente implementadas.

O campo Op<sub>4</sub> indica sempre o buffer de dados que receberá o resultado da operação, e os campos Op<sub>1</sub>, Op<sub>2</sub> e Op<sub>3</sub> indicam os buffers de dados que contém amostras de entrada ou parâmetros específicos a cada função.

**Blocos Básicos**

Foram implementadas funções correspondentes aos blocos fundamentais mais utilizados em processamento digital de sinais. Atualmente estão implementados os seguintes blocos básicos: somador, multiplicador, oscilador e bloco de retardo. Em sua utilização é necessário especificar quais são os buffers de entrada e de saída, além de parâmetros específicos a cada bloco, conforme mostrado na figura 1. Desta forma a interligação entre os blocos pode ser determinada pelo conteúdo dos buffers de operação. O significado dos parâmetros para cada função depende do bloco, como é visto a seguir.

**Somador e Multiplicador**

Os três primeiros parâmetros são índices que apontam para os buffers de dados que contém os elementos a serem somados ou multiplicados, ou seja, os valores de entrada para o bloco. O quarto parâmetro é um índice que aponta para o buffer de dados que receberá o resultado da operação, conforme mostrado na figura 1. Caso apenas duas entradas estejam sendo utilizadas, é necessário que a terceira entrada corresponda a um buffer que apresente valor inicial igual a zero no caso do somador, ou valor inicial igual a um no caso do multiplicador.

**Bloco de Retardo**

O primeiro e o quarto parâmetros são índices que apontam para os buffers de dados utilizados respectivamente como entrada e saída. O segundo parâmetro indica o número do bloco de retardo no esquema e o terceiro indica o retardo que será utilizado. Como este último valor corresponde a um buffer de dados que pode estar sendo utilizado como saída de outro bloco, é possível que o seu valor se altere durante o processamento do sinal.

**Oscilador**

O primeiro e o segundo parâmetros são índices que apontam, respectivamente, para os buffers que contém o incremento I a ser utilizado na busca em tabela, e o fator A de amplitude. O terceiro parâmetro indica o número do bloco de oscilação no esquema. Originalmente este bloco é capaz de produzir apenas a forma de onda senoidal. (Moorer, 1988).

**3. O Arquivo de Configuração**

É um arquivo de texto convencional, contendo linhas terminadas pelo par <Carriage Return> + <Line Feed>, de tal forma que pode ser gerado por um editor de texto convencional. Neste arquivo estão descritas as interligações, e os valores dos parâmetros específicos dos blocos a serem utilizados de acordo com o seguinte formato:

N						; Número de buffers utilizados
B <sub>ea</sub>	B <sub>eb</sub>					; Índices do Buffers Estéreo de Entrada
B <sub>sa</sub>	B <sub>sb</sub>					; Índices do Buffers Estéreo de Saída
Buf <sub>1</sub>						; Valor inicial para o Buffer 1
Buf <sub>2</sub>						; Valor inicial para o Buffer 2
...						
Buf <sub>N</sub>						; Valor inicial para o Buffer N
M						; Numero de blocos utilizados
Op <sub>11</sub>	Op <sub>12</sub>	Op <sub>13</sub>	Op <sub>14</sub>	F <sub>1</sub>		; Operandos e função a ser chamada para o Bloco 1
Op <sub>21</sub>	Op <sub>22</sub>	Op <sub>23</sub>	Op <sub>24</sub>	F <sub>2</sub>		; Operandos e função a ser chamada para o Bloco 2
...						
Op <sub>M1</sub>	Op <sub>M2</sub>	Op <sub>M3</sub>	Op <sub>M4</sub>	F <sub>M</sub>		; Operandos e função a ser chamada para o Bloco M

Figura 2 - O Formato do Arquivo de Configuração

As funções que serão chamadas em cada caso dependem de  $F_i$ , que atualmente pode apresentar os seguintes valores: soma = 1; multiplicação = 2; retardo = 3; oscilador = 4. Todos os valores são inteiros, exceto pelos valores  $Op_{ij}$  que são reais.

O exemplo abaixo mostra o conteúdo de um arquivo de configuração que implementa um sistema de reverberação simples (Oppenheim, 1978).

```

13                ; Número de buffers
1      7          ; Índice dos Buffers de Entrada
3      9          ; Índice dos Buffers de Saída
0.0      ; Valor inicial de B[1] = Buffer de entrada do canal A
0.0      ; Valor inicial de B[2] = Saída do somador do canal A
0.0      ; Valor inicial de B[3] = Buffer de saída do canal A
0.0      ; Valor inicial de B[4] = Saída do multiplicador do canal A
0.25     ; Valor inicial de B[5] = Retardo do Canal A
0.7      ; Valor inicial de B[6] = g
0.0      ; Valor inicial de B[7] = Buffer de entrada do canal B
0.0      ; Valor inicial de B[8] = Saída do somador do canal B
0.0      ; Valor inicial de B[9] = Buffer de saída do canal B
0.0      ; Valor inicial de B[10] = Saída do multiplicador do canal B
0.25     ; Valor inicial de B[11] = Retardo do canal B
0.0      ; Valor inicial de B[12] = Valor nulo para soma
1.0      ; Valor inicial de B[13] = Valor unitário para multiplicação
6        ; Número de blocos utilizados
3      6      13      4      2      ; B[4] = B[3] * B[6] * B[13]
1      4      12      2      1      ; B[2] = B[1] + B[4] + B[12]
2      1      5      3      3      ; B[3] = B[2] com retardo B[5]
6      9      13      10      2      ; B[10] = B[6] * B[9] * B[13]
7      10     12      8      1      ; B[8] = B[7] + B[10] + B[12]
8      2      11     9      3      ; B[9] = B[8] com retardo B[11]

```

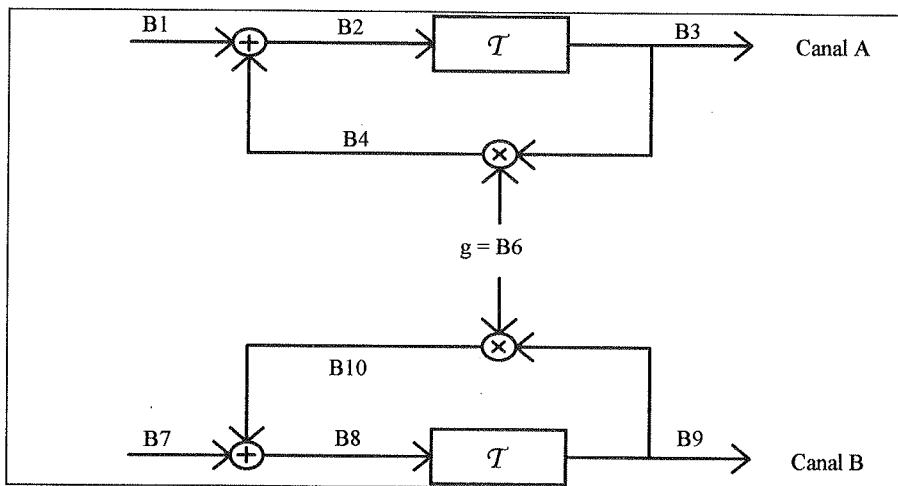


Figura 3 - Um sistema de reverberação simples

#### 4. Considerações Finais

O Processador de Efeitos aqui descrito foi inicialmente desenvolvido em linguagem C para o sistema operacional DOS durante a disciplina "Introdução à Computação Sônica" (Lobão, Martinelli, 1992) do bacharelado em Ciência da Computação da Universidade de Brasília. Atualmente o sistema está sendo migrado para a linguagem C++ para ser utilizado no ambiente Windows.

Diversas melhorias estão sendo introduzidas, dentre as quais podemos citar:

- Suporte ao formato WAV de arquivos de áudio do ambiente Windows
- Interface gráfica para a construção de estruturas de uma forma amigável, por meio de técnicas de "drag-and-drop".
- Módulo de visualização dos sinais de entrada e de saída à medida em que estão sendo processados.
- Macros para a descrição facilitada de configurações que se repetam em uma estrutura

#### 5. Referências

- Arcela, A. & Ramalho, G. (1991). A formal composition system based on the theory of Time-trees. *Proceedings of the ICMC, Montreal*.
- Arcela, A. (1984). As árvores de Tempos e a configuração Genética dos Intervalos Musicais. *Tese de Doutorado, PUC, Rio de Janeiro*.
- Arcela, A. (1989). A Linguagem SOM-A para Síntese Aditiva. *Anais do 1º Simpósio Brasileiro de Computação e Música, Caxambú, MG*.
- Arcela, A., (1986). Time-Trees: the inner organization of intervals. *Proceedings of the 12th International Computer Music Conference (ICMC), pp 87-89, Haia*.
- Lobao, A. S. & Martinelli, E. O. (1992). Processador de Sinais Estéreo. *Monografia da disciplina "Introdução à Computação Sônica". Departamento de Ciência da Computação, UnB, Brasília*.
- Moorer, V. F. (1988). Table Lookup Noise for Sinusoidal Digital Oscillators, *Computer Music Journal*, Vol. 1 N. 2, pp. 26-39
- Oppenheim, A. V. (ed), Blesser, B. & Kates, J. M. (1978). Digital Processing in Audio Signals. *Applications of Digital Signal Processing*. Englewood Cliffs, NJ, Prentice-Hall Inc.

## FracWave: Non-linear Dynamics as Timbral Constructs

JÔNATAS MANZOLLI

*Interdisciplinary Nucleous for Sound Studies (NICS)*

*University of Campinas (UNICAMP)*

*13089-730, Campinas - SP, Brazil*

*Jonatas@dsif.fee.unicamp.br*

### Abstract

FracWave produces sounds with dynamic characteristics by means of a parametric control of simple non-linear maps. It is a compositional tool which allows a composer to generate new sounds and to build up sonic structures from an atomic level. This paper discusses the basic concepts about FracWave, it elucidates how the model was derived from recent research on non-linear dynamics, it presents a compositional approach based on parametric scores, and it illustrates the musical results presenting graphics and sound examples taken from compositions created by the author using FracWave.

### Introduction

The use of Non-linear Dynamics (NLD) in music is in line with the recent development of scientific models led by the Theory of Chaos (Gleick, 1987). Methods derived from NLD have been applied to Acoustics - a basic reference is found in (Lauterborn & Parlitz, 1988), and a method of Analysis of Musical Signal is presented in (Bernardi, Bugna & De Poli 1992). In Algorithmic Composition, there has been research on applications of Iterative Maps to describe compositional systems (Bidlack, 1992). In parallel, the use of NLD have revitalised Timbral Design (Truax, 1990; Scipio, 1990), in these methods non-linear maps are used to organize musical structures from an atomic level.

FracWave is in line with the third perspective above. It aims to develop micro-structural constructs to produce new types of sonic behaviour and to create sounds with complex and dynamic characteristics. Numerical material generated by non-linear maps shapes waveforms i.e. the method is an application of mathematical iterative processes. Therefore FracWave is a synthetic and heuristic approach. Using Smith's definition (Smith, 1992) it could be described as an Abstract Algorithm for Sound Synthesis. It produces sounds with rich spectra and it does not intend to simulate either an acoustic instrument or a classical acoustic model.

This paper is in line with previous publications in which the author introduced FracWave as a sound generator, discussed its possible musical applications and presented an extensive documentation of the method (Manzolini, 1992,1993). The report here concentrates on compositional issues. Nevertheless it recapitulates FracWave basic ideas, it summarizes the relations between NLD and FracWave, and it presents a mathematical formulation of the algorithm. After that, sonic and compositional issues are discussed and followed by graphics and sound examples.



**FracWave Basic Ideas**

The basic concept behind FracWave is: the model was developed in contradistinction to digital techniques which use wavetables as invariant sound synthesis units. FracWave produces sounds using numerical buffers controlled by simple non-linear dynamical systems. These buffers are coined *Dynamic Wavetables* and they replace the traditional digital oscillators. The idea is to use algorithmic manipulation to generate complex types of sonic behaviour. There are models similar to FracWave concerned with this manipulation of data which then produces *abstract waveforms*. Berg (1979) developed a sound synthesis language called PILE. His method was described by him as *one single sound, the perception of which is represented as a function of amplitude distribution in time*. Another approach was presented recently by Serra (1992), *a new achievement in Xenaki's stochastic work*. This method is based on stochastic control of waveforms defined by polygonized lines.

The FracWave algorithm is divided into two kinds of processors: a) *non-linear processors which are a set of eight simple non-linear maps and b) linear processors which are the Dynamic Wavetables*. The basic principle is to work with a sound generator unit which produces a time-varying waveform based on two iterations in parallel: the non-linear map's iteration and the Dynamic Wavetable iteration. Finally, the basic elements of the method presented here are: a) *non-linear maps - source of waveforms, b) Dynamic Wavetables - micro-structural constructs and c) Structural Links (Sound Cells and Sound Streams) - tools to link the micro with the macro level of composition*.

**Phase Space Analysis**

Feedback applied to simple mathematical models could generate chaos. In Chaos methodology it is visualised in a space called phase-space or state-space in which the coordinates are the degrees of freedom of the system. Each point in the phase-space represents the entire state of a dynamical system in a certain moment of time. A non-linear map generates single points, limit cycles, simple or chaotic oscillators in the phase-space. These graphics are named attractors or strange attractors (i.e. in a chaotic case) which are asymptotic limits of the system's solution as time approaches infinity.

Based on these concepts the author developed a software tool to analyse the phase-space graphics. The FracWave's thesis were derived from this analysis as follows: a) a non-linear map produces periodic, quasi-periodic or chaotic motion by changing its initial conditions (X0, Y0) or its parameters (A, B, C) (see Table 1), b) this behaviour in the phase-space is visualised by an attractor (see Figure 2) and c) the numerical behaviour of the map in the phase-space mirrors the sonic behaviour of a waveform.

**Non-linear Maps**

The author created a set of eight simple non-linear maps to use in research. This text does not include the complete set, see (Manzoli, 1993) for more information. Three non-linear maps used in FracWave are presented below:

$$\begin{cases} X_{k+1} = Y_k - \text{sign}(B - Y_k) \sin(C\pi k) \\ Y_{k+1} = A - X_k \end{cases}$$

$$\begin{cases} X_{k+1} = Y_k - \text{sign}(X_k) \sqrt{|BX_k - C|} \\ Y_{k+1} = A - X_k \end{cases}$$

$$\begin{cases} X_{k+1} = Y_k - \text{sign}(X_k) + \sqrt{|BX_k - C|} \\ Y_{k+1} = A - X_k \end{cases}$$

The sequence (X<sub>k</sub>) produced by the maps is mapped into the interval [-1,-1] to be used as waveform. Additionally, linear interpolation and oversampling are applied to smooth the resultant curve.

**Implementation and Mathematical Formulation**

The computational implementation of the FracWave's sonic construct i.e. Dynamic Wavetable (DW), is based on a numerical buffer which is read using one increment (I<sub>n</sub>) while it is simultaneously refilled with new values from a non-linear map using another increment (J<sub>n</sub>). A DW is a delay line (Figure 1 illustrates FracWave implementation) and it is also used as an Average Filter. This procedure could be related to the LAsy Technique (Chareyron, 1990) and Physical Models (Smith, 1992). The average process used in FracWave slows down the energy of the generated sound working as a dumping factor. It is a digital filter which concentrates the spectrum's energy on an average partial. The implementation is described by two equation as follows:

$$Y_n = W[I_n]$$

$$W[J_n] = \alpha X_n + \frac{1-\alpha}{p+1} \sum_{m=0}^p A_m Y_{n-m}$$

where Y<sub>n</sub> is the output sequence, W[.] is a DW, X<sub>n</sub> is the non-linear map sequence, 0 ≤ α ≤ 1 is a combination factor and (A<sub>m</sub>)<sub>p=0...m</sub> is the coefficients of the average filter.

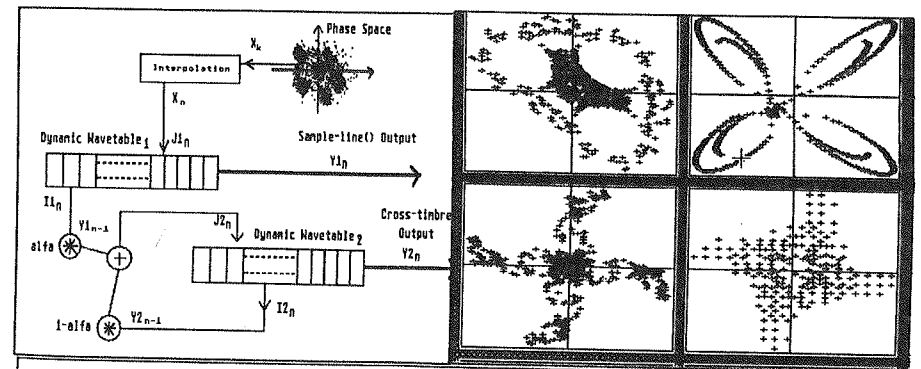
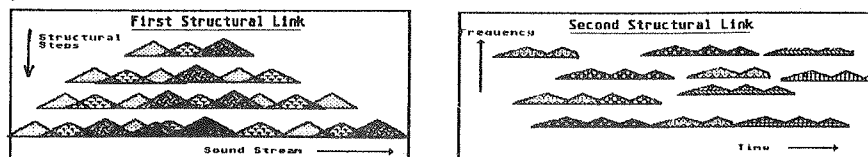


Figure 1, on the left, is a diagram of the FracWave Algorithm. Figure 2, on the right, is a sequence of four attractors in the phase-space.

## Structural Links

The structural manipulation described here was inspired by Granular Synthesis (Roads 1985). The author developed two tools to build sonic structures: a) *Sound Cell* - the parameters which controls FracWave (see Table 1) and b) *Sound Stream* - parametric score produced by Sound Cells. Starting from the micro level, two structural link build up macro structures: a) *First structural link* - Sound Cells which control sound segments (50-100 ms), form Sound Stream and b) *Second Structural Link* - Sound Segments (500-1000 ms) generated by Sound Stream are transposed and superimposed to build macro sound events.

A sequence of Structural Steps constructs Sound Stream in the First Structural. Starting from an initial set of Sound Cell, operations such as permutation and sequencing are used to produce a Sound Stream. This new Sound Stream is used as a unitary Sound Cell in the next Structural Step. This iterative process produces a growth through levels of scale which forms the Sound Streams. These parametric scores generate a *Sound Palette* which is used in the next structural link. In the Second Structural Link concatenation of the sounds of Sound Palette is produced by means of permutation in the horizontal axis (time axis), and superimposition and frequency transposition in the vertical axis (frequency axis). In parallel, a Triangular Window is used to splice these sounds for it smooths gaps between two waveforms avoiding glitches in the resultant sound.



## Mutations of Sound Cells

To create a Sound Stream a composer needs to input a great deal of data from the computer keyboard. Thus a computational tool was developed in research to generate a Sound Stream as a sequence of parametric mutations of an original Sound Cell. A composer has therefore to input less data as follows: a) an original Sound Cell, b) a set of three parameters to control the percentage of change in the Sound Cell and c) a set of three parameters to determinate the parameters involved in change. After that, the computer generates a Sound Stream using these three parameter sets to control an iterative random process. Notice that, the Mutation Operations is used as First Structural Link. Finally, the Iterative Random Equation is presented as follows:

$$P_{k+1} = P_k + ((\text{Rnd}() - 0,5) * C) / 100$$

where  $P_k$  is a Sound Cell parameter,  $(C)_{i=1...3}$  are the percentual of changing, and  $K$  is determined by a random choice in  $(K)_{i=1...3}$ .

duration	non-linear map					dynamic wavetable			average filter
	A	B	C	X <sub>0</sub>	Y <sub>0</sub>	F <sub>osc</sub>	F <sub>ref</sub>	α	
milliseconds									$\{A_m\}_{m=0...P}$

Table 1 presents the parameters of a Sound Cell. Notice that  $F_{ref}$  and  $F_{osc}$  are related to  $J_n$  and  $I_n$  by the the same equation  $J_n = \text{ROUND}(n * F_{ref} / F_s) \text{ MOD } L$  with  $F_s$  = sampling frequency and  $L$  is the number of points in the DW.

## Composing Soundscapes

The sonic aim of FracWave was inspired by the complexity of sounds found in nature. The research focused on generating sounds which evoke forces of nature recalling phenomena such as turbulences, wave-breaks, explosions etc; a timbral palette distincts from more typical electronic ones. The spectral typology of the sounds produced by FracWave can be related to the description of Smalley (1986) in *Spectro-morphology and Structuring Processes*. Most of these sounds are allocated to the *pitch-effluvium continuum*, as defined by Smalley, between *nodal spectrum* and *noise*. These sounds have complex internal morphologies. Therefore the composer has to confront the compositional challenge: how to organize them in a convincing musical way? Wishart (1985) presents a classification of sounds with complex morphology which is useful here. In his words: a *number of archetypes which allow us to classify these complex sounds perceptually, such as Turbulence, Wave-break, Open/Close, Siren/Wind, Creak/Crack, Unstable/Settling, Shatter, Explosion, Bubble*.

From one point of view the sounds produced by FracWave have complex internal morphologies, from the other these morphologies are very characteristic (such as Turbulence, Wave-break). Thus it is possible to operate with these peculiar features to produce sound-images. The compositional tool for organizing complex sounds could be to manipulate them in a symbolic context, and to associate the musical meaning with the sonic behaviour of these complex constructs.

Let us to exemplify the compositional approach presented here with ideas from Berio's vocal work *A-Ronne* (1974). This piece was based on a poem by Sanguineti in which he arranges quotations from different languages in segments and agglutinates them in hybrid sentences. There is an example of the poem *A-Ronne* as follows:

*a:ah:ha:hamm:anfäng*  
*in:in principio: nel mio*  
*principio:*  
*am anfäng: in my beginning*

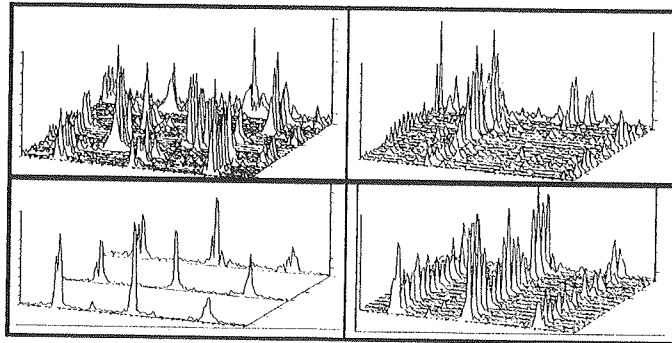
The structural manipulation used by Sanguineti was to re-create expressions from different languages copying words and linking them according to similarities of meaning. The aesthetic view of *A-Ronne* was described by Berio (1976) as *changes in expression imply and document changes in meaning*. In the same way the poem *A-Ronne* was constructed using words, a electroacoustic suite called *Turbulências* was composed using sounds. The compositional approach was an agglutinative technique in which sounds (i.e. equivalent to words) were combined to form sound-images (i.e. equivalent to sentences) - similarities between sounds were related to the similarities between sonic archetypes (see Wishart above).

For instance, the name of the first piece in *Turbulências* give us a cue to understand the aesthetic of the piece. *Agglomerados* is Portuguese for agglomerates, which are fragments of rock fused together in a mass. An agglomerate material is a structure which grows into a mass, for example a growing crystal. Within this metaphorical context, it is possible to say that sounds could be compacted to form a soundscape of musical crystals. This idea is related to two different sonic structures in *Agglomerados: Monoliths and Creatures*. The first one resembles massive and dense structures such as compacted stones, diamonds. The second one resembles granular sound structures, like graphite.

The micro structure of these two sonic entities is controlled by Sound Cells and Sound Streams as described above. They differ from each other by the way they are built at the First Structural Link. Monoliths are compact structures with an implied amalgamation of atomic sound/waveforms. This is created by using Sound Cells with durations between 50-100 milli-seconds and Sound Streams with the number of Sound Cells between 10-15. Creatures are rhythmic structures, an implied increase of distinction between the components micro sounds. This is produced by using Sound Cells with durations between 30-70 milli-seconds and Sound Streams with the number of Sound Cells between 5-10.

### Graphic Examples

The examples below illustrate Sound Transformations produced by FracWave. The first example (from left to right) describe a spectral shift which concentrates the spectrum's energy on an average partial produced by the Average Filter. The second example shows a spectral development derived from an initial Sound Cell and a subsequent Sound Stream.



In the graphics above the horizontal axis is frequency, the vertical axis is amplitude and the third axis describes the time evolution from the background to the foreground.

### Discussion and Conclusion

The application of non-linear maps as sonic constructs confronts the musician with a compositional challenge: to organize chaotic sounds in a meaningful and coherent musical structure. On one hand, this can be approached by agglutinative techniques and by the symbolic development of chaotic soundscapes. On the other hand, this complex sound material produces a sonic paradox - while it is the musical construct it may cause sonic opacity, which will serve rather to de-construct the music. The composer needs therefore to handle this material with studio techniques developed to avoid losses in structural clarity.

The next step on this investigation is to use FracWave to transform environmental sounds. These sounds could be rich sounds such as the material found in the Brazilian Soundscape. In this implementation, a DW filled by these samples could be combined with the waveforms generated by FracWave. Another possibility, could be to apply the average filter as a dumping factor creating spectral changes in the original sound. It is possible to project other transformations, but the main compositional issue is to integrate FracWave's synthetic sounds which resemble natural phenomena with similar sounds found in nature. This could produce new sonic textures which either FracWave or the natural sounds generates by itself.

### References

- Gleick, J. (1987). *Chaos: Making a New Science*. New York: Vintage.
- Lauterborn, W. & Parlitz, U. (1988). *Methods of chaos physics and their applications to acoustics*. Journal of the Acoustic Society of America 84(6):1975-1993.
- Bernardi, A., Bugna, G.P. & De Poli, G. (1992). *Analysis of Musical Signal with Chaos Theory*. Proceedings of the International Workshop on Models and Representation of Musical Signals, Capri.
- Bidlack, R. (1992). *Chaotic Systems as Simple (but complex) Compositional Algorithms*. Computer Music Journal 16(3):33-42.
- Truax, B. (1990). *Chaotic Non-linear Systems and Digital Synthesis: an Exploratory Study*. Proceedings of the ICMC, Glasgow.
- De Scipio, A. (1990). *Composition by Exploring of Non-linear Dynamic Systems*. Proceedings of the ICMC, Glasgow.
- Manzoli, J. (1992). *FracWave Sound Synthesis*. Proceedings of the International Workshop on Models and Representations of Musical Signals, Capri.
- Manzoli, J. (1993). *Musical Applications Derived from FracWave Sound Synthesis*. Proceedings of the 94th Audio Engineering Society Convention, Berlin.
- Manzoli, J. (1993). *Non-linear Dynamics and Fractals as a Model for Sound Synthesis and Real-time Composition*. PhD dissertation submitted to the University of Nottingham, England.
- Berg, P. (1979). *PILE - a language for sound synthesis*. Computer Music Journal 3(1):30-41.
- Serra, M. (1992). *Stochastic Dynamic Sound Synthesis: a new Achievement in Iannis Xenaki's Work*. Proceedings of the International Workshop on Models and Representations of Musical Signals, Capri.
- Chareyron, J. (1990). *Digital Synthesis of Self-modifying Waveforms by Means of Linear Automata*. Computer Music Journal 14(4):25-41.
- Smith, J. (1992). *Models of Music Signals arising from Results in Physics, Acoustics, and Signal Processing*. Proceedings of the International Workshop on Models and Representation of Musical Signals, Capri.
- Roads, C. (1985). *Granular Synthesis of Sound*. Foundations of Computer Music, ed. Roads, C. and J. Strawn. Cambridge, Massachusetts: The MIT Press.
- Smalley, D. (1986). *Spectro-morphology and Structuring Processes*. The Language of Electroacoustic Music, ed. Emmerson, S. London: The Macmillan Press Ltd, pg 61-93.
- Wishart, T. (1985). *On Sonic Art*. York: Imagineering Press, pg 100- 103.
- Berio, L. (1976). *Text A-Ronne*. DECCA ZAL 14741 (disc).

## An Overview of Criteria for Evaluating Synthesis and Processing Techniques

DAVID A. JAFFE  
295 Purdue Ave.  
Kensington, CA 94708  
david@jaffe.com

### Abstract

A wide variety of synthesis and processing techniques have been invented. The question may arise as to which is best. It turns out there is no simple answer. To help clarify the issues, we present a list of ten criteria for evaluating synthesis and processing techniques and give examples of well-known techniques that succeed especially well or fail especially poorly in each area.

*This paper is forthcoming (in expanded form) in Computer Music Journal.*

### Introduction

A *synthesis technique* is a strategy for generating sound samples, based on some control information called *parameters*. Parameters generally change at a rate that is significantly slower than the audio sampling rate. Also, while sound samples are usually produced at a constant rate, parameter-setting messages are often sporadic and irregular. For example, in the familiar Yamaha DX7 synthesizer (Chowning and Bristow, 1986), the synthesis technique is FM (frequency modulation) synthesis, which produces a steady stream of audio samples, while MIDI pitch bend is a parameter that changes only when the performer moves the modwheel or foot pedal. For a historical summary of synthesis techniques, see, for example, (Smith 1991.)

A *processing technique* is similar to a synthesis technique, except that it includes an additional input of one or more audio sample streams at the audio sampling rate. An example of a common processing technique is reverberation. Here, the parameters are controls such as the delay before the first reflections, the balance between the reverberant and dry signals and the decay rate of the echoes.

A wide variety of synthesis and processing techniques have been invented. The question may arise as to which technique is best. It turns out there is no simple answer--the best technique depends on the priorities of the user and the problem to be solved.

To help clarify the issues, we present a list of ten criteria for evaluating synthesis and processing techniques and give examples of well-known techniques that succeed especially well or fail especially poorly in each area. The first four criteria are concerned with the usability of the parameters: Are they intuitive? Does a change in a parameter have a perceptible effect? Do they map to physical attributes of musical instruments or other physical sound-producing mechanisms? Are they "well-behaved" or wildly non-linear? Other criteria deal with the sounds produced: Do they retain their identity in the context of variation? Can all classes of sounds be produced? Are there techniques for deriving parameters for real-world models? The remaining criteria are concerned with efficiency and implementation. How efficient is the technique? Does it have an undesirable unavoidable latency? How sparse is its parameter control stream?

The intention here is not to survey all known techniques, but to outline the criteria that make a given technique suitable or unsuitable to a specific purpose. A technique that is weak in a certain area can often be made stronger by combining it with another technique to produce a hybrid.

### How Intuitive are the Parameters?

This topic is concerned with whether the parameters map in an intuitive manner to musical attributes such as musical dynamics and articulation, or whether they are mere mathematical variables with very little correlation to real-world perceptual or musical experience.

In the old analog synthesizer technique of low-pass filtering a complex waveform, the bandwidth of the low-pass filter affects both the brightness and the amplitude of the sound. This control is very much like a musical "dynamics" parameter. In a real-world instrument, loud notes tend to have more significant higher

partials than soft notes, due to physical non-linearities becoming more prominent as the instrument is played louder. In the synthesis simulation, low values of the filter bandwidth correspond to a dynamic of pianissimo while high values correspond to fortissimo. This correlation is especially effective if the filter bandwidth is adjusted on a note-by-note basis so that a requested fundamental amplitude is obtained uniformly regardless of frequency, thus decoupling distance (largely a function of amplitude) from dynamics (primarily a function of brightness) (Jaffe and Smith 1983.)

In contrast, parameters of non-linear synthesis methods, such as complex FM (Schottstaedt 1977), can be quite non-intuitive. Although an extremely useful technique, it has the property that changing the FM index of a cascade modulator slightly can cause a drastic and difficult to predict change in tone quality, enough of a change to turn a drum into a woodblock. This situation may be merely annoying to a composer, who can take the time to find the proper value for his or her application, but it can drive a performer crazy who is trying to control such a parameter from an instrument in front of a live audience.

#### How Perceptible are Changes in Parameters?

Changing a parameter by a significant amount should have an obvious audible effect. We call such parameters *strong* or *powerful*, in contrast to *weak* parameters whose effect is barely audible. Typically, the more parameters a technique has, the weaker each parameter is. Parameters that are too weak leave the composer to wander in the dark, setting parameters to arbitrary values, since it's not possible to hear any difference. On the other hand, parameters that are too strong can also cause problems. If a tiny change causes a huge effect, a performer may have difficulty controlling the technique.

Again, the filter bandwidth of a low-pass filtered complex waveform is a good example of a reasonably strong parameter. A single parameter controls a clear effect of brightness. In contrast, additive synthesis, in which a tone is produced by a weighted sum of sinusoidal components, is an example of a technique with weak parameters. If you change the amplitude envelope of a less-important harmonic, the change can be completely inaudible.

A technique with weak parameters can be transformed into a related technique with more powerful parameters, often using the same underlying algorithm. For example, an additive synthesis brightness parameter can be defined that behaves similar to the low-pass filter brightness parameter. Such a parameter can be defined as a *meta-parameter* that simply scales a set of the existing additive synthesis parameters. We would define the user-supplied partial amplitudes and frequencies as the arrays *Amps[i]* and *Freqs[i]*, respectively, where 'i' is the partial index. In normal additive synthesis, the *Amps* values are used directly. With the new *brightness* parameter, the actual value applied could be defined as:

$$\text{ActualAmps}[i] = \text{Amps}[i] * \text{MAX}(\text{brightness}, 0.001) \wedge \log_2(\text{MAX}(\text{Freqs}[i], 100.0)/50.0)$$

Here, a *brightness* of 1.0 represents no modification and a *brightness* of .001 represents maximum low-pass filtering. ( $\log_2()$  is the *log* function, base 2 and *MAX()* returns the maximum of its arguments.) This technique can be viewed as a modified form of FFT filtering, where the additive synthesis is used in a manner similar to an inverse Fourier transform. (In standard FFT filtering, a sound is analyzed via the Fourier transform, the analysis data is multiplied by the frequency response of an FIR filter, and the result is inverse-transformed.)

A particularly rich example of the use of meta-parameters in conjunction with additive synthesis allows the user to describe the sound as a path through a multi-dimensional space of timbres. Each dimension of a line segment function describes the scaling of a particular timbre. These values are added and the results are used to scale the additive synthesis harmonics, producing complex interpolations (McNabb, 1981.)

A technique with even weaker parameters than additive synthesis is digital sampling, where each sample value can be viewed as a parameter. In fact, sampling can be considered the "identity synthesis technique," where the parameters are the samples themselves. Certainly, with the exception of introducing or removing a click, adjusting a single parameter (sample) does not produce a very noticeable effect. In practice, most samplers actually use a hybrid technique that includes filters, amplitude envelopes, and other more reasonably-parameterized processing modules.

#### How Physical are the Parameters?

We call a *physical parameter* one that not only mimics the behavior of a real-world instrument, but actually controls a synthetic instrument in the same manner as its real-world counterpart. Physical parameters are ideal for creating complex behaviors that arise during unstable moments in a tone's evolution, such as

during instrument attacks and transitions between notes. The beauty of techniques with physical parameters is that they don't require a formal analytical model or data base describing the behavior of the instrument under all circumstances. For example, neither the player/composer nor the implementer of the technique itself need understand exactly what happens in the output waveform during an attack or transition--the physical nature of the parameter causes the correct result.

One of the strongest arguments for physical modeling synthesis is that the parameters are exactly those used by a human instrumentalist. A waveguide violin model (Smith 1993) (one type of physical model) has bow pressure and bow velocity parameters; a clarinet model has a mouth breath pressure parameter (Cook 1988). Of course, playing a real-world violin or clarinet well is no trivial matter and requires years of training and practice. This same training may be needed, though in the virtual domain, to play the physical models (Chafe 1985.) For musicians who are not violinists or clarinetists, non-physical controls that are still intuitive may be more appropriate.

An example of a non-physical parameter is the amplitude of an additive synthesis overtone. Changing the amplitude of the third harmonic is not a very relevant control when attempting to synthesize a violin. While meta-parameters could in principal be defined to map pseudo-physical parameters to any underlying technique, we often don't have the acoustical knowledge necessary to adequately define this mapping.

#### How Well-Behaved are the Parameters?

Ideally, a change in a parameter produces a proportional change in the sound. If a small parametric change produces a wild unpredictable sonic result, the parameter is called "poorly-behaved" and can be a nightmare for composers and performers alike.

It is commonly thought that linear techniques, such as additive and subtractive synthesis, always have well-behaved parameters. However, such systems are linear only when the parameters change at slow rates relative to the sampling rate. When parameters change rapidly, energy is injected into the system, allowing the possibility of unpredictable or unexpected results. For example, consider a simple amplitude scaling that is allowed to be changed at an audio rate. This situation is equivalent to amplitude modulation (AM) synthesis. If the modulation is sinusoidal, two sidebands are produced for each component in the original signal, a non-linear effect. These sidebands are at frequencies corresponding to the frequency of each component plus or minus the modulating frequency. If the modulation is non-sinusoidal, the effect is even more complex. Especially dangerous are filter structures that can become unstable during transitions from one set of coefficients to another.

An example of a technique that exhibits excellent behavior during transitions is the waveguide-based vocal synthesis model of Perry Cook (Cook, 1990), which models the vocal tract as a series of waveguide filters, each representing a cross-section of the tract. Changing vowels has a physical interpretation--one or several sections shrink or increase in diameter. Interpolating between vowels corresponds directly to real-world spatial interpolation and all intermediate values have an intuitive, physical interpretation and are well-behaved.

Chaotic behavior can arise in non-linear feedback techniques (McIntyre and Woodhouse 1983), often used in physical models of wind instruments. In fact, this "extreme sensitivity to initial conditions" is often given as the very definition of a chaotic system. Still, with sufficient care, such a model can be made to operate in regions of its state space where it behaves predictably and effectively.

#### How Robust is the Sound's Identity?

Here we are concerned with how well the sound retains its identity in the context of variation. This is the author's personal favorite criterion. To do an effective musical instrument simulation--one that is more than a mere snapshot of a moment--it is essential to be able to synthesize "expression," which we define as "great variety in the context of a particular perceived source." For example, a violinist can make many changes in his or her sound, but the sound is still clearly a violin--it doesn't suddenly turn into a trumpet. Many synthesis techniques can approach a particular note on a particular instrument very closely, but fail miserably in making the family of perceptually-related sounds that is required for a true expressive instrument simulation.

Physical models do exceptionally well in this area. For example, the extended Karplus-Strong plucked string (Karplus and Strong 1983, Jaffe and Smith 1983), is a partially-physical model based on waveguides. It has the property that it sounds like a plucked or struck string no matter what you do to it. You can vary such parameters as pick position, string flexibility, string thickness, dynamics and decay characteristics to a great degree, providing a rich expressive vocabulary while never leaving the realm of string-like identity.

On the other end of the spectrum, pure sampling is a notorious offender in this area. A single trombone sample, on first hearing, is impressive because it sounds exactly like a trombone. But there is a strong tendency for a composer to assume it is an actual trombone, rather than a single snap-shot of a trombone. He or she may

be tempted to compose a variety of articulations, durations, dynamics and timbres only to discover that his or her supposed trombone is a two-dimensional cardboard cut-out of the real thing that falls over as soon as it is pushed lightly in any direction.

The power and flexibility of sampling can be greatly enhanced by using filters for dynamics and linear interpolation for timbral variety. For example, you can record a quiet trombone note and a loud trombone note and, if care is taken to match the pitch and phase exactly, do linear interpolation between the two to get a range of dynamics. If a huge amount of memory is available, another solution is to use a large library of samples that represent the trombone in all of its guises, including various attacks, dynamic contours and vibratos. Nevertheless, despite its seductive realism, sampling presents both a challenge and an opportunity to the composer attempting to use it in an expressive manner.

#### How Efficient is the Algorithm?

Efficiency is an extremely important criterion. In a real-time context, it determines the number of voices that are possible on a given piece of hardware. In non-real time contexts, it determines the amount of time a composer must wait before hearing the results of a computation. Given a fixed amount of time to complete a piece, an extremely long turn around time translates into fewer iterations, resulting in a less refined result.

Determining the efficiency of an algorithm is more complicated than it might first appear. It is not merely a matter of comparing processing benchmarks. Numerous aspects of a technique and its implementation come into play. We divide these into three categories: memory requirements, processing details and control stream attributes.

#### Memory

It is a well-known axiom of computer programming that you can often trade off memory against processing power. For example, when doing wavetable synthesis (Mathews 1969), in which a single period of a waveform is stored in memory, you can store a huge table and use a non-interpolating ("drop-sample") oscillator. Alternatively, you can store a smaller table and use a more expensive oscillator that interpolates between samples in the table. For that matter, using a wavetable oscillator at all is a memory optimization. A wavetable oscillator multiplied by an amplitude envelope can be replaced by a pre-computed version of the entire resultant waveform; the result is less real-time computation at a much greater memory cost. On the other end of the spectrum, the waveform can be computed analytically using any of a number of techniques. For example, if the wavetable is a sine wave, three alternatives to wavetable-based oscillators are marginally stable two-pole filters, evaluation of a complex phasor (Gordon and Smith 1985) and waveguides (Smith and Cook 1992.)

Some techniques have a memory requirement that changes with the parameter values. The Karplus-Strong plucked string, as well as several other waveguide-based modeling technique requires more memory for lower pitches than for higher pitches.

#### Processing

The issue of processing power itself is quite complex and depends to some degree on the details of the processor architecture. Some traditionally expensive techniques, such as finite element modeling (numerical integration of the difference equations that describe masses and springs), are well-suited to parallel architectures such as array processors. As another example, an algorithm with an active code size that fits within the cache of a RISC chip will run many times faster than one that overflows the cache (Freed, Rodet and Depalle 1993.) Techniques with minimum changes in program flow are well-supported by DSPs and other heavily pipelined architectures. Of course, special-purpose hardware can increase enormously the efficiency of a technique.

Just as memory usage can be dependent on parameter values, some techniques have a processing requirement that changes with the parameter values. The time-domain implementation of the Chant FOF (Rodet 1984), which does voice synthesis (as well as synthesis of other resonant systems) by adding up overlapping vocal tract impulse responses, becomes more expensive as the frequency rises--there are more pitch periods per second, and thus more additions and table-lookups per output sample.

Numerical characteristics also come into play. Can a technique be implemented in fixed point or does it require a large dynamic range? An algorithm that requires floating point has a higher cost associated with it on most systems. For example, some filtering structures are easier to implement in a floating point environment where overflow of intermediate values need not be a concern.

A related but different issue is how many bits of precision are required? It is important not to confuse

precision and dynamic range. When an algorithm uses floating point, it is trading off precision for dynamic range, assuming a constant word size. For example, 32-bit unsigned floating point typically has 24 bits of precision, while unsigned 32-bit integers have a full 32 bits of precision. Thus, on a 32-bit machine, integers actually have more precision than floating point numbers.

Some algorithms change their behavior depending on the word precision. This is especially likely with non-linear recursive systems, such as non-linear steady state oscillations in physical models, where small deviations in either calculation accuracy or initial conditions can completely alter the path and final state of such systems. In fact, it may make the difference between oscillating at all or not oscillating.

Another example of word precision affecting algorithm behavior is in regard to "limit cycles," annoying oscillations that arise at the end of an exponential decay and continue forever. If rounding is used, even the convergent rounding used in the DSP56001, which guarantees there will be no bias accumulated by the rounding, it is likely that limit cycles will arise. Since limit cycles tend to be confined to the lower bits, the more word precision, the less objectionable the limit cycles. One way around this is to implement truncation toward zero instead of rounding. But this tends to make exponential decays faster. For example, an exponential decay degenerates to linear in such systems; an exponential ratio constant of 0.99999999 just causes a 1 to be subtracted from the magnitude each sample, yielding a linear decay (Cook 1993.)

#### Control stream

The third efficiency consideration is the heaviness of the required control stream. Even an inexpensive algorithm, if it has a very dense control stream, may be difficult to implement in real time. Many systems use two processors, one dedicated to sound computation and the other handling control data. The difficulty comes from the cost of getting the control data to the sound processor. Even if there is only one processor, the control stream may consist of many mega-bytes of data, more than can fit in RAM, so disk space and disk bandwidth become scarce resources. Compression may be of use here.

When evaluating an algorithm's control stream requirements, it is important to differentiate between techniques that require dense bursts of parameter update messages as opposed to those with relatively steady streams of messages. A well-spaced fairly-dense control stream may be easily managed by some systems that fail when the same volume of control data is clumped in large bursts. For example, if an amplitude envelope changes with each note, it may actually be more efficient to feed the break-points to the processor one point at a time, rather than sending the entire envelope in a burst at the beginning of each note. This leads to a closely related issue...

#### How Sparse is the Control Stream?

In designing, implementing or using a synthesis technique, it is important to keep in mind where the actual work is being done. Is it the synthesis method or the control data that is actually doing the work? This is an often-overlooked distinction. Some synthesis techniques require such a bulk of control data that it actually exceeds the number of samples synthesized. This state of affairs is not necessarily bad, because the control data may be in a more useful parameterized form, as in the case of phase vocoder-based additive synthesis (Dolson 1986, Portnoff 1977.) Nevertheless, it is important to keep this criterion in mind. In fact, many synthesis techniques can be made nearly equivalent when fed enough data. A single sine wave that is frequency-modulated very quickly with a complex signal can produce intelligible speech. We can easily imagine absurd synthesis techniques where all the information is in the control stream. An example is the "modulated constant technique," defined as the number 1.0 multiplied by an "appropriate" control stream!

It is hard to come up with an example that ideally optimizes this criterion because the amount of control information that is necessary (a negative characteristic) is often directly proportional to the amount of control that is possible (a positive characteristic.) The best technique here is one where everything is pre-determined, a situation that is, by definition, non-interactive. Once interactivity is removed from the picture, very little information is required. For example, there is an old joke where a sailor says "I know two songs. One is Yankee Doodle. And the other isn't." Thus, with one bit you can represent both Beethoven's Ninth Symphony and the entire Ring Cycle of Wagner. Similarly, most MIDI synthesizers are designed for a minimum control stream because of the limitations of MIDI itself. This is not necessarily an advantage, since it limits the expressive bandwidth between the player and the synthesis. Still, especially in real-time implementations, control stream density must be monitored carefully or it can become a major bottleneck. Control stream optimizations, such as suppressing duplicate parameter-setting messages, lazy garbage collection and shared resource management can significantly reduce the density (Jaffe 1990.) Meta-parameters, such as the additive brightness parameter described above, can be defined. These are then expanded into lower

level parameters on the processor itself, thus reducing dramatically the control bandwidth. This is actually one form of control stream compression.

Perhaps the best way to look at the criterion of control stream density is to hold the amount of possible control constant, and proceed to minimize the control stream density for that amount of possible control. Physical models excel in this regard, since they tend to have a small number of powerful parameters.

Many control stream optimizations are specific to particular techniques. Pure phase vocoder-based synthesis (oscillator bank resynthesis), as well as short-time inverse Fourier synthesis (inverse FFT resynthesis) (Rodet and Depalle 1992), are extremely data-intensive techniques. Yet, there are ways this cost can be reduced. First, if an appropriate window is chosen, the window may skip ahead in the overlap-add process. This produces a dramatic decrease in the amount of control data. Further decreases come from the field of psycho-acoustics. In frequency domain techniques such as MPEG audio encoding (MPEG 1991), partials may be omitted if they are of low enough amplitude that they are entirely masked by neighboring partials. Similar techniques make possible the DCC and mini-disk consumer audio formats. However, such data reduction should be avoided if post-processing is planned, since a partial that was originally masked may be exposed during the post-processing phase. Finally, an efficient alternative to the phase vocoder has recently been developed (Serra and Smith 1990), that separates the analyzed sound into deterministic and stochastic components. It then models the deterministic portion using traditional phase vocoder methods, while modeling the stochastic portion with filtered random noise. For a wide class of real-world sounds, this results in a dramatic reduction not only in the control stream density, but also in the synthesis computation itself.

#### What Classes of Sounds can be Represented?

Is there a class of sounds that is impossible to produce with a given synthesis technique? While some techniques can produce any sound, given appropriate control data, others are less general. For example, inharmonic sounds are impossible with pure wavetable synthesis. However, once you add frequency or amplitude variation, which is almost always used to some degree, inharmonic sounds become possible. If you have an amplitude or frequency envelope with a rapid attack, you are actually producing inharmonic partials momentarily during the attack portion of the sound. For some synthesis techniques, the lack of flexibility can actually be viewed as an asset. A particular physical model, say a brass instrument, may synthesize only brass sounds. But this is exactly the property of "identity in the context of diversity" that was lauded as a great advantage in the fifth criterion above ("How Robust is the Sound's Identity?").

#### What is the Smallest Possible Latency?

Some techniques have an inherent and unavoidable latency that may cause problems in an interactive situation, where a sound must be computed and played immediately in response to an asynchronous event. A typical example is a MIDI keyboard triggering a sound that is synthesized on the fly. The situation also arises in the processing of live sound. Clearly, the sound cannot be computed in advance because it has not happened yet. If the technique has an inherent latency, there will be no way of avoiding a latency between the onset of the incoming sound and the resultant output sound.

Techniques with minimum latency are those that can consume and produce samples one at a time, with the first output sample emerging as soon as the triggering event has occurred or the input sound has begun. Many of the techniques mentioned in this paper, including FM, waveguide-based physical modeling, additive synthesis and wavetable synthesis have this property. Sound processing techniques such as reverberation (as it is ordinarily implemented), flanging, and equalization similarly follow this rule.

Other techniques may require significant latencies. FFT-based sound processing methods with large block sizes have a latency on the order of the FFT size. Other techniques are simply impossible in real time, regardless of the processor speed. An example of such a technique is non-causal filtering. You can't make something laugh before you tickle it. Another example is playing a sound backwards. You have to wait for the sound to finish before you can even start playing it. The latency is the length of the sound itself.

#### Do Analysis Tools Exist?

When computer and electronic music was first invented, both the technical literature and the popular press were full of claims that now, finally, we could "make any sound." However, as composers started exploring the space of parameter values, they discovered that most of that landscape produced sounds of undistinguished character. For example, although all excerpts of white noise have different sample values, they sound very much the same. It is not enough to know in theory that any sound can be produced. You need tools to derive the proper parameter values from a specification of a desired result. This process can be manual

or automated, as well as intuitive or analytical.

The question we ask of our synthesis technique, then, is whether well-understood analysis techniques exist for deriving parameters from a real-world description? While all synthesis is certainly not directed at imitating real-world sounds, this class of sounds is of great interest to many composers. Coming up with the right parameters to induce a technique to make the sound that in theory it is supposed to be able to make can be tricky, to say the least.

It is generally easiest to come up with an analysis technique for a synthesis technique that is capable of exactly reproducing any sound. The phase vocoder analysis/synthesis system is an example of such a technique. It derives additive synthesis parameters from the samples of a recorded waveform, then reproduces exactly those samples. We call this the "identity property."

Another popular analysis technique is linear prediction (LPC) (Markel and Grey 1976), which designs a series of recursive filters for subsequent resynthesis. Like the phase vocoder, it starts from a recorded waveform, but it can produce only an approximation of some input sounds. As an example, nasal vocal sounds are not well-represented by linear prediction. And while there are other analysis techniques such as Prony's method (Prony 1795) that attempt to take this into account, all are approximations--none has the identity property of the phase vocoder.

Yet, the identity property is only part of the picture. Unless mere data compression is the goal, composers want not only to derive parameters from real-world examples, but also to modify these parameters to create related but different sounds. One of the most basic transformations is to change the pitch of a sound. In this area, LPC has a significant advantage over the phase vocoder. While transposing a phase vocoded note also distorts its spectrum, LPC allows the pitch of the sound to be easily modified with no change to the spectral envelope. To do the same thing to phase vocoded data requires first fitting a spectral envelope to the data, then computing a new spectrum under that envelope--a much more computationally-intensive process. The reason LPC is superior in this case is the fact that the fundamental frequency is decoupled from the spectrum, while with the phase vocoder, the two are merged.

Samples of a waveform represent only one possible input to an analysis system. One can imagine techniques for deriving synthesis parameters and algorithms from other sorts of descriptions, such as a spatial representations of physical sound-producing mechanisms.

#### Conclusions

It's safe to assume that all existing synthesis/processing techniques have some benefit, otherwise they would have died out from lack of use. The plethora of hybrids now commonly in use shows that various techniques can be combined to maximize strengths and minimize weaknesses from several areas. This hybridization typically shows up after a technique has been around for some time and its characteristics have been extensively explored. FM is a case in point. Even the DX7 featured multi-carrier FM, which is actually a hybrid between FM and additive synthesis. This model turned out to be particularly useful for synthesis of formant structure in the human voice (Chowning 1989.) The NeXT Music Kit (Smith, Jaffe and Boynton 1989) includes a hybrid wavetable/FM synthesis in which the carrier and modulators can have arbitrary waveforms. Thus the ability to match a given periodic spectrum, which is provided by wavetable synthesis, is combined with the dynamic quality of FM. A similar capability is provided by Waveshaping synthesis (Le Brun 1979.)

In the final analysis, the appropriateness of a given technique depends on the task at hand. To quote the cellist who fruitlessly tries to teach loser Fielding Melish in the Woody Allen film "Take the Money and Run":

*"He has no conception of the instrument--he blows into it!"*

The moral is to avoid wasting time blowing into a cello--get a bassoon. On the other hand, blowing into a cello can occasionally have its own peculiar rewards and lead to discovery of unforeseen sound treasures.

#### References

- (Chafe, 1989) Chris Chafe. "Simulating Performance on a Bowed Instrument." *Current Directions in Computer Music*, M. Mathews, ed., MIT Press, Cambridge, MA.
- (Chowning, 1989) John Chowning. "Frequency Modulation Synthesis of the Singing Voice." *Some Current Directions in Computer Music Research.*, Cambridge MA, MIT Press, pp. 57-63.
- (Cook, 1993) Perry Cook. Personal communications.

- (Cook, 1988) Perry Cook. "Implementation of Single Reed Instruments with Arbitrary Bore Shapes Using Digital Waveguide Filters." *CCRMA Dept. of Music Report* No. STAN-M-50.
- (Cook, 1990) Perry Cook. "Identification of Control Parameters in an Articulatory Vocal Tract Model, with Applications to the Synthesis of Singing." Ph. D. Dissertation, Elec. Eng. Dept., Stanford University.
- (Chowning and Bristow, 1986) John Chowning and David Bristow. *FM Theory and Applications*. YAMAHA Music Foundation, Tokyo.
- (Dolson, 1986) Mark Dolson. "The Phase Vocoder: A Tutorial." *Computer Music Journal* 10(4):14-27.
- (Freed, Rodet and Depalle, 1993) Adrian Freed, X. Rodet and P. Depalle. 1993. "Synthesis and control of hundreds of sinusoidal partials on a desktop computer without custom hardware." *Proceedings of the International Conference on Signal Processing Applications and Technology*.
- (Gordon, 1985) John Gordon and J. O. Smith. 1985. "A Sine Generation Algorithm for VLSI Applications." *Proceedings of the 1985 International Computer Music Conference*.
- (Jaffe, 1990) David Jaffe. "Efficient Dynamic Resource Management on Multiple DSPs, as Implemented in the NeXT Music Kit." *Proceedings of the 1990 International Computer Music Conference*, Glasgow, Scotland, Computer Music Association, pp. 188-190.
- (Jaffe and Smith, 1983) David Jaffe and Julius Smith. "Extensions of the Karplus-Strong Plucked String Algorithm." *Computer Music Journal* 7(2):56-69. Reprinted in *The Music Machine*, Roads, C., ed., MIT Press, 1989.
- (Karplus and Strong, 1983) Kevin Karplus and Alex Strong. "Digital Synthesis of Plucked String and Drum Timbres." *Computer Music Journal* 7(2):43-55. Reprinted in *The Music Machine*, Roads, C., ed., MIT Press, 1989.
- (Le Brun, 1979) Marc LeBrun. "Digital Waveshaping Synthesis". *Journal of the Audio Engineering Society* 18(2):250-266.
- (Markel and Gray, 1976) J. D. Markel and A. H. Gray. *Linear Prediction of Speech*, Springer-Verlag, Berlin Heidelberg.
- (Mathews, 1969) Max Mathews. *The Technology of Computer Music*. MIT Press, Cambridge MA.
- (McIntyre and Woodhouse, 1983) McIntyre and Woodhouse. "On the Oscillations of Musical Instruments," *Journal of the Acoustical Society of America* 63(3):816-8253.
- (McNabb, 1981) Michael McNabb. "Dreamsong: the Composition." *Computer Music Journal* 5(4):36-53.
- (ISO-IEC, 1991) International Standards Organization ISO-IEC. 1991. "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbits/s." *Motion Picture Experts Group Audio Specification ISO-IEC JTC1/SC29/WG11*, Documents 3-11171, 3-11172.
- (Portnoff, 1977) M.R. Portnoff. "Implementation of the Digital Phase Vocoder Using the Fast Fourier Transform." *IEEE Trans. on Acoustics, Speech, and Signal Processing* ASSP-25 pp. 235-238.
- (Prony, 1795) R. Prony. "Essai experimental et Analytique sur les lois de la dilatabilite des fluides elastiques et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alcool, 'a diff' erentes temperatures." *J. Ecole Polytech*, Paris, 1:24-76.
- (Rodet, 1984) Xavier Rodet. "Time Domain Formant Wave Function Synthesis." *Computer Music Journal* 8(3):9-14.
- (Rodet and Depalle, 1992) Xavier Rodet and P. Depalle. "Spectral Envelopes and Inverse FFT Synthesis," *Proceedings of the Audio Engineering Society*, San Francisco.
- (Schottstaedt, 1977) William Schottstaedt. "The Simulation of Natural Instrument Tones using Frequency Modulation with a Complex Modulating Wave." *Computer Music Journal* 1(4):46-50.
- (Serra and Smith, 1990) Xavier Serra and Julius O. Smith. "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic plus Stochastic Decomposition," *Computer Music Journal* 14(4):12-24.
- (Smith, 1993) Julius O. Smith "Physical Modeling using Digital Waveguides." *Computer Music Journal* 16(4):74-87.
- (Smith, 1991) Julius O. Smith. "Viewpoints on the History of Digital Synthesis." *Proceedings of the International Computer Music Conference*, Montreal, pp. 1-10.
- (Smith, Jaffe and Boynton, 1989) Julius Smith, David Jaffe and Lee Boynton. "Music System Architecture on the NeXT Computer." *Proceedings of the Audio Engineering Society Conference*, Los Angeles.
- (Smith and Cook, 1993) Julius O. Smith, Perry. R. Cook, "The Second-Order Waveguide Oscillator." *Proceedings of the International Computer Music Conference*, San Jose, California, pp. 150-153.

### Acknowledgements

We have attempted to present a balanced view of the synthesis techniques discussed. Nevertheless, our bias toward physical modeling techniques can not be hidden. Also, we apologize to those researchers and techniques not explicitly cited in this article--the choice was biased by the techniques with which the author has the most familiarity, having implemented them himself or used them in musical compositions. Thanks to Julius Smith for consultations and suggestions.



## The Music Kit on a PC

DAVID A. JAFFE, JULIUS O. SMITH and NICK PORCARRO  
*david@jaffe.com, jos@ccrma.stanford.edu and nick@ccrma.stanford.edu*  
Stanford University Office of Technology Licensing and  
the Center for Computer Research in Music and Acoustics  
Stanford University, Stanford, CA 94305

### Abstract

We have recently ported the Music Kit to the Intel PC NEXTSTEP architecture, using inexpensive DSP and MIDI cards. We describe the port, which is soon to be released as Music Kit 5.0. We also introduce SynthBuilder, a graphic Music Kit instrument design and performance system. Finally, we discuss future plans for the Music Kit architecture.

### What is NEXTSTEP?

In 1989, NeXT Inc. introduced NEXTSTEP, an operating system environment based on an object-oriented architecture. The promise to developers was an environment that fostered rapid application development with fewer lines of code and higher levels of reusability than ever before. In addition NEXTSTEP is an exceptionally convenient and powerful operating system for users. At that time, NEXTSTEP ran exclusively on NeXT's own hardware, which included a Motorola DSP56001 signal processor, built-in 16-bit audio output, telephone quality microphone input, and two RS232 serial ports suitable for MIDI. In addition, NEXTSTEP included the Music Kit software package.

### What is the Music Kit?

The Music Kit is an object-oriented software system for building music, sound, signal processing, and MIDI applications running on the NEXTSTEP operating system. It has been used in such diverse commercial applications as music sequencers, notation packages, computer games, and document processors. Professors and students in academia have used the Music Kit in a host of areas, such as music performance, scientific experiments, computer-aided instruction, and physical modeling (Jaffe, 1991). The Music Kit is the first to unify the MIDI and Music V paradigms, thus combining interaction with generality. It was developed by NeXT Computer, Inc. from 1986 to 1991 and has been supported since then by Stanford University and developers such as Pinnacle Research, Inc. For further information, see (Jaffe and Boynton, 1989) and (Smith et al., 1989). The Music Kit is available free of charge, as described at the end of this paper.

### Is There Life After NeXT Hardware?

In 1993, NeXT announced discontinuation of its hardware line, choosing instead to focus on its software environment, which it has ported to a variety of architectures. The first port of NEXTSTEP was NS486, which runs on Intel 486 and Pentium chips. Soon after came NEXTSTEP for Hewlett-Packard HP-RISC. NeXT has recently announced ports to Sun and DEC architectures.

While we expect owners of NeXT hardware to continue to use the Music Kit on NeXT hardware, clearly the future of the Music Kit is to be found on other hardware architectures. We have chosen the Intel platform as our first port because of the many availability options and because of the price/performance advantages of the PC. We are also considering other ports, as discussed below.

### Hardware Architecture

The 5.0 release of the Music Kit features support for both Intel and NeXT hardware. The Intel support is based on the strategy of providing DSP, sound I/O and MIDI via inexpensive external PC cards with which the Music Kit communicates via software device drivers. For ease of porting and backward compatibility, we chose DSP cards with hardware similar to that of the original NeXT. These cards also have features beyond that of the original NeXT hardware, such as more memory, faster processor speed and built-in sound peripherals.

We currently support the Ariel PC56d (Ariel, 1994) and expect to soon support the ILink i56 (Ilink, 1994). Both are based on the Motorola DSP56001 and support a DSP serial port connection compatible with that of the original NeXT hardware. This allows a variety of external peripherals to be used with the Music Kit, including the Ariel ProPort (high-quality DAC/ADC), the Ariel DatPort and Stealth DAI2400 (digital I/O), the Singular Solutions AD64x (high-quality ADC and digital I/O), etc.

Both the Ariel and ILink cards feature DSPs that run faster than the original NeXT hardware, allowing for more processing or synthesis in real time. In addition, both include built-in CoDecs for DAC/ADC. The following chart summarizes the features of the cards, in comparison with the NeXT hardware.

Hardware:	NeXT	Ariel PC56d	Ilink i56
DSP fast static RAM:	8 K, 32 K or 192 K words	16 K or 64 K words	8 K words
Processor speed:	25 mhz.	27 mhz.	33 mhz.
CoDec:	8 bit mono	14 bit stereo	16 bit stereo
DSP can send IRQ:	Yes (unused)	No	Yes

Both the Ariel and ILink cards are priced at under 500 US\$ at the time of this writing.

For MIDI support, we use the standard MPU-401 architecture that is ubiquitous in the PC world. We currently support the MusicQuest MIDI cards. Depending on which version of the card you buy, it may include one or two MIDI inputs and one or two MIDI outputs.

### Software Architecture

#### Portable File Formats

Intel x86 and Motorola 68x00 processors differ in the order in which bytes are represented within words. Intel orders the bytes from low to high ("little endian"), while Motorola orders them from high to low ("big endian"). This is no problem for ASCII files such as the Music Kit .score file. However, binary files must be carefully handled to allow users to freely move files between systems. The Music Kit uses the following binary files:

1. **.dsp**—compiled DSP monitor files (load faster than .lod files)
2. **.sound**—sound files
3. **.playscore**—compiled score files (load faster than .score files)
4. **.midi**—Standard MIDI files

To prevent confusion, these files are *always* stored in big-endian order. This means there is a slight cost in reading a file on a little-endian system, since the bytes must be swapped. However, the swapping can be done quite rapidly and the overhead is not significant.

#### Portable Applications

Just as users may move files between architectures, they may also move applications. It is desirable for a given copy of a Music Kit application to work on both the NeXT and Intel machines. That way, a user can buy an application for his NeXT hardware and, when he buys an Intel machine, simply copy the file to his new computer.

To make this inter-operability possible, the Music Kit provides "fat libraries" and "fat applications." These are actually copies of the object code that includes both Intel and Motorola machine code. The NEXTSTEP

operating system then automatically chooses the appropriate code to execute for a given architecture.

#### Custom DSP Monitors

Another aspect of making it possible to move applications from one computer is for the application to be self-contained—it should have no dependency on the Music Kit run-time library being installed on a given machine. To make this possible, beginning in release 4.0, we have made it possible to include the DSP run-time monitor in the application "wrapper", i.e. the directory containing the application resources. This monitor, with a .lod (ASCII) or .dsp (binary) file extension, then travels with the application as it is moved from one computer to another.

In addition, the Music Kit supports variable DSP memory configurations. Each memory configuration has its own DSP monitor file. Release 4.0 included monitors for the 32k and 8k configurations of the NeXT DSP, as well as monitors for the hub and satellites of the Ariel QuintProcessor (Ariel, 1990). (The QuintProcessor is a five-DSP board for the NeXTcube, with the DSP arranged in a hub/satellite configuration. Each DSP has its own bank of fast static RAM and serial ports, and the hub processor has a large pool of DRAM.)

Release 5.0 introduces new monitors for the PC56d and i56 cards, as well as for the 192k expansion memory board for NeXT hardware. The Music Kit's Orchestra object automatically chooses a monitor that is optimal for a given machine. For example, on NeXT hardware, it probes for the 192k memory expansion card and, if such a card is found, uses the 192k monitor. If it finds no 192k card, it checks for a 32k card. If no 32k card is found, the default 8k monitor is used. On Intel hardware, the monitor is chosen based on the kind of driver that is installed.

The support for different monitors, coupled with the wrapper search convention, makes it possible (and suggested) for an application to include a set of monitors that allows it to run optimally on a wide variety of computers.

#### Communicating with the DSP

Communication with the DSP on Intel hardware is through a loadable device driver. The Music Kit provides drivers for each of the cards it supports. The user can easily install the driver with the Configure application, which allows I/O ports and interrupt requests to be set from a convenient graphic interface. The Music Kit automatically searches for an installed driver of the type "DSP driver" and assumes there is a matching card installed. The only concern of the user is that he avoid collisions between the IO ports and interrupt requests of the various cards in his computer. This is a familiar headache for PC users. Recent "plug and play" architectures are beginning to address it. We hope to make the process of installing cards easier as the technology evolves.

The Music Kit's Orchestra object supports any number of DSPs, which made it easy to port to the Ariel QuintProcessor (Jaffe and Smith, 1992). On the Intel hardware, at the time of this writing, we support use of one DSP card at a time. This is really only a limitation of the way in which the Music Kit senses the presence or absence of the DSP drivers. It can be easily made more flexible if there is enough interest. We are hoping that PC cards with multiple DSP56001s, similar to the QuintProcessor, will be made soon and we look forward to supporting them. As an example of how we handle such a multi-DSP card, we examine briefly the Ariel QuintProcessor support. Support for multiple cards would follow a similar pattern:

The primary Music Kit class that supports the QuintProcessor is the *ArielQuintProcessor* class, which serves a dual purpose. First, it is a subclass of Orchestra that represents the hub DSP. As such, it can be sent Orchestra messages to allocate UnitGenerators, SynthPatches, etc. But it also represents the QuintProcessor as a whole and provides master control methods. The satellite DSPs are represented by instances of the subsidiary class, *ArielQPSat*, which is also a subclass of Orchestra. Creating an instance of *ArielQuintProcessor* automatically creates the associated *ArielQPSat* objects. A developer can choose to allocate DSP resources on a particular DSP by sending the appropriate allocation messages to the appropriate *ArielQPSat* or *ArielQuintProcessor* object.

#### Sound Input and Output

Our experience implementing sound output on the QuintProcessor paved the way for our approach to sound

output on the Intel hardware. On the QuintProcessor, we implemented direct sound output from the hub DSP serial port, rather than doing sound output the way it was done on the original NeXT hardware, where sound traveled first from the DSP to the main CPU (68040), then from the main CPU to the NeXT sound hardware. This has a number of advantages, including virtually eliminating latencies due to sound buffering, simplifying the software architecture, and reducing the load on the main CPU.

We took the same approach on the PC. The sound output, as well as the sound input, is done via the NeXT-compatible DSP port of the DSP cards. Alternatively, the codecs on the cards themselves may be used. In either case, the main CPU of the controlling computer has no responsibility with respect to sound output and input—the card handles everything.

To use the DSP serial port, the programmer sends the Objective-C message `setSerialPortOutput:YES` to the Orchestra object that represents the DSP. No change is necessary to the SynthPatch or UnitGenerator code. Since all SynthPatches already have an output UnitGenerator (such as *OutLaUG*), the DSP system simply routes the sound from this output to the serial port. To use the internal codec of the card, the programmer sends `setSerialPortOutput:NO`.

The *DSPSerialPortDevice* class encapsulates the details about the external device that is plugged into the DSP serial port. The Music Kit provides subclasses of *DSPSerialPortDevice* support various commercially-available devices. For example, if you have an Ariel ProPort, you simply send the message `setSerialPortDevice:[ArielProPort alloc] init]` to the Orchestra object, which then defers to the *ArielProPort* object to set up the external device. The *ArielProPort* object also takes care of sending the appropriate commands to the DSP SCI port to set the sampling rate. The Music Kit's DSP system automatically handles half sampling rates. E.g. if the serial port device supports only 44100 and the sampling rate of the music is 22050, the Music Kit will automatically up-sample the sound data. If a hardware designer creates a new serial port device, he need only subclass *DSPSerialPortDevice* and override a few methods—the Music Kit then automatically supports the new device. One interesting new device is a quadraphonic interface, developed by Fernando Lopez-Lezcano, Stephen A. Davis and Atau Tanaka at CCRMA (Lopez-Lezcano, 1993) (Tanaka, 1991), which allows the Music Kit to generate four-channel sound.

The 4.0 Music Kit also added support for receiving 16-bit sound sent to the serial port from an external ADC such as the AD64x or ProPort, or an external AES/EBU interface such as the Stealth DAI2400. This sound input runs simultaneously with sound output, enabling incoming sound to be processed and sent back out the serial port.

Sound input support is enabled in a manner similar to sound output—you merely send `setSerialSoundInput:YES` to the Orchestra that represents the DSP. To create a sound-processing SynthPatch, you provide a sound input source UnitGenerator. For example, to receive sound input from the left channel, you include an *InLaUG* UnitGenerator object, just as you use an *OutLaUG* to send sound output to the left channel. The output of *InLaUG* can then be connected to any other UnitGenerators to create a signal processing network. Any number of *InLaUG*s may be instantiated; each gets a copy of the incoming sound so it is easy to create parallel banks of processing modules.

Since the SSI input and output may be used simultaneously, and since the SSI output path gets rid of buffer latency as described above, a real-time signal processor with no noticeable latency can be implemented using the Music Kit tools we have described.

## MIDI

To support MIDI, we have written a device driver that provides the same functionality as the driver for NeXT hardware, including time-stamping of input bytes, timed output of time-stamped bytes, support for multiple MIDI cables and synchronization to incoming MIDI time code. The driver supports the standard NeXT MIDI driver API, allowing it to be used by both Music Kit and non-Music Kit applications.

## Performance Evaluation

The greater speed of the higher-end Pentium systems, as well as the higher-performance DSP cards allow for an improvement in Music Kit efficiency. In particular, the Pentium speeds up such operations as inverse Fourier transforms, which are used in the Music Kit to convert from Partial objects (frequency-domain representation) to DSP wavetables (time-domain representation). Other operations that are improved include

soundfile mixing using the *mixsounds* command-line utility, parsing of long scorefiles, etc.

On both the NeXT and Pentium hardware, the DSP's "timed message queue" (TMQ), which controls the precise timing of events on the DSP, is of a fixed size and sometimes fills up. However, the larger amount of DSP static RAM on the PC56d allows for a larger timed message queue and thus minimizes the probability of the TMQ filling, in turn leading to better behavior than the NeXT hardware had with its minimum DSP memory configuration. The small size of the TMQ was enough of a problem on the NeXT hardware that we supported extensive buffering of DSP commands in the driver for NeXT hardware. While the larger memory size of the PC56d lessens the need for buffering on the main CPU, it remains to be seen whether it turns out to be necessary to add this extra level of buffering.

Even with a larger TMQ, in extreme situations, the TMQ may fill up. If that occurs, it is important for the driver to detect when room again becomes available in the TMQ. To accomplish this optimally, the DSP should be able to interrupt the main CPU to tell it there is room for more commands. However, the PC56d card is unable, to generate interrupts. The DSP driver for the PC56d card must poll the DSP periodically, to see when room in the TMQ becomes available again. This is not only inefficient, it can lead to less-than-ideal response, since there may be some time between the time that room becomes available in the TMQ and the time that the driver wakes up and notices it's time to send more data. Other cards, such as the i56, support interrupts and should improve performance in this area.

## SynthBuilder

One of the most exciting developments in the Music Kit project is a new graphic application for creating Music Kit instruments, configuring them to respond to MIDI, and performing them. Synthesizer "patches" are represented by networks consisting of digital signal processing elements called "unit generators" (Music Kit UnitGenerator objects) and MIDI event elements called "note filters" and "note generators" (Music Kit NoteFilter and Performer objects, respectively.) SynthBuilder is based on GraSP, a student project by Eric Jordan, created at Princeton University in 1992, with advisory assistance by the author (Jaffe), who was a visiting faculty. Since that time, it has been extensively developed by Nick Porcarro, in collaboration with the authors.

The graphical interface enables construction of complex patches without having to write a single line of code, and the underlying Music Kit software provides support for real-time DSP synthesis and MIDI. This means there is no "compute/then listen" cycle to slow down the process of developing a patch. It can be tried out immediately on a MIDI keyboard, and unit-generator and note-filter parameters can be adjusted in real time while a note is still sounding.

Sixteen bit stereo sound is synthesized immediately on the NeXT's built-in DSP56001 signal processing chip, and can be controlled from the user interface with software-simulated or physical MIDI devices. In addition to synthesis, the system supports configurations for sound processing via the DSP serial port as well as for sound output to DACs and other digital I/O devices.

MIDI control signals can be mapped to unit generator object control methods, permitting high-level control of patch parameters. For example, a MIDI key number can be readily mapped into frequency, and then mapped into a delay line length via a graphically constructed lookup table. A single MIDI event can be fed to (or through) multiple note filters, each of which can modify the event stream and/or control one or more unit-generator parameters.

Polyphony is handled in SynthBuilder by graphically specifying a voice allocation scheme. Optionally, a Music Kit SynthPatch can be generated and used in another application. Other types of code generation are possible, such as generic C code, or assembly code for another digital signal processor.

Dynamically loadable custom "inspectors" (user interfaces) can be created for patch elements. Dynamic loading promotes easy distribution and sharing of inspector modules, and promotes a fast, efficient development cycle. The process of creating a custom inspector is facilitated by a code generator, which takes a DSP assembly macro and a signal flow/parameter list specification as input, and outputs working interface code which can then be customized.

As of this writing, SynthBuilder had more than 50 graphical custom inspectors, including an envelope editor, digital filter responses editors, and a MIDI event lookup table.

SynthBuilder is being used by researchers at the CCRMA to explore new synthesis techniques. It is now in

the alpha release stage on both NeXT 68040 based systems and Pentium systems.

#### Beyond the 56001

Recently-announced new versions in the DSP5600x series promise much greater compute power. Additionally, there are plans to modify SynthBuilder to generate code for a variety of other DSP chips. In addition, as main CPU speeds continue to increase at their current rate, eventually it is likely that we will be able to run the Music Kit's synthesis and sound processing code on the main CPU itself, rather than on a DSP, using UnitGenerators written entirely in C. The object-oriented nature of the Music Kit makes such a change manageable because all references to the DSP are localized in the Orchestra, UnitGenerator and SynthData classes. We have made a prototype of the low-level portion of such a system, with all current Music Kit DSP unit generators translated into C and have been able to use it to do some simple interactive real-time synthesis on a Pentium-based computer. One of the advantages of doing this exercise is that it enabled a comparison between the Pentium and DSP-based solutions. Currently, the DSP came out many times faster and has the advantage of being dedicated exclusively to sound production. Nevertheless, many more hours have gone into optimizing the DSP implementation than the Pentium implementation. If the Pentium unit generators were written in assembly language, they would probably be more efficient. We plan to watch closely the developments in the area of both DSPs and RISC chips and plan our migration path appropriately.

#### Availability

The Music Kit is available via ftp from [ccrma-ftp.stanford.edu](ftp://ccrma-ftp.stanford.edu) (email: [musickit@ccrma.stanford.edu](mailto:musickit@ccrma.stanford.edu).) It is also available on CD ROM as part of the *Big Green CD ROM* at P.O. Box 471645, San Francisco, CA 94147 (email: [disc@skylee.org](mailto:disc@skylee.org), fax: 415 474 7896, phone: 415 474 7803.) At the time of this writing, the CD ROM contains version 4.0, which runs only on NeXT hardware. To subscribe to a Music Kit news group, send to [mkdlist-request@ccrma.stanford.edu](mailto:mkdlist-request@ccrma.stanford.edu).

#### References

- (Ariel, 1994) Ariel Corp, 433 River Road, Highland Park, NJ 08904. (201) 249-2900, (201) 249-2123 fax.
- (Ariel, 1990) Ariel Corp. *Ariel QuintProcessor Installation, Technical and Programming Manual*. Ariel Corp.
- (Ilink, 1994) i\*link. Kommunikationssysteme GmbH, Nollendorfstrasse 11-12, 10777, Berlin, Germany. phone: 49 30 - 216 20 48, fax: 49 30 - 215 82 74, mail: [info@ilink.de](mailto:info@ilink.de).
- (Jaffe, 1993) David A. Jaffe and Julius O. Smith. *Real Time Sound Processing & Synthesis on Multiple DSPs Using the Music Kit and the Ariel QuintProcessor*. Proceedings of the 1993 International Computer Music Conference, Tokyo, Japan.
- (Jaffe, 1990) David A. Jaffe. Efficient Dynamic Resource Management on Multiple DSPs, as Implemented in the NeXT Music Kit. Proceedings of the 1990 International Computer Music Conference, Glasgow, Scotland, Computer Music Assoc., pp. 188-190.
- (Jaffe, 1991) David A. Jaffe. Musical and Extra-Musical Applications of the NeXT Music Kit. Proceedings of the 1991 International Computer Music Conference, Montreal, Canada, Computer Music Assoc., pp. 521-524.
- (Jaffe, 1989) David A. Jaffe. An Overview of the NeXT Music Kit. Proceedings of the 1989 International Computer Music Conference, Columbus, Ohio, Computer Music Assoc., pp. 135-138.
- (Jaffe and Boynton, 1989) David A. Jaffe and Lee Boynton. An Overview of the Sound and Music Kits for the NeXT Computer. *Computer Music Journal*, MIT Press, 14(2):48-55, 1989. Reprinted in book form in *The Well-Tempered Object*, ed. Stephen Pope, 1991, MIT Press.
- (Jaffe and Smith, 1983) David A. Jaffe and Julius O. Smith. Extensions of the Karplus-Strong Plucked-String Algorithm. D. Jaffe and J. O. Smith. 1983. *Computer Music Journal*, 7(2):56-69. Reprinted in *The Music Machine*, ed. Curtis Roads, 1989, MIT Press, pp. 481-494.
- (Lopez-Lezcano, 1993) F. Lopez-Lezcano. A four channel dynamic sound location system. Proceedings of the 1993 Japan Music and Computer Society.
- (McNabb, 1990) Michael McNabb. Ensemble, An Extensible Real-Time Performance Environment. Proc. 89th Audio Engineering Society Convention, Los Angeles, CA, 1990.

- (Smith et al., 1989) Julius O. Smith, David A. Jaffe and Lee Boynton. Music System Architecture on the NeXT Computer. Proceedings of the 1989 Audio Engineering Society Conference, L.A., CA.
- (Tanaka, 1991) Atsu Tanaka. Implementing Quadraphonic Audio on the NeXT, Proceedings of the 1991 International Computer Music Conference, Montreal, Canada, Computer Music Assoc.

#### Acknowledgements

Work on SynthBuilder and the corresponding Music Kit support was provided by the Stanford Office of Technology Licensing. Support for the Music Kit was provided by the Stanford Center for Computer Research in Music and Acoustics.

## Sistemas de Notação Musical

## NotaCor - Impressão de Partituras em Cores

ALEX DE OLIVEIRA MEIRELES

*Laboratório de Processamento Espectral do Departamento de Ciência da Computação  
Universidade de Brasília - Brasília DF, Brasil, 70910-900*

### Resumo

Partituras musicais são estruturas de dados com certo grau de complexidade que podem ser analisadas e executadas por músicos através de algoritmos advindos da teoria/prática musical. Nem sempre os símbolos que compõem a partitura conseguem especificar com precisão e completude as informações necessárias a sua execução. Isto exige do músico, além de cuidadoso estudo prévio da partitura, conhecimentos extras sobre a interpretação e estilo da peça que não se encontram escritas. A utilização de cores na impressão das partituras permite enriquecer a semântica dos seus símbolos, ao mesmo tempo em que simplifica a notação e permite uma apresentação mais limpa das informações.

### Introdução

Entre os problemas de formatação de documentos, a formatação de partituras musicais é um dos mais complexos e menos estudados. Mesmo na atualidade, a impressão de partituras musicais é feita na maioria dos casos, pelos mesmos métodos utilizados no século passado. Mesmo em centros de pesquisa de ponta da área musical, recorre-se a métodos ultrapassados.

O Laboratório de Processamento Espectral (LPE), do Departamento de Ciência da Computação, desenvolve diversas pesquisas e projetos na área de Inteligência Artificial envolvendo Computação Sônica e Computação Gráfica. Diversos produtos foram gerados por estas pesquisas e são amplamente utilizados pela equipe do LPE nos seus estudos. Deste modo se desenvolvem as ferramentas que dão continuidade ao trabalho e abrem novas possibilidades de pesquisa. O presente trabalho pretende ser também uma ferramenta prática que possa ser de utilidade para as futuras pesquisas a serem desenvolvidas pela equipe, e que facilite o curso das atividades atualmente conduzidas.

O LPE tem entre suas pesquisas a geração de peças musicais por computadores, ou seja, o laboratório produz músicas que são criadas através de composição algorítmica. Estas peças podem ser facilmente executadas pelo próprio computador que trata apenas das notas e seus respectivos parâmetros, mas para um observador humano, a leitura/interpretação destas peças é muito difícil pois as máquinas apresentam como saída apenas uma seqüência de dados que envolvem freqüência, duração, dinâmica e outras mais que em geral são apenas números. Ora, para um músico uma seqüência de números dificilmente representa música, por isso se faz necessária uma interface mais natural entre máquina e homem no que se refere à saída visual das peças musicais geradas. Portanto, este é um dos objetivos do presente projeto: gerar uma saída impressa das músicas geradas no LPE em forma de partituras musicais para que possam ser lidas e executadas com mais propriedade por seres humanos.

Além da geração automática de partituras musicais pelo computador, busca-se no LPE pesquisar campos pouco explorados, tal como a geração de partituras em cores. Porém, não se quer apenas partituras contendo cores, mas busca-se novas formas de expressão e notação musical, pois neste caso os cromatismos não serão mero adorno mas expressarão a dinâmica da música e também a possibilidade de percepção da evolução melódica segundo um mapeamento som-luz a ser descrito posteriormente. Isto permitirá uma apresentação mais compacta, limpa e expressiva das músicas, quando comparadas às representações tradicionais.

### Abordagem

As peças musicais geradas de forma algorítmica são apresentadas pelo computador na forma de *cartas espectrais* chamadas CAR e também por cartas CES, que são cartas CAR com instrumentação sinfônica (Gioia 1994). Estas contêm os parâmetros necessários para a descrição das músicas, tais como frequência, duração, dinâmica e tempo inicial. O sistema a ser descrito lê estas cartas e a partir delas gera um arquivo em linguagem PostScript. Este arquivo PostScript definirá as partituras já em cores, que poderão ser então impressas num dispositivo adequado ou observadas na tela com o auxílio de um sistema de visualização PostScript.

### Conceitos Aplicados ao Sistema NotaCor

#### Composição Algorítmica

Quando se diz que uma das áreas da computação sônica é a síntese de sons, isto não se refere somente a produção de timbres e a manipulação digital de sons. O estudo da coordenação de sons também é objeto de estudo desta área e existem diversos tipos de sistemas que auxiliam neste objetivo. Os modos de se produzir música podem variar mas o resultado pretendido é o mesmo, que é composição algorítmica. O computador pode assumir diversos papéis, desde um simples auxiliar funcionando como um editor ou um verificador, pode auxiliar na busca de caminhos a serem seguidos pelo compositor, ou pode tomar o controle após uma inicialização ou mesmo gerar praticamente sozinho as peças.

#### Impressão de Partituras Musicais

Este é o tema primordial deste trabalho. Ao se abordar este problema, deve-se sempre ter em mente a busca da qualidade da impressão e o respeito às regras e notações musicais, pois de outro modo o resultado não seria satisfatório em termos profissionais, ou não se poderia utilizá-lo por não atender às exigências da escrita musical.

Para se buscar qualidade, deve-se considerar o melhor resultado possível a ser obtido. Não podemos resolver o problema apenas para um dispositivo de baixa resolução (qualidade) pois quando estiver disponível um dispositivo melhor, o resultado será no máximo igual ao do dispositivo para o qual foi projetado. Ou seja, uma boa impressora não pode "melhorar" o resultado se este foi originalmente definido para um resultado de pior resolução. Neste sentido adotou-se que a saída do sistema seria na forma de arquivos PostScript. O PostScript é uma linguagem de impressão, ou seja, destina-se a impressoras e possui comandos para a definição dos elementos de página, tais como, letras, traços e formas geométricas. Além disso, o PostScript é independente de dispositivo, ou seja, o resultado, se nele definido será tanto melhor quanto maior for a resolução do dispositivo de saída desde que este suporte a linguagem. Deste modo tem-se o melhor resultado possível, o que concorda com os objetivos.

Outro problema envolvendo a impressão de partituras musicais é a questão dos símbolos, ou seja, de como se desenhar notas, claves e outros elementos musicais. A qualidade é importante neste ponto, e se mal desenhados, os símbolos podem se tornar até ilegíveis, o que invalidaria o trabalho por não permitir uma boa leitura da peça. A solução encontrada e adotada foi a utilização de uma fonte musical PostScript (conjunto de caracteres definidos para música). A sua qualidade atende às nossas necessidades e por ser uma fonte PostScript, é totalmente compatível com o procedimento adotado até agora.

Um problema particularmente interessante é a criação do algoritmo de distribuição dos símbolos musicais na partitura. Ele é muito semelhante ao problema de formatação de texto mas com regras mais particulares à notação musical. Resolvidos estes problemas, tem-se agora o problema de notação, pois a música possui regras no que se refere à colocação de seus símbolos em uma partitura, e para que as partituras geradas pelo sistema tenham utilidade prática, elas devem ser fiéis à notação musical.

#### Som e Cor

Para o embasamento do trabalho é necessário reportarmos-nos ao artigo "Síntese de Imagens com Pedacos de Tempo" de Aluizio Arcela. Ele demonstra que a quantização do tempo e da luz são fenômenos com uma correlação matemática e perceptual. De maneira simplificada seu artigo nos permite dizer que as 10 (dez)

oitavas audíveis de som podem ser quantizadas em uma oitava de luz visível. Deste modo, a frequência de uma nota dentro de uma oitava é mapeada para o matiz; a oitava em que se encontra a nota (de 1 a 10 oitavas) é mapeada para a saturação; e a amplitude do som é mapeado para o brilho.

#### Partituras em cores

O sistema habitual de notação musical está sempre em contínua modificação à medida que a música se desenvolve. Com o desenvolvimento da tecnologia musical, a notação foi necessariamente modificada. Novos aprimoramentos nessa notação são feitos à medida em que novas formas surgem. Mas é também possível pensarmos que modificações realizadas na notação atual, mesmo sem o acréscimo nas formas musicais, possam vir a trazer contribuições à música. A sintaxe atual não tem toda a capacidade de trazer em si a semântica necessária à execução de uma certa peça. Percebe-se também que esta sintaxe não facilita muito a sua interpretação dada a disposição de seus elementos (Figura 1). Tenta-se mapear um elemento de várias dimensões em apenas duas dimensões.



Figura 1: Partitura na forma atual de codificação.

Neste trabalho, estuda-se uma modificação na notação atual de forma a simplificá-la sem perda de conteúdo semântico. Essa simplificação pode vir a permitir a aquisição de outros conteúdos pela introdução de novos elementos sintáticos. De outro modo, poder-se-ia pelo menos ter simplificado a sintaxe musical. Como foi visto, é possível fazer um mapeamento das oitavas e intensidades (frequências e intensidades, o som menos sua duração) em matizes, saturações e brilhos de cores. Neste trabalho aplica-se o mapeamento na codificação de partituras. Em lugar da representação da dimensão de intensidades (a dinâmica, figura 2), i. e., *ppp*, *mp*, *ff*, etc, utiliza-se a saturação das cores para a codificação da oitava do som. Além disso, tem-se uma ênfase das alturas das notas pelo matiz utilizado. A partir dos parâmetros de frequência e intensidade calcula-se o matiz, a saturação e o brilho de cada nota. Ao imprimir-se a nota, ela é pintada.

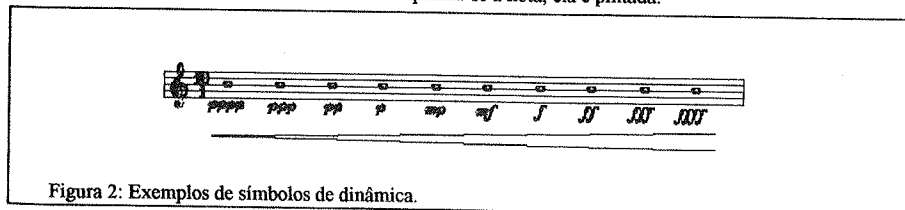


Figura 2: Exemplos de símbolos de dinâmica.

### Recursos de Hardware e Software

Neste projeto o sistema será desenvolvido tendo como plataforma máquinas SUN utilizando o sistema Unix. Para visualização dos resultados da utilização do sistema serão necessários vídeo de média/alta resolução e/ou impressora que aceite entrada PostScript. Dado que o projeto trabalha com saída impressa em cores será necessária uma impressora colorida. O sistema será implementado em linguagem C e em linguagem PostScript. Deve-se atentar para a utilização de um interpretador Postscript nível II. Serão usados o compilador C da Sun (SUN Systems), o software DevGuide (SUN Systems) para desenvolvimento da interface e o compilador CPS para a geração de código C a partir de rotinas PostScript (Adobe Systems). No caso do módulo de interface gráfica serão utilizadas bibliotecas gráficas adequadas a cada plataforma (Xview para SUN da SUN Systems).

### Estrutura do Sistema

O sistema será organizado em 4 módulos lógicos inclusos no mesmo código executável. A execução será coordenada pela interface do sistema. As entradas do sistema são arquivos tipo CAR ou CES e a configuração feita na interface. A saída será um arquivo EPS que conterá as informações de impressão da partitura. A seguir uma descrição das funções genéricas de cada módulo (figura 3).

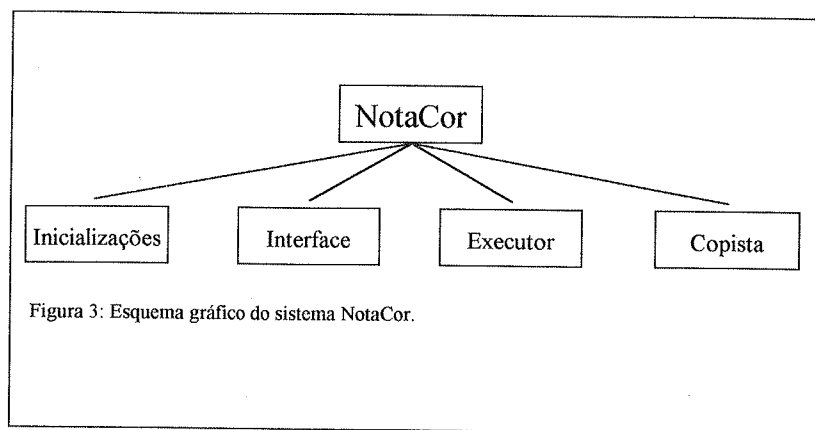


Figura 3: Esquema gráfico do sistema NotaCor.

#### Interface

É responsável pelo controle de execução do sistema. Permite a seleção dos arquivos de entrada e saída. Tipo de entrada (Car ou CES); início da execução; chamada de impressão, configurações tais como especificações de página (tamanho de papel, número de instrumentos, etc), diretórios default, etc. Este módulo controla as chamadas aos outros módulos do sistema.

#### Inicializações

É responsável pela criação dos objetos da interface e inicialização de variáveis.

#### Executor

É responsável pela leitura e interpretação dos arquivos de entrada, pelo tratamento dos objetos e pelo espaçamento dos objetos musicais.

#### Copista

É responsável pela compilação dos objetos musicais e seus respectivos espaçamentos em objetos gráficos na linguagem PostScript.

### Algoritmo Básico

- Inicialização de todos os objetos da interface;
- Entrada de nome de arquivo CAR ou CES;
- Parametrização do número de compassos, linhas por página, sistemas por página;
- Leitura do objetos musicais e conversão de frequências em cores;
- Transformação de objetos musicais em listas de notas em memória;
- Determina-se a distribuição das notas em compassos da partitura;
- Determina-se a união ou não das notas em termos de grafia musical;
- Ajusta-se o espaçamento horizontal das notas;
- Ajusta-se o espaçamento vertical das notas;
- Gera-se saída PostScript em cores (figura 4).

### Exemplo de arquivo CAR

```

(VAL 0 60 10000 0.1041666 1 1 720)
(INS 80 I1
(1 210.1316 ((0 0) (4379 180) (-2911 360) (0 720)) 0)
(0.3016 211.1962 ((0 0) (-2049 180) (-9339 360) (0 720)) 0)
(1.6969 211.1803 ((0 0) (3640 180) (0 720)) 0)
(0.7468 328.3099 ((0 0) (-6315 180) (0 720)) 0)
)
(INS 80 I2
(1 264.5851 ((0 0) (5067 180) (10339 360) (0 720)) 1)
(2.1143 274.7800 ((0 0) (772 180) (6044 360) (0 720)) 1)
(0.2324 271.4896 ((0 0) (-1263 180) (0 720)) 1)
(0.9991 272.4606 ((0 0) (-3817 180) (0 720)) 1)
)
(INS 80 I3
(1 264.5851 ((0 0) (5067 180) (10339 360) (0 720)) 1)
(2.1143 274.7800 ((0 0) (772 180) (6044 360) (0 720)) 1)
(0.2324 271.4896 ((0 0) (-1263 180) (0 720)) 1)
(0.9991 272.4606 ((0 0) (-3817 180) (0 720)) 1)
)
(INS 80 I4
(1 264.5851 ((0 0) (5067 180) (10339 360) (0 720)) 1)
(2.1143 274.7800 ((0 0) (772 180) (6044 360) (0 720)) 1)
(0.2324 271.4896 ((0 0) (-1263 180) (0 720)) 1)
(0.9991 272.4606 ((0 0) (-3817 180) (0 720)) 1)
)
(EXE 0 576)
(I1 0 16 47.7502 1.0850)
(I2 0 16 74.9939 1.0850)
(I3 0 9 239.2969 0.5865)
(I4 0 9 769.5821 0.5865)
(I3 9 3 153.0344 0.4604)
(I2 9 3 487.4481 0.4604)
(I1 12 9 237.7306 0.5494)
(I4 12 9 768.7121 0.5494)
(I3 16 16 14.9417 1.0748)
(I3 12 9 237.7306 0.5494)
(I4 12 9 768.7121 0.5494)
(I3 16 16 14.9417 1.0748)
(STP)
(FIM)
  
```



## Projeto de Interface do NotaCor

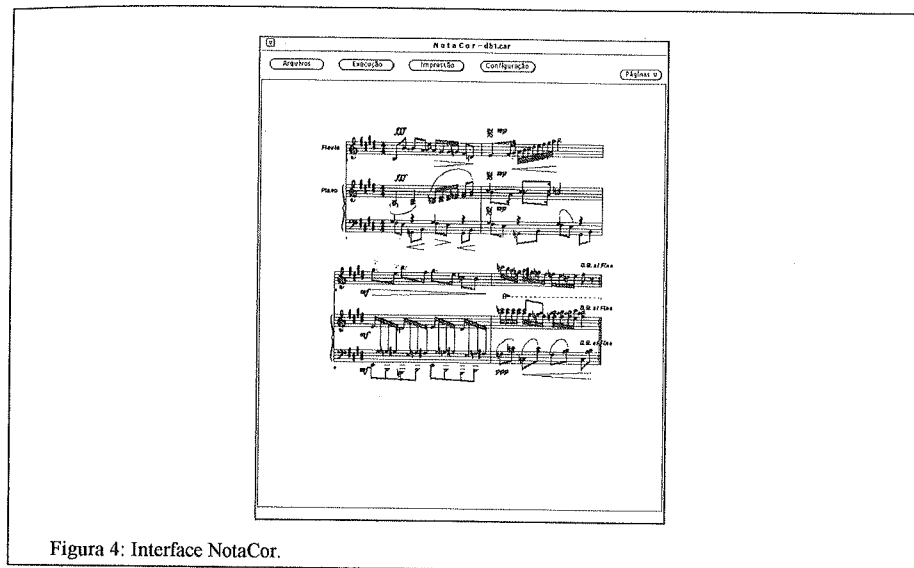


Figura 4: Interface NotaCor.

## Referências

- Adobe Systems, *PostScript Language Reference Manual*, Addison-Wesley, USA, 1990.
- Ames, Charles, *Automated Composition in Retrospect: 1956-1986*, Journal of the International Society for the Arts, Sciences and Technology, USA, 1987.
- Arcela, Aluizio, *Síntese de Imagens com Pedacos de Tempo*, Publicação Interna, LPE/CIC/UnB, Brasília, 1990.
- Beauchamp, James, e outros, *Music by Computers*, John Wiley and Sons Inc, New York, 1969.
- Gioia, Osman G., *Orquestrador MIDI Sinfônico*, na corrente publicação.
- Gourlay, John S., *A Language for Music Printing*, Communications of the ACM, vol. 29, #5, pp 388-401, USA, maio 1986.
- Olson, Harry F., *Music, Physics and Engineering*, Dover Publications, Inc, New York, 1967.
- Producao Interna ao Laboratório de Processamento Espectral do Departamento de Ciência da Computação, UnB, 1989-1993.
- Smith, Leland, *Editing and Printing - Music by Computers*, Journal of Music Theory, vol. 17, vol. 2, pp 292-307, USA, 1973.

## Representação Angular para Notação Musical

EDILSON EULALIO CABRAL

Departamento de Artes

CH - UFPB

Av. Aprígio Veloso 882 - 58109-970

Campina Grande - Paraíba - Brasil

E-mail : eulalio@dec.ufpb.br

Telefone: 333-1000 R:135

## Resumo

A proposta deste trabalho é mostrar uma forma de representação musical cujo objetivo seja facilitar a percepção visual dos intervalos musicais. Isso nos levará além, uma vez que iremos entender de modo diferente a forma como devemos pensar as notas musicais, pois passaremos a raciocinar em termos de ângulos, aberturas e distanciamentos entre elas, etc...

Para cada nota será atribuído apenas um símbolo, inclusive para as enarmônicas, como por exemplo o dó<sub>2</sub> e o ré.

Enquanto na notação convencional utiliza-se, para o Piano, duas claves (a de Sol e a de Fá) e para o Violão somente a clave de Sol, neste sistema a representação para qualquer instrumento poderá ser feita de uma única maneira.

## 1. Introdução

Há várias maneiras de se representar as notas musicais usando-se pentagrama, cifra, tablatura, o nome (dó, ré, etc.), etc... Geralmente adotamos a que mais satisfaz aos nossos objetivos, ou seja, quem deseja tocar um instrumento apenas para se acompanhar, pode achar ser suficiente apenas aprender os acordes através de seu desenho numa tablatura em vez de aprender sua codificação na forma de cifras. Qualquer sistema de notação é deficiente para representar inteiramente todas as formas de expressão musical. Isso porque quanto mais elementos forem utilizados, maior número de símbolos serão necessários para representar o que se deseja tornando-se, muitas vezes, impraticável e confusa sua aplicação de forma integral.

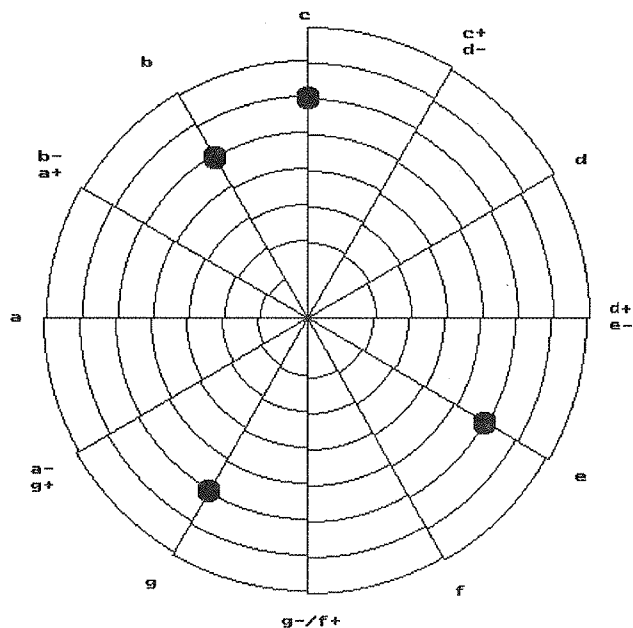
Cada sistema de notação tem, portanto, melhor aplicabilidade para determinados casos. No nosso, o sistema que passamos a descrever se presta melhor para a percepção visual de intervalos musicais o que nos ajudará bastante em termos de análise harmônica e melódica.

## 2. O Sistema

Como sabemos, as notas musicais utilizadas comumente no sistema musical do Ocidente (doze semitons, escala temperada), guardam uma relação que pode ser expressa apropriadamente de forma geométrica (Só, 1961).

A Representação Angular para Notação Musical é uma maneira de representar as notas musicais de tal modo que cada nota seja escrita em um determinado ângulo onde o espaçamento de 30 graus, entre as linhas, representa uma relação de meio tom.

Basicamente este sistema é representado por uma espiral cortada por linhas, onde pequenos círculos, representativos das notas, serão colocados em locais determinados pelas intercepções das linhas com a espiral. A altura das notas são determinadas pela proximidade ou afastamento destes pontos em relação ao centro da espiral onde as mais agudas são aquelas mais próximas daquele centro. Ver ilustração seguinte:



### 3. O Software

Sua função é a demonstração deste sistema com a utilização de uma interface gráfica que poderia ser utilizada para vários outros tipos de programa. Com esta interface podemos exibir simultânea e integralmente, inclusive em tempo real, todas as vozes possíveis em um mesmo canal e, dependendo de uma melhor implementação, também em vários canais.

Aplicando este sistema, o software desenvolvido consta de duas partes: A parte de Edição e a parte de Execução de arquivos MIDI onde apenas o formato padrão será utilizado.

No Editor utilizamos as figuras das notas representadas conforme a seguinte ilustração:

- ⇒ Semibreve
- ⊙ ⇒ Mínima
- ◐ ⇒ Semínima
- ◑ ⇒ Colcheia
- ◒ ⇒ Semicolcheia
- ◓ ⇒ Fusa
- ◔ ⇒ Semifusa

Na parte de Execução não utilizamos as figuras das notas representadas na parte de Edição, apenas pequenos círculos simples :

Exemplo: ○

Obs. - A utilização de figuras com um formato mais complexo, como no Editor, só se justifica caso o computador tenha velocidade suficiente para exibi-las de forma integral e sem interferir no desempenho do programa.

### 4. Conclusão

Pelo exposto, pode-se dizer que isso é apenas o começo para o desenvolvimento de um sistema maior que abrangeria de forma mais completa o fenômeno musical (incluindo a representação gráfica da dinâmica, timbres, etc.).

### Referências

Hugo de Andrade Só (1961). *Ciência e Música - Física dos sons musicais*.

POSITIONAL RHYTHMIC NOTATION: AN IMPLICATION FOR A POSITIONAL  
THEORY OF RHYTHM.

MORAES, M. R.

Departamento de Formação Artística  
Universidade Federal do Espírito Santo  
Av. Fernando Ferrari S/N,  
Vitória, ES - CEP 29060 - Brasil  
E-mail: mmoraes@npdl.ufes.br

Abstract

Computer modelling of music, and for that matter any applications in music, reflects a musical theory, which is in itself a *model*. Thus, if there is any inadequacy in the theory, a computer model that successfully embodies that theory will also include that inadequacy. One of such inadequacies is the assumption that duration is a concept central to rhythm. We claim, instead, that onset positions play a central role. Among the many potential consequences of such a claim we maintain that a purely rhythmic notation must reflect different positions within a tree hierarchy, rather than durations. Bearing this in mind, we devised a notation such that only either presence or *absence* of the onset of a sound on a specific locus are represented. Some issues related to the way a user thinks and acts when inputting musical data into the computer *vis-à-vis* positional notation, and the positional *concept* itself, are discussed. The impact of positional notation (and the positional *concept*) on music teaching begs special attention.

Brief Remarks on Interdisciplinarity

The interdisciplinary nature of Computer Music - computer modelling being one of its sub-areas - is one of its many appealing aspects. Moreover, interdisciplinarity *per se* is often, and most of the times rightfully, welcome in the academic world. Nevertheless, with the proviso that we strongly support interdisciplinary work, we will point out some problematic issues related to interdisciplinarity which constitute the more general and, in a way, more theoretical concerns that motivate the specific questions to be addressed below.

The first precaution one must take in interdisciplinary work concerns terminology. Technical terms, as we know, must have their meanings as precisely defined as possible. Assuming, on the one hand, that this is not a major problem (and sometimes it *is*) within one single, self-contained field of study, on the other hand, not few problematic situations arise when it comes to relating two or more disciplines, each of them built upon its own set of fundamental concepts, its own premisses and axioms. This situation requires work akin to that of a translation which will establish that one concept is attached to the term 'x' belonging to a given discipline 'A' and that the same (or approximate) concept is represented by 'y', that belongs to another discipline 'B'. An interesting, and potentially dangerous, situation occurs when 'x' (the *signifier*, as borrowed from semiotics) can be found both in 'A' and in 'B'. More often than we would like to admit, we tend to assign the same meaning to 'x' in 'A' and 'x' in 'B', and this, if their correct meanings are not the same, and if the concepts are of fundamental importance to the theories they belong to, can have disastrous consequences. (Of course we are much more aware of the same sort of situation when two different natural languages are involved). This problem, however, is mentioned here as a subsidiary issue as regards our main subject (see Grillmer 1986).

The other precaution we must take refers to the status of the disciplines or the theories involved. True interdisciplinarity must be established between theories bearing the same status. Most fields of study, with their corresponding theories, have a systematic character, due to rigorous and continuing investigation. These could be called true theories, theories *stricto sensu*, or simply theories. There are, however, some cases in which a collection of terms and rules related to a certain area of knowledge and activity is unduly known either as a

theory or as a science. Typically, this situation occurs along with a process we might call *linguistic naturalization*, i.e., those constitutive terms and concepts, which could once have been strictly defined and interrelated, go through a loss in precision proportional to their ever increasing usage as everyday natural language. But, in spite of such *naturalization* - due among other causes to a lack of continuing investigation - tradition keeps ascribing this 'theory' the status of Theory. Hence, if there happens that a systematic discipline (i.e. theory proper) gets involved with a discipline of this second type (i.e. a pseudo-theory), the resulting interdisciplinary field is bound to lose some of its consistency. In other words, even when we are attentive enough to method and logic in our procedures, if we inadvertently accept the pseudo-theory as an adequate rendering or *model* of the object or problem under investigation, we might not only weaken our interdisciplinary field (from a formal point of view), but also bring into it some descriptive inadequacy, i.e., we may *distort the image of the object*, since the 'natural truths' embodied by the terms of a pseudo-theory tend to hide their lack of strict denotative relation to whatever referent in the phenomenal world (see Moraes 1991).

#### Music Theory

Music theory is a typical instance of what we have called pseudo-theory. Many authors, in particular those engaged in interdisciplinary work, corroborate this assertion. Lindblom (1976) states that "*traditionally, music theory works with impressionistic, non formalizing methods*". Hackman (1975) says: "*it took far too long for me to realize that the methods of music analysis had to bear at least a superficial resemblance to other methods of scholarly and scientific inquiry*". We shall also mention Babbit (1975) (quoted by Hackman): "*if scientific method is not extensible to music theory, then music theory is not theory in any sense of the word*", and Jackendoff & Lerdahl (1983): "*It (music theory) severs questions of art from deeper rational inquiry; it treats music as though it had nothing to do with other aspect of the world*".

Of course, much has been done in the last decade in the way of filling this gap. Work done by psychologists, linguists, computer scientists, and others doing research in musical cognition are all decisive contributions towards a systematic theory of music. Yet, there is a shady area that remains apart from mainstream spotlights. This regards those very elementary concepts related to music, i.e., *not* those related to larger structures but those 'natural truths', those fundamental terms that, once *naturalized*, are used as universal premisses upon which theories and models are built. As to rhythm, which is our central concern here, this situation might well be illustrated by Martin (1972), to whom "*rhythm appears to be taken so much for granted in music training that there is only one book on rhythm theory although there are many on melody, harmony and counterpoint*". Martin is not very accurate as to the number of books he mentions, but we would argue that the situation has not changed essentially (i.e. rhythm taken for granted=*naturalized*) since then.

#### Computer Modelling of Music: is it interdisciplinary work?

At first glance we could admit that computer modelling of music - let us take it as an applied branch of the computer sciences - does not correspond to a strict notion of interdisciplinary work, since this modelling would represent a relation between a discipline (computer science) and its objects (musical phenomena in this case). However, one could argue that computer modelling of music should be considered as a branch of the musical discipline in which the computer (considered not only as hardware but as a complex notion including related theories and methods) would have the status of a privileged tool. For our part, as a music teacher, we could choose to support this last view, but we are obviously far from having computer modelling of music and systematic music theory (which is, in a way, a *model* of music as far as it is Theory) as one and the same discipline in which the conceptual model and its physical counterpart would be complementary aspects of the same *inquiring* process (explanatory, not only descriptive).

Furthering this discussion is a task that is obviously beyond the scope of this paper. For the sake of our interests, we will only add that, at present, computer modelling of music should be understood as interdisciplinary work. In modelling music, one is not modelling *music itself* but rather relying on much knowledge about music that is *taken for granted*. This means that what we have is not a simple relation between a discipline or technique and its object but a potential relation involving two disciplines. Provided that due attention is given to unresolved and problematical issues still belonging to a 'pure', independent music theory, we will have true interdisciplinary cooperation. Computer (or programming) courses within music curricula are still exceptions rather than rule, as should be the case, and music courses - maybe theoretical courses mainly - in computer curricula would, of course, enhance interdisciplinarity.

#### Rhythm: a problem

No one would deny the fundamental importance of rhythm to music. Many would agree that music is *par excellence* the art of time and rhythm, and that this idea has more heuristic power than the truism *music is the art of sound*. Nonetheless, we should also agree that rhythm is a very elusive subject. Linguists (see Benguerel & D'Arcy 1986) would say: "*it is already obvious that a detailed account of language will require a lot more knowledge about rhythm*", (but) "*rhythm is very difficult to define satisfactorily*". Addressing a similar situation, Willems (1954) tells us that back in 1738 "*Matthenson reconnaissait l'importance de la théorie du rythme mais la regardait 'une science confuse'*". Meschonic (1982) quotes Paul Valéry: "*ce mot 'rythme' n'est pas clair: Je ne l'emploie jamais*".

However, in spite of that elusiveness, it seems that music theory (the elementary *naturalized* 'theory') has some sort of answer to all that: just pick up a series of proportional durations (most of them materialized as sound) and put rhythm within our rational reach. Better still: look at those simple arithmetic relations made visible by quarter and eight notes or even by their x.y rendering like in a *piano roll window* of a MIDI sequencer. No more mystery. Durations! that is the stuff rhythm is made of.

Contrary to this, Piaget (1946) would conclude that duration (pure duration) could be "*but a myth*", or at the most, a concept that is not a primitive (fundamental) one but a result of previous operations based much more on topologies than on any kind of linear measurement. On the musical side, we could agree with Piaget by saying that we cannot directly assess duration in a categorical way (like in: duration of note a equals 0.25 of note b's duration). Bachelard (1933) would say that "*in music, a note's duration is not one of those pure elements, clearly primitive, as sight-singing teachers would make us believe*".

If we discard duration as a concept central to rhythm (as common music theory and notation would make us believe), we must have some other concept in its place. We will claim that this key concept is *position* as proposed, among others, by Martin (ibid.) and by Howard and Perkins (1974). After having stated that rhythm cannot be viewed only as a linear concatenation of segments, Martin, whose article involves both music and speech rhythm, states that "*temporal patterning would refer to the onset of each musical note or syllabic vowel*"... and that a certain rhythmic rule "*applies not to syllable duration but to syllable loci, specifically their vowel onsets*". From a specifically musical perspective, Howard and Perkins define *impulse* as "*certain but not all perceived discontinuities, abrupt changes in the ongoing auditory stimulus*" (...)"we follow Allete (1951) in considering such auditory events as central to rhythm, in contrast with durations of notes, for instance". They will also add that "*an impulse is 'at' a point in time and not at other neighboring point*".

#### Positional Notation.

Notation is a very economic, yet powerful, encoding tool. Unlike words, i.e., the linguistic-discursive apparatus, a set of graphic signs like that of music notation bears no symbolic-arbitrary relation to the thing it represents, but, to a certain degree, it has an iconic relation to the thing it represents. Thus, we expect to see reflected in music notation every important property of the thing represented. Sometimes, specially in music notation, that relation may (unconsciously) be perceived as an indexical relation (in the semiotic sense), and written notes become, as it were, a symptom of the thing represented, if not the thing itself! (whereas no one has ever tried to bite the word "apple").

If we now go from musical notation back to music, we would be tempted to admit that if music notation *has* (represents) durations, then music (rhythm) has duration as one of its important properties. This is one of the mechanisms (of sophistic logic) that, by virtue of the subliminal convincing power of notation, make us believe that duration is the stuff rhythm is made of. However, if - considering what has been claimed above - we seriously reconsider our premisses regarding the important properties or, the relevant structural and perceptual properties of rhythm, we can, again, go from music to notation, with the result that the notation must, in some way, reflect the new premiss. One of such results is *positional notation*.

Positional Notation is supposed to be a purely rhythmic notation (it does not allow for pitch representation) and is not intended as a universal substitute for conventional notation. It is not a descriptive or analytical notation but rather an extremely economic and synthetic tool, both graphically and conceptually. As we have been testing it in several situations (teaching f.ex.) since 1980, we claim that positional notation bears a closer relation to musical-rhythmic perception and cognition (as compared to 'durational' notation).

We start from the idea that rhythmic pulse-meter is structured as a topology similar to that of verbal phrase syntax (see Hackman-1975, Martin-1972, Jackendoff-1977, Jackendoff & Lerdahl 1983). Unlike verbal syntax,





## A Visual Programming Environment for Constraint based Musical Composition

CAMILORUEDA  
*Ingeniería de Sistemas, Universidad Javeriana de Cali*  
*Cali, A.A 26239 Colombia*

### ABSTRACT

We describe a visual programming environment called *Niobé* in which the composer can easily construct and operate on *template* musical structures defined by a set of relations. *Niobé* provides primitives allowing the composer to graphically program arbitrary constraints on some musical domain (harmonic, rhythmic, etc) and also a mechanism for computing one or several instances of specific musical structures whose elements satisfy the given constraints. The composer can in this way construct a potentially large data base of different musical structures, each having the same precisely defined properties. Resulting structures can be visualized and hand modified in different supplied music notation editors. *Niobé* has been carefully optimized for computing sequences of harmonic or rhythmic elements. It uses the graphical interface of *PatchWork*, the visual music composition language developed at IRCAM in Paris. *Niobé* is implemented in Common Lisp-CLOS and is extensible.

### I. Introduction

We present *Niobé*, a graphical environment for rule based music composition. *Niobé* is well suited to the incremental construction of musical structures obeying precisely defined properties. Structures are built in *Niobé* either by setting control parameters of built-in relations or by imposing a set of new user defined constraints. This way of regarding computation falls within the realm of what is called *Constraint Satisfaction*. *Niobé* is logically divided in two components: A computational engine adapted for solving constraint satisfaction problems in the musical domain and a graphical programming interface. We show how the composer can take advantage of the interaction between these two components to interactively construct and refine harmonic or rhythmic structures in an incremental process. That is, instances of roughly specified structures computed by the system can be visually represented in suitable music notation editors. The composer might then see the need to impose further melodic or harmonic constraints which are used by *niobé* to compute new refined instances. *Niobé* runs on top of *PatchWork* (Laurson, Duthen & Rueda, 1992), a graphical music composition language adapted to the representation of precesses as a sequence of functional transformations. We describe how to exploit in *niobé* the two alternative ways of regarding programming, constraint-oriented and functional, to effectively compute complex musical structures. *Niobé* is entirely programmed in Common LISP-CLOS (Steele, 1990). Finally, we present some examples of the usage of *Niobé* in real musical applications.

### II. Background

Several music composition languages such as Pla (Schottstaedt, 1983) have been defined in the past. These languages consider the compositional activity as divided in two basically independent processes, a score defining scheduling of different types of events and a functional part where structures are built by composition of transformations. In the latter, the composer is responsible for programming the appropriate transformations leading to the desired structure. This activity requires in general good programming skills. To alleviate somewhat the burden of the interaction of the composer with the computer, these systems provide a library of predefined transformations the composer can use as basic building blocks. In Common Music (taube, 1991), this idea is complemented with a set of built-in pattern structures that the user regards as templates for instantiating the desired structure. Though very powerful, this schemes leaves the problem of having to decide appropriate ways of combining patterns and functions. In a broad sense, all of these composition languages reflect the functional paradigm of the underlying implementation language in which it is necessary to describe a desired result by making explicit the way of achieving it. There are situations, however, in which it is not at all obvious to find appropriate algorithms to compute structures that nevertheless admit simple descriptions. Recently, the composition environment PatchWork (Laurson, Duthen & Rueda, 1992) has been proposed as a way of easing the programming task by redefining it as a visual activity. A program in this language is a graphical patch where boxes represent computations and links between boxes define functional composition of transformations (see figure 1). As in the above mentioned languages, a library of predefined boxes provide building blocks for programming. Although we believe that giving composers an entirely visual programming environment is a step in the right direction, this might not reduce significantly the programming effort in situations where the actual musical processes are not conceived algorithmically (i.e as a sequence of transformations) at the beginning but rather as evolving sets of precisely defined relations. What is needed in this case is the capability to propose *descriptions* of structures in a declarative manner, leaving to the system the task of actually computing them.

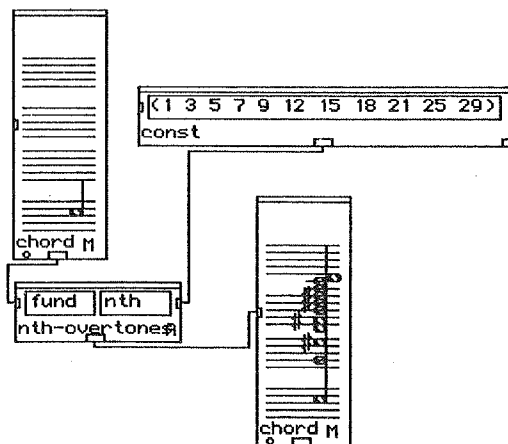


Figure 1. A PatchWork patch computing a chord from a set of harmonics of a base note

Recently, a system based on this idea, called *Echidna* (Ovans, 1990), has been proposed to support counterpoint generation. *Echidna* is not in itself a music composition language but rather a general purpose declarative programming tool based on the principle of constraint satisfaction. The rules of counterpoint are first stated as a set of constraints on (finite) values

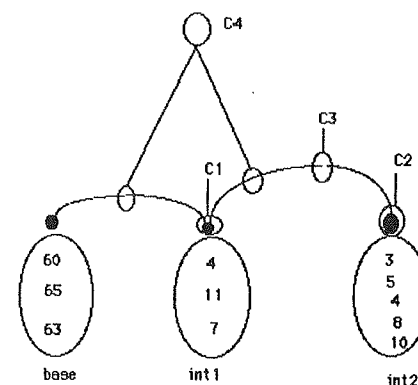
representing pitches and then *Echidna* is used to choose subsets of these satisfying the given harmonic and melodic constraints.

The two approaches, functional and declarative, are complementary in the sense that for certain type of material one or the other proves to be more convenient. Our research concern has been to unify both ways of conceiving the process of constructing musical structures. The underlying notion sustaining this aim is that of a *partially instantiated musical structure* (PIMS). Loosely speaking a PIMS (Assayag & Rueda, 1993) is a generalization of a structure in the functional sense whose elements are sets and augmented with a collection of relations or constraints. In what follows we precise this notion and describe its implementation in *Niobé*.

### III. The theory of PIMS

A PIMS is the basic building block for generating musical material. It is defined as the structure  $\langle D, R, C \rangle$  where  $D$  is a finite collection of finite sets (called *Domains*),  $R$  is a binary relation on  $D$  and  $C$  is a set of constraints (relations) on  $D$ . Basically,  $R$  is a *structuring* relation on  $D$  whereas elements in  $C$  are *filtering* relations on elements of  $D$ . Constraints in  $C$  define subsets of the cartesian product of the sets in some subset of  $D$ . Any element in the cartesian product defined by a constraint  $c$  of  $C$  is said to *satisfy*  $c$ . If all constraints in  $C$  define non empty sets the PIMS is said to be (locally) *consistent*. A PIMS in which  $D$  contains only singleton sets is called a PIMS *instacnce*. A PIMS *exemplary* is a consistent PIMS instance.

A Partial order can be defined on PIMS as follows: Let  $P = \langle D1, R, C \rangle$  and  $Q = \langle D2, R, C \rangle$  be PIMS. If each cartesian product on  $D1$  is contained in some product on  $D2$  then  $P \leq Q$ . Given a PIMS  $P$ , the *PIMS instantiation problem* consists in finding a PIMS exemplary  $E$  such that  $E \leq P$ . Seen from this perspective a PIMS is a structure scheme representing the set of its exemplary structures. The graph in the figure below represents a PIMS for the set of all three note chords starting at any one of the notes in the set *base* (in MIDI), having consecutive intervals taken from the sets *int1*, *int2* (in semitones), not containing octaves and positioned within the register from 60 to 79 in MIDI.



C1:  $int1 \neq 12$ ; C2:  $int2 \neq 12$   
 C3:  $int1 + int2 \neq 12$ ; C4:  $base + int1 + int2 \leq 79$   
 R:  $base \rightarrow int1 \rightarrow int2$ ; D:  $\{base, int1, int2\}$   
 Figure 2. A PIMS

#### 3.1 Structure instantiation by arc consistency.



Building musical material can thus very generally be seen as a two step process, first constructing a suitable PIMS and then solving the PIMS instantiation problem on it. For the latter we use in *Niobé* arc consistency techniques. These are well known algorithms in the constraint satisfaction field aiming at improving the efficiency of finding a solution by trying to reduce the given domains. Domains are reduced by insuring that constraints are locally consistent. A constraint can be represented by a graph (see figure 2) having domains as nodes and constraints as arcs linking those domains it constrains. An arc in this graph is said to be consistent if for any element in any of the linked domains there can always be found elements in the other linked domains such that all taken together satisfy the constraint. Values in the linked domains not obeying this property can be eliminated, thus reducing domain sizes. Algorithms for achieving arc consistency are described in (Mackworth, 1977). Recently, a more efficient arc consistency procedure called AC-5 has been proposed in (Deville & Van Hentenryck, 1991). AC-5 runs in time proportional to the square of the biggest domain size, but can easily be specialized to a linear time algorithm for useful categories of constraints. These are referred to in (Deville & Van Hentenryck, 1991) as functional and monotonic constraints. Briefly stated, these are constraints such that suitable representatives in each domain suffice to test the validity of the constraint for the whole domain. Constraint C4 in figure 2 above is of this type. AC-5 forms the core of the structure instantiation scheme in *Niobé*. Additional optimizations are considered by defining hierarchies on the PIMS domains reflecting frequently encountered musical constraints. One example is the problem of instantiating chord sequences structures where constraints imposing particular melodic movements on the upper and lower voices are frequently stated. Chord structures (PIMS) are thus supplied with an additional (hidden) domain comprising possible sums of consecutive intervals in the chord. Melodic constraints can thus be (automatically) redefined to act on the base note and sum-of-intervals domains avoiding the need to look into each particular interval composition of a chord. Domains in *Niobé* are thus *trees* allowing constraint impositions at any level. Figure 3 below shows an example this tree for a chord intervals domain. A similar structuring technique has been proposed in *Echidna* (Sidebotton & Havens, 1991) for representing constraints on real numbers.

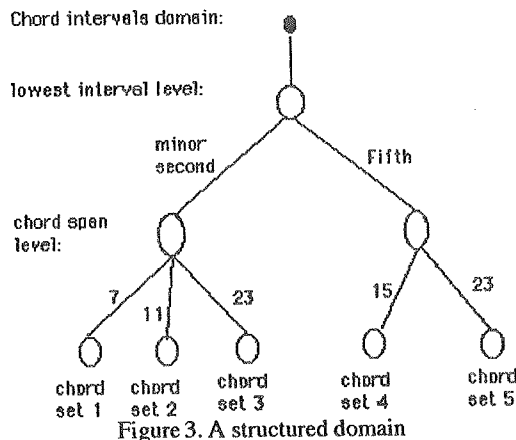


Figure 3. A structured domain

Musical constraints within a PIMS are in general conceived by the composer as having different degrees of importance. We describe next a mechanism implemented in *Niobé* for taking account of this fact.

3.2 Soft constraints in PIMS.

Constraints in a PIMS can be assigned a degree of importance. In *Niobé* this is simply a number between zero (useless constraint) and one (required constraint). A valuation function is defined on PIMS instances. The value of an instance P is equal to one minus the importance degree of the most important unsatisfied constraint c in P. *Niobé* computes highest valued instances of a PIMS by extending AC-5 with a process similar to the standard *alpha-beta* procedure used in several Artificial Intelligence applications such as game tree search. Formally, The PIMS instantiation problem is redefined as follows: Given a PIMS P, find a PIMS instance  $Q \leq P$  such that, for any other PIMS instance  $R \leq P$ ,  $Valuation(R) \leq Valuation(Q)$ . A detailed account of this way of handling constraint preferences (or *soft* constraints) can be found in (Schiex, 1992). In *Niobé*, both the degree of importance of constraints and the minimum value required of a solution are user controlled parameters. We develop next the user interface of *Niobé* looking into the details of some examples.

3.3 Using *Niobé*.

*Niobé* is implemented in Common Lisp-CLOS. A PIMS, its domains and constraints are CLOS objects. A graphical interface in *PatchWork* is supplied for constructing and parameterizing these objects and for triggering the instantiation mechanism. PIMS instances can also be graphically interpreted, functionally transformed and displayed/edited in standard music notation by using suitable *PatchWork* editors. A *PatchWork* box (called *harmonic-constraints* in figure 4) representing *Niobé* defines entries for domains and constraints specifications. These entries consist of a set of parameters controlling *templates* of built-in constraints and domains. In the example of figure 4, domains are sets of chords and the PIMS to be computed is simply a sequence of chords obeying precise vertical (harmonic) and horizontal (melodic) constraints. The entry called *ambitus*

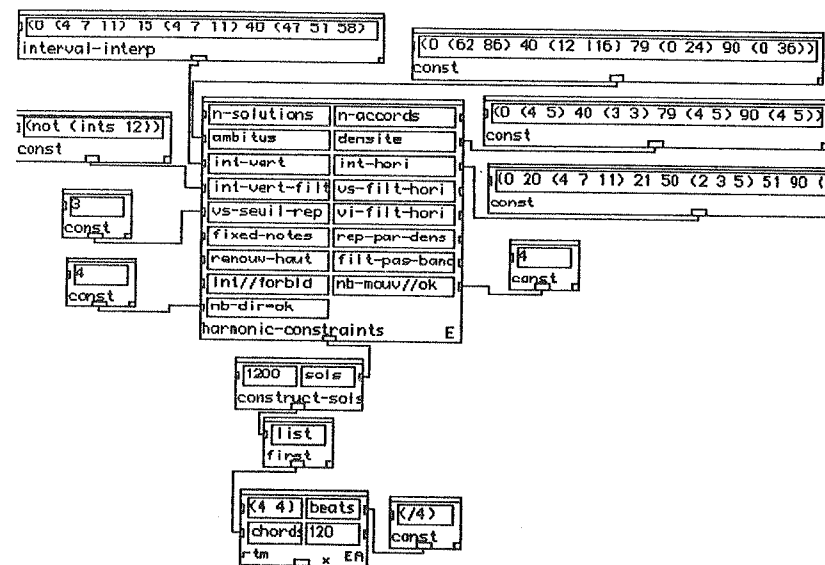


Figure 4. The graphical interface of *Niobé*.

is connected to a *const* box defining the global vertical span of the sequence. It covers two octaves (62 to 86 in MIDI) in the first chord then goes to about 9 octaves (12 to 116) for the 40th chord and comes down to 3 octaves at the last chord (0 to 36). The span given for selected chord numbers represent points of a linear interpolation computing the global span for the rest of the chords. This thus gives the general form of the region where the chord sequence should fit. The box connected to the entry *int-vert* (vertical intervals) defines the domain of intervals (taken between consecutive notes) for the chords. For the first 40 chords only major third (4 in semitones), fifth (7 in semitones) and/or major seventh (11) are allowed. In the middle of the sequence very long intervals are demanded. The intended effect is to shift perception from harmonic to melodic. At the end of the sequence the original intervals return. Entry *int-vert-fill* defines any filtering relation on the interval contents of each chord. Here only the elimination of octaves is imposed (by the predicate (*not (ints 12)*)). *Densite* is just the number of allowed notes in each chord. Here either 4 or 5 notes (as *Niobé* likes it) in each chord is established for all but the middle of the sequence where exactly 3 note chords are demanded, the reason being to precisely control melodic movement of each of the three voices in this part of the sequence. Melodic control is done at the *int-hori* (horizontal intervals) entry of the box. Here the same intervals of major third, fifth and major seventh (4, 7, 11) are imposed *horizontally* for the upper and lower voices in most of the sequence, except in the middle where intervals of a second, minor third and fourth (3, 5, 8) are actually required horizontally for each one of the three voices. Although not shown in this example, melodic movement can be further controlled by drawing break-point functions defining curves to be followed by the highest point of each chord. Other entries (not used in the example) can define parallel or contrary movement of voices (entry *nb-mouv/ok*), chord intervals contents as a function of register (*filt-pas-band*) or lower bounds on the total number of different notes in arbitrary sections of the sequence (*renouv-haut*). Figure 5 shows part of a sequence computed by *Niobé* from the constraints specification of figure 4. The rhythm has been

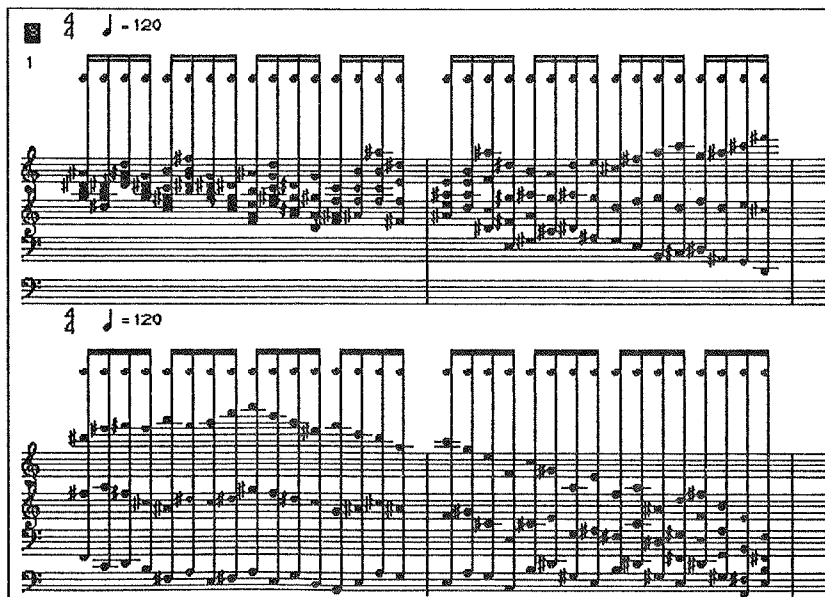


Figure 5.

arbitrarily set (in one of the entries to the *rim* box) to eighth notes. The above example solves a chord sequence generation problem by a simple adjustment of parameters of built-in harmonic and melodic constraints. This is only one way of proceeding. The composer can draw any patch computing an arbitrary constraint and then connect it to one of the entries of the *harmonic-constraints* box. *Niobé* will then compute instances of the sequence taking into account the new constraint. Figure 6 shows one example, used by the Italian composer Marco Stroppa to compute a sequence of chords following a given progression of *homogeneity*, defined as the difference between the biggest and smallest chord intervals. The patch to the left of the figure takes in the *parameter* box the current chord proposed by *Niobé* and also its position in the sequence. The patch computes the homogeneity of the chord and tests it against the acceptable range desired for a chord in that position of the sequence (see box *interval-interp*), giving TRUE (acceptable chord) or NIL (non-acceptable) accordingly.

IV. Conclusions.

We have described *Niobé*, a music composition system integrating the relational and functional conceptions of programming through the use of a visual programming environment. *Niobé* has successfully been used by several composers to compute harmonic and rhythmic sequences. A refined

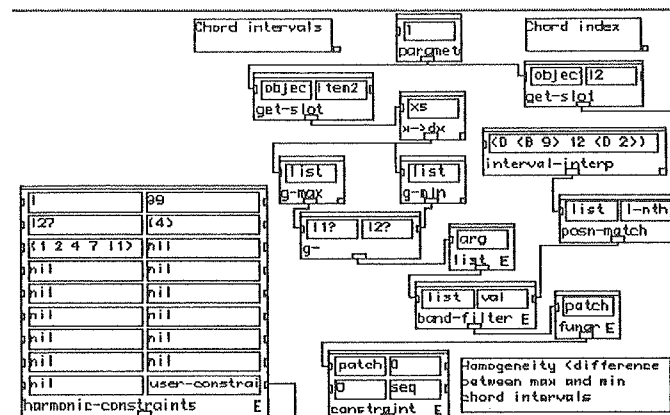


Figure 6.

version of the example in figure 4 was used by the french composer Antoine Bonnet in his piece *Építaphe*. The german composer Michael Jarrell has used *Niobé* to generate melodic sequences for his piece *Rhizomes -IV*. We are currently studying more uniform models of integrating the functional (and object-oriented) and relational aspects in *Niobé*. Although *Niobé* has proved to be reasonably efficient (a few minutes in a Mac Quadra 700 for computing sequences of less than a hundred chords) in several practical situations, we have accomplished that at the price of restricting the allowed types of *horizontal* constraints. We are exploring different optimizations of the base algorithm to allow the specification of more global vertical or horizontal relations.

References

Assayag, G. & Rueda, C. (1993). The Music Representation Project at IRCAM. *Proceedings of the*

ICMC . Tokyo, 1993.

Deville, Y. & Van Hentenryck, P. (1991). An Efficient Arc Consistency Algorithm for a Class of CSP

Problems. *Proceedings of the IJCAI* . Sydney, 1991.

Lurson, M. Duthen, J & Rueda, C. (1992). The PatchWork Reference Manual. *IRCAM* , 1992.

Mackworth, A. (1977). Consistency in Networks of Relations. *Artificial Intelligence* . 99-118.

Ovans, R. (1990). Music Composition as a Constraint Satisfaction Problem. *Proceedings of the ICMC* .

Schiex, T. (1992). Possibilistic Constraint Satisfaction Problems or "How to Handle Soft Constraints".

*CERT-ONERA* . Personal e-mail communication.

Schottstaedt, B. (1983). Pla: A Composer's Idea of a Language. *Computer Music Journal* . 7(1).

Sidebottom, S & Havens, W. (1991). Hierarchical Arc Consistency Applied to Numeric Processing in

Constraint Logic Programming. *CSS-IS TR 91-06* . Simon Fraser University, Burnaby, Canada.

Steele, G. (1990). *Common Lisp: The Language* . Digital Press.

Taube, H. (1991). Common Music: A Music Composition Language in Common Lisp and CLOS.

*Computer Music Journal* . 15(2).

## Incremental evaluation in a musical hierarchy

M. DESAINTE-CATHERINE

K. BARBAR and A. BEURIVÉ

*LaBRI*<sup>1</sup>

*Université Bordeaux I*

*351, cours de la Libération*

*33405, Talence Cedex*

*France*

*myriam@labri.u-bordeaux.fr*

### Abstract

The work we present in this paper is a formalism of a dynamic computational model in a hierarchy. We are interested in representing musical hierarchies and bindings of characteristics (such as the mode, measure, tempo, duration, key, etc.) within them in order to provide the composer a means to verify the consistency of the piece during the compositional process. The model transfers any modification from the composer to the representation in an incremental way, without computing again the whole hierarchy.

## 1 Introduction

The complexity of a musical piece can be organized in a hierarchical way based on its temporal structure. Musical characteristics (such as the mode, measure, tempo, duration, key, etc.) can be defined at any point of the hierarchy (that is any sub-piece). These characteristics are then bound together according to the temporal structure and the musical rules imposed by the composer. We are interested in representing musical hierarchies and bindings of characteristics within them in order to provide the composer a means to verify the consistency of the piece during the compositional process.

Our work may be situated between constraints propagation techniques and hierarchical representations à la Balaban. We are interested in designing the representation and the computation model which is appropriate to it. From our point of view, a musical piece is an object that is composed of several dimensions. Classic dimensions are time, frequency, timbre and volume. The variations of the values in these dimensions are not independent from each other. The result of a musical analysis is exactly a set of correlations between variations within a single dimension and between different dimensions. In order to formalize those correlations, we define several relational operators which are dedicated to specific dimensions. The set of values in each dimension can then be structured in a hierarchical way using these operators. Hierarchical way means that the object representing the structure is not always a simple tree, but a directed acyclic graph (see the notions of shared occurrences and repetitions of Mira Balaban (Balaban 1993)). The originality of this work relative to the others based on hierarchical representations is the addition of a semantics to the hierarchy. This semantics provides a very sound way to represent

<sup>1</sup>Laboratoire Bordelais de Recherche en Informatique - Unité de Recherche Associée au Centre National de la Recherche

some correlations in order to verify or apply them. The obvious limit of this representation is that it does not compute the correlations that are not based on an operational structure. We believe, however, that there almost always exists a structure underlying every kind of correlations.

Our initial study concerns the time dimension. Time relational operators have been widely studied (in particular, the musical concatenation of Mira Balaban (Balaban 1991), relations of Allen (Allen 1983) and their application to music by Alan Marsden (Marsden 1994)). The two operators *concatenation* and *superimposition* provide a simple model with an acceptable power of expression. We first investigated the static aspect of the model (see (Barbar, Desainte-Catherine, Miniussi 1993) and (Barbar, Desainte-Catherine 1992)) in the following way. We first transform a musical equational program defining the structure of a musical piece into a derivation tree according to an attribute grammar. This derivation tree is then considered as a data structure which represents the musical hierarchy. Each attribute in the derivation tree represents a musical characteristic and the associated semantics represents the musical rules binding these characteristics. The evaluation step computes a solution (the values of all characteristics of the hierarchy), if it exists.

This previous work provides a very sound model but is insufficient in the context of an interactive compositional environment. A dynamic model is needed. This model must transfer any modification from the composer to the representation in an incremental way, without computing again the whole hierarchy.

The work we present in this paper is a formalism of a dynamic computational model in a hierarchy. The data representation is the same than the previous one. Only the computational model has changed. This model manages modifications (giving a value to a characteristic, changing a value of a characteristic, modifying the hierarchy itself by substituting one sub-piece by another) and maintain the overall consistency of the piece. The first two operations necessitate the propagation of the modifications of a characteristic in any direction in the hierarchy. The last operation implies the management of several hierarchies at the same time. Our formalism is no longer based on attribute grammars, but on systems of equations.

In section 2, we present the syntactic aspect of a musical hierarchy. It is represented by an equational program which is given with a set of syntactic equations. We give in section 3 the musical systems or the relations between characteristics attached to nodes of a hierarchy in terms of sets of equations on these characteristics. We define the solution of a musical system in section 4. An incremental strategy for the determination of the solution is given in section 5. The section 6 contains our conclusion.

## 2 Equational Program

### 2.1 Temporal Operators

Let us denote by  $(t, d, f, s, v)$  an event, where  $t$  is the beginning time,  $d$  the duration,  $f$  the pitch,  $s$  the sound and  $v$  the volume of the event. Let  $e_1 = (t_1, d_1, f_1, s_1, v_1)$  and  $e_2 = (t_2, d_2, f_2, s_2, v_2)$  be two events. The operators of concatenation, denoted by  $\cdot$ , and superimposition, denoted by  $|$ , are defined by:  $e_1 \cdot e_2 \Rightarrow t_1 + d_1 = t_2$ ;  $e_1 | e_2 \Rightarrow t_1 = t_2, d_1 = d_2$ .

### 2.2 Syntactic Equations

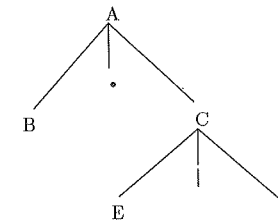
The temporal structure of a piece is defined by the means of syntactic equations whose forms are given by general syntactic equations. For example, let us define the two syntactic equations that will be used in this paper:  $e :: A = B \cdot C$ ,  $e_1 :: A = B | C$  where  $A$ ,  $B$  and  $C$  represent musical pieces.

The equation  $e$  means that the piece  $C$  is concatenated to the piece  $B$ , i.e. it starts exactly when  $B$  ends. The piece  $A$  is the concatenation of  $B$  and  $C$ . The equation  $e_1$  means that the two pieces  $B$  and  $C$  start and end at the same time. The piece  $A$  is the superimposition of  $B$  and  $C$ .

### 2.3 Equational Program or Hierarchy

An equational program is a set of syntactic equations. We will only consider equational programs which can be represented by a tree i.e each symbol of musical piece occurs at most one time in the left hand side

**Example:** Let  $P = \{e_1 : A = B \cdot C, e_2 : C = E | F\}$ . The tree associated with the equational program  $P$  is:



## 3 Musical Systems

The semantics of a hierarchy is built by a kind of union (called a *cartesian union*) of the musical systems of each syntactic equation composing the equational program representing the hierarchy. In what follows, we will only study the case of the characteristics measure and duration which will be denoted, respectively, by  $m$  and  $d$ .

### 3.1 Musical Systems associated with Syntactic Equations

We introduce the concept of musical systems associated with a syntactic equation with the two following examples. The reader interested in the formal definition can refer to (Barbar, Desainte-Catherine, Beurivé 1994). We give two examples of musical systems for the characteristics measure.

**Example 3.1** Let  $S_{e,m} = \{s_1, s_2, s_3, s_4\}$  be the set of equations systems associated with the syntactic equation  $e :: A = B \cdot C$ , where

$$\begin{aligned}
 (s_1) \left\{ \begin{array}{l} m(B) \neq \varepsilon, m(C) \neq \varepsilon, \\ m(B) \neq m(C), m(A) = \varepsilon, \\ (m(A), m(B), m(C)) \in \text{Domain}(m)^3 \end{array} \right. & \quad (s_2) \left\{ \begin{array}{l} m(A) \neq \varepsilon, m(A) = m(B), \\ m(A) = m(C), \\ (m(A), m(B), m(C)) \in \text{Domain}(m)^3 \end{array} \right. \\
 (s_3) \left\{ \begin{array}{l} m(A) = \varepsilon, m(B) = \varepsilon, \\ (m(A), m(B), m(C)) \in \text{Domain}(m)^3 \end{array} \right. & \quad (s_4) \left\{ \begin{array}{l} m(A) = \varepsilon, m(C) = \varepsilon, \\ (m(A), m(B), m(C)) \in \text{Domain}(m)^3 \end{array} \right.
 \end{aligned}$$

The value  $\varepsilon$  denotes a measure that is not constant. The musical meaning of this musical system is the following:

- When two parts have different measures, the measure of their concatenation is not constant.
- When two parts have the same measure  $m$ , their concatenation has also the measure  $m$ .
- If a part  $A$  has got a measure  $m$ , every subpart of  $A$  gets the measure  $m$ .

Let us now define a simple system for the measure and the superimposition operation: two parts that are superimposed have the same measure. The set of systems is reduced to the following equation system:

$$(s_5) \left\{ \begin{array}{l} m(A) = m(B), m(A) = m(C), \\ (m(A), m(B), m(C)) \in \text{Domain}(m)^3 \end{array} \right.$$

**Example 3.2** The following musical system for the measure involves also the characteristics duration, denoted by  $d$ . Let  $S_{e,m} = \{s_8, s_9, s_{10}\}$  be the set of equations systems associated with the syntactic equation  $e :: X = Y \cdot Z$ , where

$$(s_8) \begin{cases} m(Y) \neq m(Z), \\ d(Y) > d(Z), \\ m(X) = m(Y) \\ (m(X), m(Y), m(Z)) \in \text{Domain}(m)^3 \end{cases} \quad (s_9) \begin{cases} m(Y) \neq m(Z), \\ d(Y) < d(Z), \\ m(X) = m(Z) \\ (m(X), m(Y), m(Z)) \in \text{Domain}(m)^3 \end{cases}$$

$$(s_{10}) \begin{cases} m(X) = m(Y) \\ m(X) = m(Z), \\ (m(X), m(Y), m(Z)) \in \text{Domain}(m)^3 \end{cases}$$

### 3.2 Musical Systems associated with Equational Program

We have defined the formal object representing musical systems involving syntactic equations and sets of characteristics. Now, from small pieces which are those musical systems, let us define how to build the musical system which is associated to a whole equational program. For this purpose, we introduce the cartesian union operation which simplifies the final definition.

**Definition 3.3** Let  $E_1$  and  $E_2$  be two finite sets of sets. The cartesian union of  $E_1$  and  $E_2$  is defined by:  $E_1 \uplus E_2 = \{e_1 \cup e_2 | e_1 \in E_1, e_2 \in E_2\}$ .

**Example 3.4** Let  $E_1 = \{\{1, 2, 3\}, \{4, 5\}, \{6\}\}$ ,  $E_2 = \{\{a, b\}, \{c\}\}$ , then  $E_1 \uplus E_2 = \{\{1, 2, 3, a, b\}, \{1, 2, 3, c\}, \{4, 5, a, b\}, \{4, 5, c\}, \{6, a, b\}, \{6, c\}\}$ .

Let be  $P$  an equational program. We denote by  $e$  any syntactic equation in  $P$  and by  $S_e$  the set of equations systems associated with  $e$ . Let be  $\Gamma$  a set of characteristics. Then, the set of equations systems associated with the equational program  $P$  is:

$$S = \biguplus_{e \in P} \left( \biguplus_{\gamma \in \Gamma} s_{e,\gamma} \right)$$

where  $s_{e,\gamma}$  is the musical system associated with the equation  $e$  for the characteristics  $\gamma$ .

**Example 3.5** Let  $P$  be the equational program  $\{e_1 : A = B.C, e_2 : C = E|F\}$ . Then, we have:

- the musical systems associated with the equations  $e_1$  and  $e_2$  are:

$$\begin{aligned} -s_{e_1,m} &= \{s_1, s_2, s_3, s_4\}, \text{ the equation systems given in 3.1} \\ -s_{e_1,d} &= \{s_5\} = \{d(A) = d(B) + d(C), d(A) \geq 0, d(B) \geq 0, d(C) \geq 0\} \\ -s_{e_2,m} &= \{s_6\} = \{m(C) = m(E), m(C) = m(F), (m(C), m(E), m(F)) \in \text{Domain}(m)^3\} \\ -s_{e_2,d} &= \{s_7\} = \{d(C) = d(E), d(C) = d(F), d(C) \geq 0, d(E) \geq 0, d(F) \geq 0\} \\ -s_{e_1} &= s_{e_1,m} \uplus s_{e_1,d} = \{s_1 \cup s_5, s_2 \cup s_5, s_3 \cup s_5, s_4 \cup s_5\} \\ -s_{e_2} &= s_{e_2,m} \uplus s_{e_2,d} = \{s_6 \cup s_7\} \end{aligned}$$

- the musical systems associated with the program  $P$  are:

$$S = \biguplus_{e \in \{e_1, e_2\}} \left( \biguplus_{\gamma \in \{m, d\}} s_{e,\gamma} \right) = s_{e_1} \uplus s_{e_2} = \{s_1 \cup s_5, s_2 \cup s_5, s_3 \cup s_5, s_4 \cup s_5\} \uplus \{s_6 \cup s_7\}$$

$$S = \{s_{1,5,6,7}, s_{2,5,6,7}, s_{3,5,6,7}, s_{4,5,6,7}\}$$

where for all  $i, j, k, l : s_{i,j,k,l} = s_i \cup s_j \cup s_k \cup s_l$ . As example:

$$s_{2,5,6,7} = \begin{cases} m(A) \neq \varepsilon, m(A) = m(B), m(A) = m(C), (m(A), m(B), m(C)) \in \text{Domain}(m)^3 \\ d(A) = d(B) + d(C), d(A) \geq 0, d(B) \geq 0, d(C) \geq 0 \\ m(C) = m(E), m(C) = m(F), (m(C), m(E), m(F)) \in \text{Domain}(m)^3 \\ d(C) = d(E), d(C) = d(F), d(C) \geq 0, d(E) \geq 0, d(F) \geq 0 \end{cases}$$

## 4 Musical Equational Program

A musical equational program (MEP) is the main object of our model. It represents the state of the composing process at one time, that is:

- the state of the hierarchy, which is represented by a set of syntactic equations,
- the state of the musical system which is associated to the current hierarchy,
- the set of all the assignments of some parts characteristics that have either been given by the composer or either been computed from the musical system.

**Definition 4.1** A musical assignment is an equation of the form  $c(A) = v$  where  $c$  is a characteristic symbol,  $A$  is a piece symbol and  $v$  a value in the domain of  $c$ .

**Example 4.2** Let be the following MEP:  $\prec \{e_1 : A = B.C, e_2 : C = E|F\}, \{s_{e_1}, s_{e_2}\}, \{m(A) = 3/4, d(E) = 10\} \succ$ , where  $s_{e_1}$  and  $s_{e_2}$  are the musical systems of example 3.5.

### 4.1 Solutions of musical equational systems

Intuitively, the solution of a MEP is the intersection of non empty solutions of all musical systems associated with the equational program.

**Definition 4.3** Let  $\prec P, S, G \succ$  be a MEP. Let  $\text{sol}(s)$  be the set of all the solutions of a system  $s \in S$ , each solution being given by a set of assignments of the form  $c(A) = v$  where  $c$  is a characteristic and  $A$  is a symbol representing musical piece. Let be  $\text{sol}_G(s) = \{\sigma \in \text{sol}(s) | G \subset \sigma\}$  and  $\text{sol}_G(S) = \bigcup_{s \in S} \text{sol}_G(s)$ . The

solution of  $\prec P, S, G \succ$  is the set of assignments  $\bigcap_{\sigma \in \text{sol}_G(S)} \sigma$ . So we will write  $\prec P, S, G \succ \vdash \bigcap_{\sigma \in \text{sol}_G(S)} \sigma$ .

The solution of  $\prec P, S, G \succ$  is the empty set if  $G$  does not constitute a part of some solution of  $S$ . In that case, the system  $\prec P, S, G \succ$  is said to be **invalid** (or not consistent).

**Example 4.4** Let us consider the MEP  $\prec P, S, G \succ$ , where  $S$  is given in example 3.5 and  $G = \{m(A) = 3/4, d(E) = 10\}$ . Then, we have:

$$\text{sol}_G(s_{2,5,6,7}) = \{m(A) = m(B) = m(C) = m(E) = m(F) = 3/4, d(E) = d(F) = d(B) = 10, d(A) = d_A, d(C) = d_C\} / d_A - d_C = 10, \text{ (it contains an infinite number of solutions)}$$

$$\text{sol}_G(s_{1,5,6,7}) = \text{sol}_G(s_{3,5,6,7}) = \text{sol}_G(s_{4,5,6,7}) = \emptyset.$$

Thus, the set of all solutions of  $S$  is  $\text{sol}_G(S) = \bigcup_{s \in S} \text{sol}_G(s) = \text{sol}_G(s_{2,5,6,7})$

$$\text{and the solution of } \prec P, S, G \succ \text{ is } \bigcap_{\sigma \in \text{sol}_G(S)} \sigma = \bigcap_{\sigma \in \text{sol}_G(s_{2,5,6,7})} \sigma =$$

$$\{m(A) = m(B) = m(C) = m(E) = m(F) = 3/4, d(E) = d(F) = d(B) = 10\}.$$

**Definition 4.5** A MEP  $\prec P, S, G \succ$  is saturated if  $\prec P, S, G \succ \vdash G$ .

**Example 4.6** The MEP  $\prec P, S, G \succ$  given in the previous example is not saturated because the assignments  $m(B) = m(C) = m(E) = m(F) = 3/4, d(F) = d(B) = 10$  do not belong to  $G$ . On the contrary,  $\prec P, S, G \cup \{m(B) = m(C) = m(E) = m(F) = 3/4, d(F) = d(B) = 10\} \succ$  is saturated.

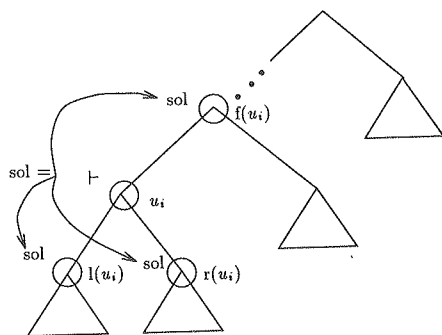
We note that here we calculate the solutions of the musical system associated with the whole hierarchy (or tree) with respect to all assignments given by the composer. An interesting way for the determination of the solutions is the elimination of all invalid musical systems each time the composer gives an assignment.

## 5 The Incremental Strategy

The incremental evaluation on a musical equational program is the computation of a solution step by step. It consists of the computation of assignments which are deduced by the musical systems with initial assignments which are given by the composer. Let  $\langle P, S, G \rangle$  be a saturated musical equational program. A slight modification of  $\langle P, S, G \rangle$  implies modification of the solution. The incremental strategy consists of the computation of the new solution by modifying the old one without computing again the whole solution. Now, we give the principle of the incremental evaluation on a hierarchical structure. The nodes of the tree are denoted by  $u_1, \dots, u_n$ . We start with a saturated MEP  $\langle P, S, G \rangle$  associated with the tree. Then we add a new assignment  $g_i$  on a variable of the sub-system associated with the node  $u_i$ . Then, in order to saturate  $\langle P, S, G \cup g_i \rangle$  i.e. to calculate the solution  $G'$  (s.t.  $\langle P, S, G \cup g_i \rangle \vdash G'$ ), we proceed as follows:

- we calculate the solution of the sub-system at the node  $u_i$  w.r.t the assignments  $G \cup g_i$ ;
- we propagate to the father and the sons of the node  $u$  the assignment of the solution which concerns variables in their sub-systems and so on.

We give a recursive function *sol* for the computation of the solution of the musical equational program. This is represented in the following schema:



It shows the decomposition of the function *sol* at the node  $u_i$  in a resolution ( $\vdash$ ) of the musical systems at  $u_i$  and three recursive calls to the father, left son and right son of  $u_i$  which are denoted respectively, by  $f(u_i)$ ,  $l(u_i)$  and  $r(u_i)$ , on the figure. The definition of the relation  $\vdash$  can be given by an automata (see (Barbar, Desainte-Catherine, Beurivé 1994))

## 6 Conclusion

We have presented a model for representing musical pieces without repetitions by the means of a temporal hierarchy. Moreover, this model provides a way to compute automatically some musical characteristics by using equations systems and values that are given by the composer. The result is a very efficient software based on automatas solving the systems. Now, the power of expression of the model is too restrictive. It is necessary to integrate repetitions and several concurrent structurations. Those extensions will complexify the model and improve its efficiency. Now, we are currently working on the concept of abstraction of musical hierarchies for representing musical forms and items in the context of an interface for the composer. The model would then be useful for analyzing too. At last, the study of operators on other musical dimensions will increase again the power of structuration of the composer (and the power of expression of an analysis).

## References

- Balaban, M. (1991). Music structures: the Temporal and Hierarchical Aspects in Music. Technical report FC-035 MCS-327, Ben-Gurion University of the Negev.
- Balaban, M. & Samoun, C. (1993). Hierarchy, Time and Inheritance in Music Modelling. *Languages of design*, 1(3).
- Allen, J.F. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11), pp.832-843.
- Marsden, A. (1994). The Representation of Temporal Relations in Music writing in progress.
- Barbar, K. & Desainte-Catherine, M. (1992). Using attribute to find solutions for musical equational programs. Technical report 92-77, LaBRI, Univ BX-1.
- Barbar, K. Desainte-Catherine, M. & Miniussi, A. (1993). The semantics of a musical hierarchy. *Computer Music Journal*, 17(4), pp 30-37.
- Barbar, K. Desainte-Catherine, M. & Beurivé, A. (1994). Incremental Resolution of Musical Systems. Technical report 964-94, LaBRI, Univ BX-1.

## CAMM - Automatic Composer of Musical Melodies

Eloi Fernando Fritsch \*  
Rosa Maria Viccari †

### Abstract

The purpose of this paper is to present the implementation of a grammar-based software named CAMM - Automatic Composer of Musical Melodies (or in Portuguese Compositor Automático de Melodias Musicais, in portuguese) - which is capable of generating melodies. CAMM has a set of rules that represent the musical knowledge needed to generate simple melodies in a limited and well-defined musical universe, i.e. style.

Musical parameters, such as notes, durations and intensities, are put together by means of a grammar in order to generate simple melodies which can be used for the composition of pieces of music.

### 1 Introduction

In order to produce a system that uses rules to compose melodies (Miranda, 1990), it is necessary to study how music can be composed and the ways to compose, to arrange in musical harmony and to improvise in music (Cope, 1987). Based on these compositional principles, it is possible to extract certain cognitive aspects of music composition with a computational approach. In this manner, we believe that it is possible to abstract from the musical universe certain aspects related to the task of melody composition.

We believe that musical models and structures represented in the listener's mind are made using three basic components :

- Melody
- Harmony
- Rhythm

In order to represent these models and structures in the computer it is necessary to define the rules that govern the relationship between their components. The musical language, from the computational point of view, must be treated as an organized symbol system (Roads, 1985). This enables the creation of the grammar syntax and rules. From now on we will refer to these rules as components of a grammar.

CAMM was developed with the following objectives:

- to create musical grammars using the Prolog programming language to represent the musical knowledge;

\*MsC student in Computer Science (CPGCC/UFRGS), Graduated in Computer Science (UCS, 1991). Areas of interest: Artificial Intelligence & Music. Universidade Federal do Rio Grande do Sul, Instituto de Informática - UFRGS, Curso de Pós-Graduação em Ciências da Computação - CPGCC, Av. Bento Gonçalves, 9500 Bloco IV - Agronomia - Campus do Vale, CEP 91501-970 - Porto Alegre - RS - Brazil, Caixa Postal: 15064 FAX: ++55 (051) 336-5576, E-mail: fritsch@inf.ufrgs.br

†PhD in Computers and Electronics Engineering (Coimbra/Portugal). Areas of interest: Artificial Intelligence, Logic Programming. Universidade Federal do Rio Grande do Sul, Instituto de Informática - UFRGS, Curso de Pós-Graduação em Ciências da Computação - CPGCC, Av. Bento Gonçalves, 9500 Bloco IV - Agronomia - Campus do Vale, CEP 91501-970 - Porto Alegre - RS - Brazil, Caixa Postal: 15064 FAX: ++55 (051) 336-5576, E-mail: rosa@inf.ufrgs.br

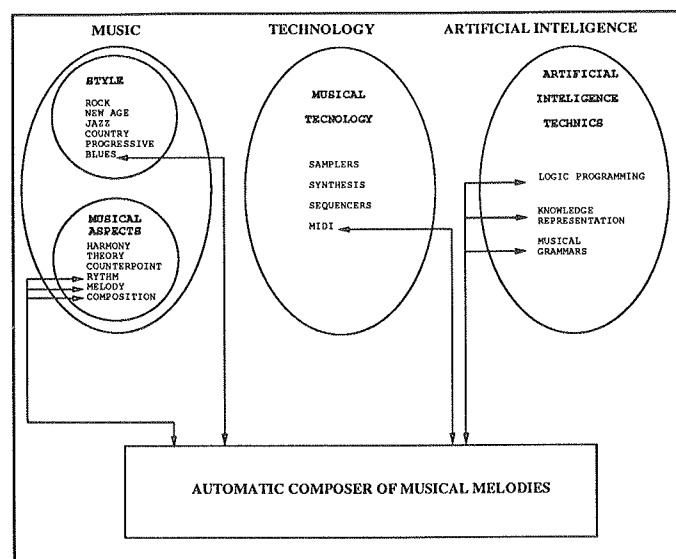


Figure 1: Representation of the universe of music, science and technology used to model CAMM

- to construct a musical program capable of using the MIDI system resources, such that the melody generated can be executed by an instrument and stored in a sequencer;
- to use the program in the composition of melodies where musical phrases generated by the program may be used by a human composer in a piece of music;
- to show that some melodies generated by a grammar written in Prolog may be artistically beautiful;

Although there are other softwares similar to CAMM, the system architecture, the heuristics and the grammar used in CAMM are slightly different. Among many musical systems used for automatic music composition we can mention here two systems which share similar characteristics with CAMM: EMI and the Computational Generation and Study of Jazz Music. The EMI system (Experiments in Music Intelligence), created by David Cope, also uses grammars. It is more complex though. It works with many other different musical aspects, such as the ability to manipulate intervals (Cope, 1987). In CAMM we limit the scope of the grammar to fewer parameters. The Computational Generation and Study of Jazz Music, by Francesco Giomi and Marco Ligabue, generates harmonic paths e improvises jazz melodies (Giomi, 1989). This system uses rhythmic cells similar to those of CAMM and also has more sophisticated functions. Nevertheless, CAMM approaches mainly melodic aspects, instead of harmonic ones and more, it generates blues melodies, and not jazz melodies.

As we are interested in the use of AI in music we selected a declarative programming style for implementation using Prolog. There are many other systems programmed in a declarative way, such as the Program for Music Segmentation, by John Roeder and ARTIST (for Artificial Intelligence-based Synthesis Tools) by Eduardo Miranda (Miranda, 1994). The former does not generate music though. Roeder's system was developed with pedagogic interest in the field of music analysis (Roeder, 1989). However it is very inspiring how Roeder uses a kind of grammar-orientated paradigm for music segmentation. The latter is a system that uses a kind of natural language (i.e. words in English) to communicate with a synthesiser aimed for producing sounds from qualitative, perceptually-orientated descriptions.

## 2 HARMONY AND MUSICAL IMPROVISATION

Most popular music styles which involves improvisation, such as contemporary jazz, originated from blues. Thus we selected blues as the musical style to be studied in this work. From the melodic point of view, a blues scale may be used very efficiently within the twelve bars, typically found in blues (Prediger, 1983).

CAMM uses only one scale at a time to build a complete melody. The amount of variations that CAMM is able to generate is ilimited, taking into account the amount of possible combinations of the notes and of rhythmic figures that would be possible to do. On the one hand this limitation is good because it constraints the system to produce, let us say, only a small set of consistent melodies according to a simple grammar. On the other hand the output can get quite repetitive and loose musical interest.

## 3 BASIC ASPECTS OF IMPLEMENTATION

CAMM's grammar deals with four basic parameters of a melody:

- notes
- durations
- pauses
- intensities

We represent these musical parameters in textual form. Each statement has its corresponding MIDI code (Gomes, 1988). The distribution and grouping of parameters are arranged according to their use in the construction of musical phrases. For example: all the scales and all rhythmic cells are grouped. In this manner we can have musical knowledge bases that represent exactly the four parameters mentioned above.<sup>1</sup> Other parameters are already completely represented in the knowledge base. They do not need to be linked to other parameters to mean something. Pauses, for example, do not need to be mapped to intensities nor to notes.

The DGC formalism (Definite Clause Grammar) and Prolog language are used to build the musical grammars (Ariy, 1986). With the use of the DCG formalism it has been possible to write the grammar in a simple way (Viccari, 1992). The grammar provides means for the system to select, from a finite set, the parameters which generates a musical event.

CAMM still misses a bold graphic interface. For this reason, the program input is directly made in the interpreter's command line, typing the appropriate command followed by parameters. The first parameter is the name of the scale to be used. The second parameter is the melody's tempo. Since the melody is always built on a quaternary rhythm, what varies is the duration of notes. Therefore, the melody's tempo will be slower if it has time figures with bigger values and it will be faster if it has time figures with smaller values. The third parameter is the melody's intensity, i.e. how loud its notes should sound.

For example: `printtema(a,lento,forte)`. means that the grammar will compose a melody using the A scale, with a slow tempo (*lento*) and that most of the notes have to be played very loud (*forte*).

## 4 THE NOTE PARAMETER

The way CAMM treats musical notes is based upon the way we believe pop music composers do (Fritsch, 1992) (see fig 2). CAMM has a knowledge base that contains scales of blues. The user selects the scale that will be used to produce the melody. Each scale has a set of notes whose sequency is determined by a distribution function.

<sup>1</sup>In a later step, these elements are gathered. For example: a musical note without duration and intensity can not be part of the context of the melody, but when this same note is mapped to a duration value and to a intensity value, than it becomes to be part of the melody.



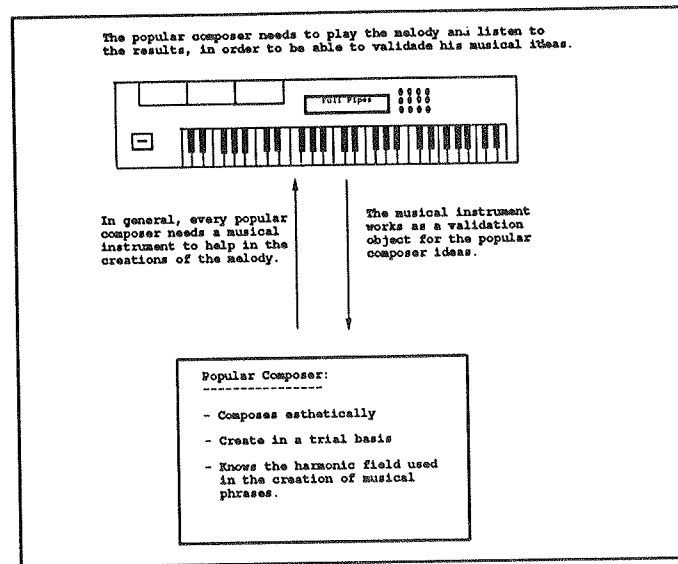


Figure 2: An attempt to model the cognitive process of music creativity.

Once the user has selected a scale CMM will manipulate the relation among its notes based on the harmonic field of the scale. We say that this harmonic field defines how notes are related to each other. The interaction between the user and CMM is illustrated in Fig. 3.

There is no deterministic mechanism which defines the order or how many times the same notes will occur in a melody because, as shown in Fig. 3, the purpose of CMM is to present new options to the composer. Sometimes these will be very unpredictable indeed.

The repetition of the function that selects notes is determined by the sum of the durations which they are mapped to. When the twelve bars are filled with notes concatenated with its respective duration, then the function that selects notes will not be invoked anymore and the composition will be sent to system output.

The blues scales that compose the knowledge base are: C, F, Bb, Eb, Ab, Db, F#, B, E, A, D,

G.

Figure 4 shows the possible values for the first parameter of the grammar. Each of these scales is just a list of coded notes, treated as such by the program.

As an example, we present a scale extracted from the CMM knowledge base:

```
notas_de_blues (c, [ nota (c1,60), nota (eb1,63), nota (f1,65),
nota (gb1,66), nota (g1,67), nota (bb1,70), nota(c2,72) ] ).
```

The c, outside the square brackets, is a constant which indicates that the notes set is a C scale. All notes have the same chance of being chosen. Since they are seven notes, each one has a 14.3% chance of being selected. The decision for the octaved tonic was made according to an aesthetic experiment made by the author. As shown in Fig. 5, the Prolog program can be altered, in order to increase or decrease the probability to generate specific notes (Roeder, 1989). Since the tonic represents a rest sensation when

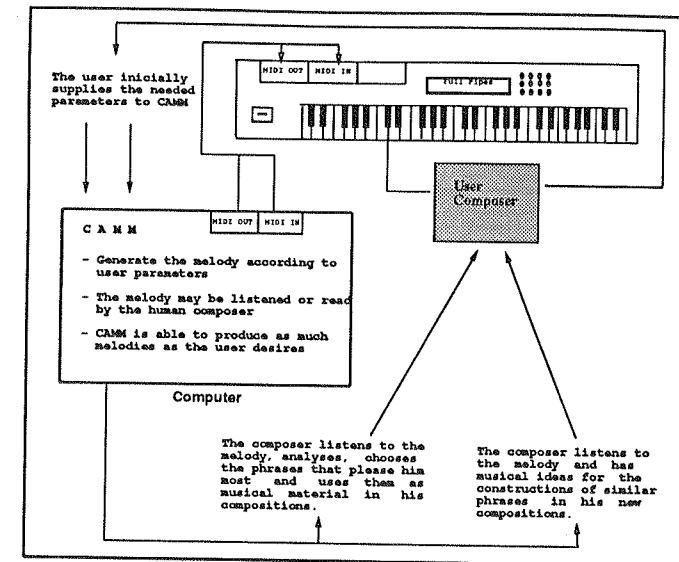


Figure 3: Process where CMM provides melodic phrases, unexpected by the composer, which originate new musical ideas that, later on, will be transformed in new compositions.

it is played, it was set to be selected with more probability in order to create a good musical output <sup>2</sup>.

If the knowledge base has to be altered, it is necessary that the music expert and the knowledge engineer interact. Ideally both should be the same person. The knowledge engineer's function is to represent the experience of the music expert, altering facts and rules already in CMM.

As mentioned earlier, CMM generates notes in textual and coded representation. For this reason, each note in the scale is associated to its decimal representation of the standard MIDI code (Yavelow92). Hence, when the grammar generates the textual musical notes, it will also generate a set of MIDI messages for the synthesiser. The codification for flat notes and sharp notes are the same. This means that, for example, the code for a F# is the same than that for a Gb, if they are in the same octave. The current octave is indicated by the number that follows the note, e.g. C1 corresponds to the central C of the piano, C2 one octave above the central C of the piano, and so on.

CMM uses a distribution function to determine whether the events of the melody will be a note or a pause. However the probability for selecting a note is much superior than for selecting a pause. Otherwise the output would probably have only dispersed notes and a few individual short sequences - which is not the characteristic of blues melodies.

## 5 THE DURATION PARAMETER

In CMM, before the melody is outputted, its notes are mapped to duration values. Durations values are written textually along the program.

A bar is constituted by pauses and notes with their durations. Each bar generated by CMM uses by default a 4/4 rhythm. CMM selects and fill a bar duration values according to the space still available after the previous selection.

<sup>2</sup> according to the authors's musical taste.

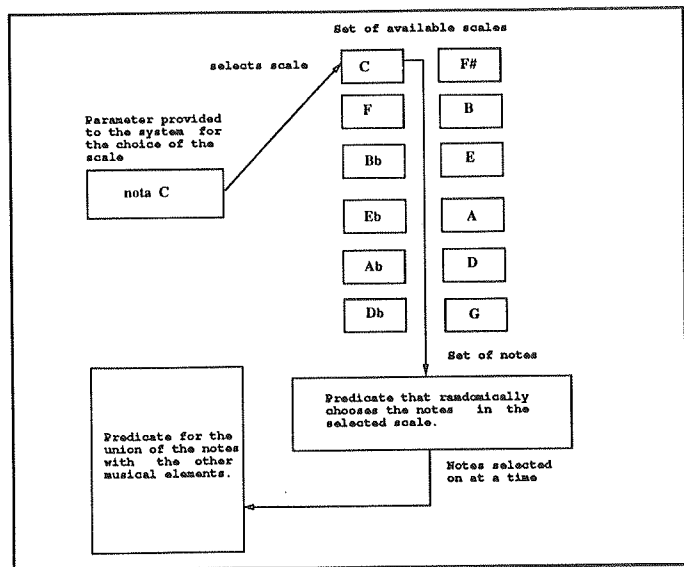


Figure 4: Selecting notes from a blue scale.

Depending on the tempo specified by the user, the duration values will be longer or shorter. If they are longer the melody will have a slower tempo and if they are shorter the melody will have a faster tempo.

There are two ways to provide duration values to notes: by selecting a value for each individual note or by selecting a rhythmic cell, i.e. a short rhythm pattern. In the current version of the program there is a probability of 60% that the system uses the latter method and 40% the it uses the former method.

### 5.1 SELECTING DURATION VALUES

CAMM selects duration values according the tempo of the melody. Tempo is provided by the user. There are three options for tempo: slow tempo, medium tempo and fast tempo. See below an example of a set of duration values for the slow tempo.

```
fig_de_tempo (
lento, [semibreve,semibreve,semibreve,semibreve,
semibreve,semibreve,semibreve,semibreve,
minima_pontuada,minima_pontuada,minima_pontuada,
minima_pontuada,minima_pontuada,minima_pontuada,
minima_pontuada,minima,minima,minima,minima,
minima,minima,minima,minima,minima,minima,minima,
minima,minima,minima,minima,minima,minima,minima,
minima,minima,minima,seminima_pontuada,seminima_pontuada,
seminima_pontuada,seminima_pontuada,seminima,seminima,
colcheia_pontuada,colcheia,semicolcheia_pontuada,
semicolcheia,fusa_pontuada,semifusa]).
```

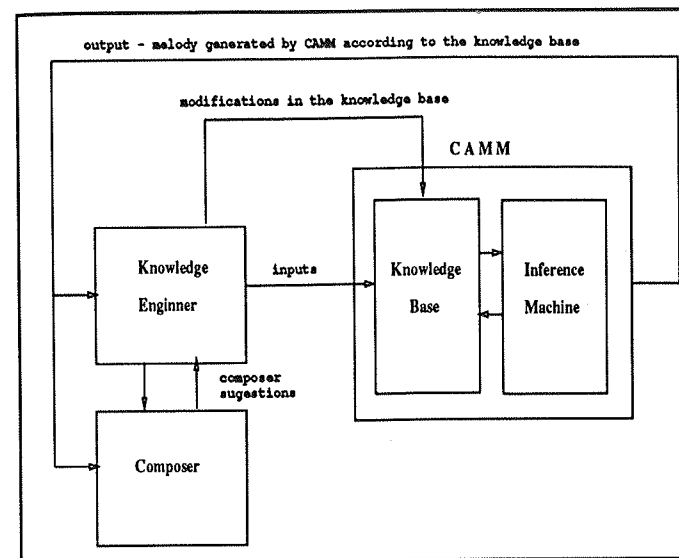


Figure 5: Knowledge Base modification schema, according to the needs of the composer

In the above example, extracted from CAMM's knowledge base, we can observe that some duration values, or duration figures if you like, are repeated and that, for this reason, they will have a bigger probability to be selected. For this reason it is more likely that longer duration values will be selected more often, originating a slower tempo rather than a fast one.

CAMM'S grammar also features a mechanism for providing bars with duration figures. For each bar it considers the time space still available, the tempo specified by the user and what is available in the knowledge base in the terms of valid duration figures and rhythmic cells. As the available time space in a bar decreases CAMM selects those time figures which still fit. CAMM's grammar was designed such that it avoids to attach very short durations to notes too often. This is not desired here specially when the melody's tempo is to be slow. The same rule is also considered when the melody's tempo is to be medium or fast. Notes with short duration (relative to the tempo of the melody) are avoided and notes with long duration are encouraged. This is so because we want to avoid beginning a bar with, let us say, long duration notes and dratically ending it with short notes.

Speaking in terms of numbers, the algorithm works as follows: If the total duration of a bar is other than 64 (which corresponds, let us say, to four quarter-notes) then it will try to select a suitable figure. In the case of a slow tempo, for example, in order to select figures that are not too short it tests if the difference between the total size of the bar and the space still available is bigger than or equal to 8 (eight-note). Eight is the value of an eight-note which is the smallest allowed duration figure for the rest of the bar. Following this rule, CAMM will only choose figures whose values are bigger than or equal to the eight-note to fill the bar. The same is valid for the medium tempo and for the fast tempo. The difference is that for a medium tempo the shortest value is 4 i.e. (a sixteenth note) and for the fast tempo the shortest value is 2 (thirty-second note) (see Fig. 6).

### 5.2 RHYTHMIC CELLS

The rhythmic cells, or rhythmic patterns, are sets of pre-defined duration figures stored in CAMM's knowledge base. In the current version the knowledge base has stored 75 different rhythmic

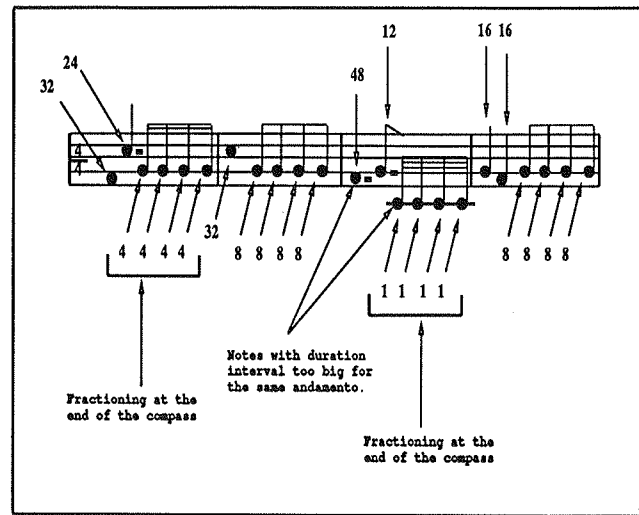


Figure 6: The problem of finding right the duration value for the end of the compass.

cells. These cells are different of each other and may be used for all. Each cell is also characterized for its own duration, which is the sum of the durations of its duration figures.

The statement bellow (one of the 75 rhythmic cells) obtains the set of rhythmic cells through the predicate `andamento_cel_rit`, according to the parameter supplied by the user. In this manner, the `Conj_cel_rit` variable receives the group of cells related to the referred tempo. The `random_pick` predicate randomly chooses just one element from the list and passes this element to the `dur_celular` predicate, which computes the total size of the rhythmic cell. The `Duracao_celular` variable is instantiated with the total duration of the chosen cell. If the size of the chosen rhythmic cell is bigger then the available space left in a bar, then the system selects another cell that fits.

```

células_rítmicas(P1,P2,P3,Cont)->
{ andamento_cel_rit(P2,Conj_cel_rit),
  random_pick(Conj_cel_rit,X), dur_celular(X,Duração_celular),
  Dur_Tot is Duração_celular + Cont, Dur_Tot = < 64 },
célula_rit(X,P1,P3), continua_compas(P1,P2,P3,Dur_Tot).

```

## 6 THE PAUSE PARAMETER

Pauses are manipulated by the grammar in a similar way to notes (see Fig. 8). The difference is that notes are entities outside time at the moment of selection. Only afterwards a note is mapped to a duration value (which is also given by the grammar). On the other hand, pauses don't need to be mapped

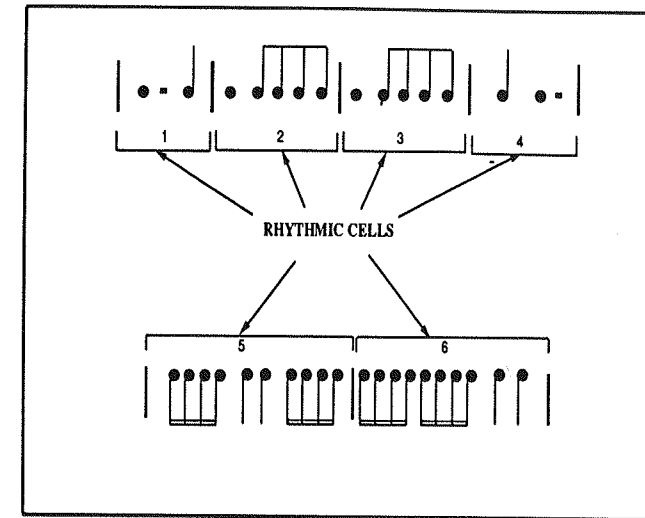


Figure 7: Rhythmic Cells

to a duration value because their representation already includes an inherent duration value.

Pauses should not occur too frequently in a melody. This could produce too many gaps in the melody. Nevertheless, pauses should exist, in order to strengthen the rhythmic sensation of musical phrases.

## 7 THE INTENSITY PARAMETER

Another important aspect of a melody is the expressiveness given to its performance. This aspect has to do with the music dynamics, i.e., whether it is played in a loud, medium or quiet way. So CAMM also maps an intensity value to each note of the melody. The user also informs the system the intensity he or she wants. For example if the user wants a loud melody, then most of the intensity values will be within a loud bandwidth of values.

For a better representation of the intensity parameter with which the melody notes must sound, the intensity values are divided into groups that correspond to a particular kind of expression. For example, for loud melodies values within a bandwidth of loud intensity values will have a higher probability to be selected than any other values.

## 8 PRESENTATION OF THE GENERATED MELODY

Once CAMM creates a melody according to the parameters provided by the user, it can present it in two ways: textually and coded. Both outputs are displayed in the screen. Alternatively, melodies can also be saved in a MIDI file (Ratton, 1992).

The textual presentation was devised to ease the visualisation of the MIDI file and the notes played by the instrument. Through this textual presentation, any user, with a minimum understanding of music, can transcribe the output to traditional music notation when writing larger pieces of work by hand. This task could also be automatically accomplished by sequencer with music notation facilities.

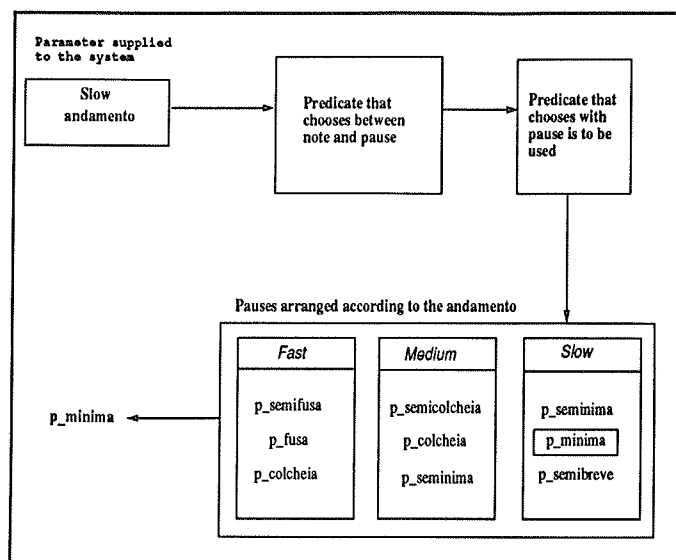


Figure 8: Hence, the pauses are also chose by a randomic predicate.

We illustrate below a melody in its textual form. Here the user inputted: note A, slow tempo and loud intensity.

```

di_minima_mf gi_minima_f / ci_minima_f ebi_minima_f /
gi_seminima_pontuada_mf p_minima ebi_colcheia_f / ci_minima_f ai_minima_f //
ei_minima_f ebi_colcheia_f di_seminima_f ai_colcheia_f /
ci_minima_f di_minima_f //
ci_minima_f p_seminima ci_colcheia_f ai_colcheia_f /
ci_minima_f ci_minima_f //
p_minima di_minima_mf //
ai_minima_f gi_colcheia_f gi_seminima_f ai_colcheia_f //
ci_seminima_pontuada_mf di_seminima_pontuada_f ci_semicolcheia_pontuada_f
gi_semifusa_mf ei_semifusa_f ci_colcheia_f //
ebi_minima_f di_colcheia_f di_seminima_f ai_colcheia_f //

```

## 9 MIDI IMPLEMENTATION

In the MIDI file, only the codification for note duration is not implemented according to the standard MIDI specification (Ratton, 1992) (Yavelow, 1992). Our program that sends MIDI codes from files to the synthesiser uses a different technique for MIDI control, which is not aimed to be fully described in this paper (Korg, 1992). This program has been implemented in GFA Basic, running in a ATARI 1040 ST.

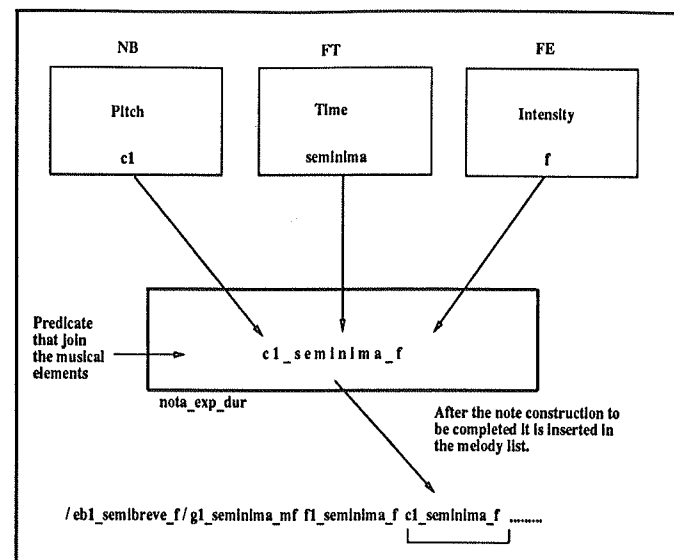


Figure 9: Mapping between musical elements.

The file produced by the grammar has the following format:

```
< note > < duration > < intensity >
```

where each of the three elements are formed by three decimal numbers.

The Amount of time between a message NOTE ON and a message NOTE OFF is specified by a loop in the program, which executes a wait function, CAMM performs this loop as many times as required to delay the program's flow.

The following example illustrates how the program send data to the MIDI interface using the OUT command (of ATARI's GFA Basic):

```

OUT 3,144
OUT 3,060
OUT 3,127

```

These three commands send through the MIDI channel 1 (represented by the MIDI code 144) a note C (represented by the MIDI code 60) with maximum intensity i.e. MIDI velocity (represented by the MIDI code 127). The number three, in each command, is used because it addresses the computer's output port 3. Any MIDI synthesiser may be used to play the generated melodies. Any MIDI instrument that owns the MIDI IN and MIDI OUT ports may be used to reproduce the generated sounds.

## 10 Conclusion

In this paper we introduced CAMM, a computer implementation of a grammar for automatic melody composition. In this work we wanted to show that the computer can compose interesting melodies using a simple grammar which defines their style. What is different in this work is that we use declarative programming for defining the grammar and for implementing the engine for melody composition. We believe that declarative programming is a very good way for communicating ideas to the computer. Rather than describing "how" the computer has to compose melodies, i.e. procedural programming, one needs only to describe "what" the machine has to do.

Although it in its infancy, CAMM proved to be a good starting point for future developments. Perhaps the next step is to provide a machine learning mechanism for automatic construction of grammars either from a set of examples or from user interaction. Also we plan to devise an interface aimed for enabling easy communication between the musician and the computer. At the moment the user still have to master Prolog in order to edit the grammar.

We have been effectively using CAMM. A Porto Alegre pop band has stored several CAMM generated melodies in a MIDI sequencer which are triggered during the show. Here the computer acts as another musician in the stage.

## REFERENCES

- ADOLFO, Antônio. (1983) *O Livro do Músico: Harmonia e Improvisação*. Rio de Janeiro: Lumiar Editora.
- ARITY Corporation. (1986) *The Arity/Prolog Programming Language*. ARITY Corporation.
- COPE, David. (1987). An Expert System for Computer-Assisted Composition. *Computer Music Journal*, 11/4, 30-46.
- FARIAS, Nelson (1963) *A Arte da Improvisação Para Todos os Instrumentos*. Rio de Janeiro, Lumiar Editora.
- FRITSCH, Eloi Fernando. (1993) *Um Estudo Sobre Música & IA e a Implementação de um Sistema Especialista Teórico Musical*. (Trabalho individual 301). Porto Alegre: CPGCC da UFRGS.
- GIOMI, Francesco. (1989) LIGABUE, Marco. *Computational Generation and Study of Jazz Music..*
- GOMES, Luis Carlos Elias (1988) *Som Três - Pequeno Dicionário MIDI*. São Paulo: Editora 8 Três.
- KORG Incorporation. (1992) *Music Workstation: 01/W FD Owner's Manual* Tokyo, Japan: KORG Incorporation.
- MIRANDA, Eduardo Reck. (1990) *Música e Inteligência Artificial Paradigmas e Aplicações*. Porto Alegre: CPGCC-UFRGS. (Trabalho Individual, 200)
- MIRANDA, Eduardo Reck. (1994) *From Symbols to Sound: AI Investigation of Sound Synthesis*. in *Contemporary Music Review*, in press.
- MOORER, J.A. (1975) *On the segmentation and analysis of continuous musical sound*. Stanford California: RepStan-m3, Dpto of Music, Stanford Univ.
- PREDIGER, José Aluísio. (1993) *Blues, Harmonia e Improvisação Musical*. Porto Alegre: Prediger Academia, Notas de Aula.
- RATTON, Miguel Balloussier. (1992) *MIDI: Guia Básico de Referência*. Rio de Janeiro: Editora Cam-

pus.

- ROADS, Curtis. (1985), Research in music and artificial intelligence. *Computing Surveys*, Cambridge, v.17, n.2.
- Roeder, John. (1989) *A Prolog Program for Music Segmentation*, School of Music, University of British Columbia, Musicus 1/ii, Dezembro.
- YAVELOW, Christopher. (1992) *Music & Sound Bible*. San Mateo, California, IDG Books WorldWide, Inc.
- VICCARI, Rosa. (1992) *Ferramentas para Inteligência Artificial*. Porto Alegre: CPGCC da UFRGS, Notas de Aula.

## Interfaces Musicais - Um problema antigo

*Domingos Aparecido Bueno da Silva  
Mestrando em Antropologia Social  
Progr. Pós Grad. em Antropologia Social  
Universidade Fed. de Santa Catarina  
Florianópolis - S.C. - Brasil  
Cx. P 476 - CEP 88010-970  
cso3dab@ibm.ufsc.br*

### Resumo

O interesse deste ensaio é tentar compreender até que ponto a notação musical, enquanto um sistema de signos que permite representar acontecimentos sonoros (limitados), deveria ser pensada não somente em termos de suas características intrínsecas, mas também enquanto uma interface entre o homem e a sua criação.

As questões referentes a esta interação/ferência que serão levantadas no decorrer do texto, procuram incentivar uma discussão mais aprofundada à cerca da nossa relação com o meio informático enquanto interface, levando-se em conta que, embora criativa e original, tem em seus princípios constitutivos, uma dependência muito grande com esta outra, a notação, com todas as implicações e limitações que lhe são características.

### A procura de um pensamento musical

Nesse texto, assumo a premissa de que devemos ampliar a discussão para além das definições binárias simplistas quando referimo-nos à cultura de outras sociedades, no nosso caso à música, (e por outras entendidas todas as culturas ágrafas e não ocidentais), quase sempre colocada em termos etnocêntricos do tipo nós/eles, desenvolvido/primitivo, sempre com um julgamento de (des)-valor, se quisermos efetivamente penetrar com mais profundidade no universo do acontecimento sonoro.

Nossos conceitos de desenvolvimento e progresso nos colocam frente a situações e definições em que o outro lado aparece desprovido de sentido ou de qualquer forma de lógica formal. A realidade, no entanto, é de que esses são os nossos padrões ocidentais de lógica, progresso desenvolvimento e mesmo linguagem, assim como são exclusivamente ocidentais a perspectiva, o racionalismo, o positivismo, o materialismo e o individualismo. O conceito de Tonalidade, assim como todas as suas ramificações dialéticas (atonalismo p. ex.), é exclusivamente ocidental e totalmente depende de um 'meio' simbólico específico para desenvolver-se.

Quer me parecer então que a utilização e o desenvolvimento no nosso cotidiano de compositores e/ou programadores, de samplers, filtros, aplicativos, interação c/ imagem, estaria um tanto comprometido enquanto interface, pela própria maneira como percebemos o universo sonoro, que em última instância, é condicionada pela cultura.

### A grafia e a notação musical

As questões ligadas às diferenças entre as culturas sempre foram alvo de interesse dos principais pensadores da história da humanidade. As teorias mais modernas dispensam quaisquer julgamentos de valor entre as sociedades, posicionando a cultura no centro das especulações.

Entre os antropólogos, foi Lucien Levy-Bruhl, filósofo e antropólogo francês, que primeiro coloca essa questão nestes termos, sugerindo que a explicação das diferenças entre culturas pudesse ser entendida à partir "do pensamento criador e dos processos mentais que, em cada e todas as sociedades, determinam sua cultura", distinguindo entre pensamento lógico e pré-lógico.

Já em 1962 um outro francês, Claude Levi-Strauss publicava "O pensamento Selvagem" onde delineava esta questão das diferenças em termos de modos de pensar, reconhecendo duas instâncias básicas: o pensamento selvagem, (concreto, sensorial e sensível), e o pensamento domesticado, (abstrato e racional), criticando Levy-Bruhl no sentido de que cada sociedade teria uma lógica própria. Levi-Strauss fala-nos em termos de dois níveis estratégicos, sendo "um muito próximo da intuição sensível e outro mais afastado", porém não nos dando nenhuma pista de em função de que se dariam esses dois níveis.

Será o inglês Jack Goody no seu livro "A domesticação do pensamento selvagem" que, retomando a problemática levantada por seus sucessores, e em especial Levi-Strauss, que parece ter chegado ao centro da questão. Sua discussão da problemática situa-se em como os modos de pensar e as formas de pensamento mudaram no espaço e no tempo. Vai demonstrar que são os instrumentos culturais que dispõe uma determinada sociedade e não outra, os determinantes de "estilos cognitivos, de modos de pensar o universo".

Não é nenhuma novidade que foi o surgimento da escrita o grande divisor de águas da história da humanidade. Em nossa civilização ocidental (e gráfica) não há nenhuma área em toda atividade humana, principalmente a nível do pensamento, que não tenha sido profundamente afetada pelo surgimento da grafia. A passagem do universo mágico ao científico, a organização do estado e da economia (a burocracia), do coletivo ao individual, foram mudanças tomadas possíveis, necessárias e por vezes, consequência desta nova interface.

Para Elsje M. Langrou, a abordagem de Goody é nova quando "entra mais fundo nas categorias de entendimento de povos com e sem escrita. Uma crítica e uma análise sistemática de uma informação supõe um distanciamento que só a escrita possibilita, e esta objetivação vai acompanhada de uma esquematização em listas e diagramas que abstraem e opõe, de uma maneira "coerente e consistente", fenômenos que, na fugacidade do fluxo do tempo, não aparecem desta maneira esquematizados e simplificados ...".

O mesmo se dá em relação à música. A possibilidade de registrar sons numa partitura, ou seja, o ato de registrar graficamente um som, priva-lhe de suas características eminentemente sensoriais/temporais (diferentemente de um quadro, p. ex., um acontecimento sonoro tem uma duração delimitada no tempo), e lhe confere um outro status, muito mais abstrato/estático; matéria básica para o seu desenvolvimento posterior.

A partitura, da mesma maneira que a escrita em relação à fala, estanca o fluir dos sons, permitindo então a busca de elementos de contradição e redundância, a harmonia, o contraponto, a orquestração, a reflexão sobre o desenvolvimento e a forma. É claro que todas estas características seriam impossíveis de serem trabalhadas de uma maneira puramente auditiva, ainda mais em peças relativamente longas.

O paralelo com a grafia é necessário pois, apesar da notação ser muito mais recente, as conseqüentes resultantes de seu surgimento já demonstradas sugerem grandes semelhanças, principalmente no sentido em que ambas criam novos "estilos cognitivos" e modos diferentes de pensar o mundo. Isso permite-nos algumas especulações.

Por exemplo, no caso da escrita musical, se é através da representação gráfica dos sons que pode-se manipulá-los, não seria também através dela que eles seriam anteriormente processados? E mesmo no caso da audição interior (ouvido interno), não haveriam categorias apriorísticas definidas, como temperamento, escalas, modos e o "tempo", que novamente nos remetem a um universo simbólico que é dado pela interface?

O computador, assim como todos os dispositivos para produção/reprodução de som, altera radicalmente a relação do homem com o universo sonoro. Pensado enquanto uma interface que se utiliza de uma outra interface (gráfica) para comunicar-se com o usuário, que por sua vez tem o seu próprio arcabouço empírico e teórico, somada às várias tecnologias "indispensáveis" a um estúdio digital contemporâneo, (cabos, impedância, analógico/digital, multicanal, etc), ele acaba, creio eu, nos distanciando mais e mais de um suposto pensamento musical, que deveria interagir criativamente com as várias instâncias de mediação.

No nosso caso, minha hipótese é a de que a interface assumiu uma parte muito maior do que lhe cabe na produção artística.

Como elemento de comparação podemos utilizar a música européia do século X e o barroco do século XVI, e o quanto a notação alterou as suas características. Portanto, os meios de que se dispõe para operar em qualquer área do conhecimento humano alteram radicalmente os modos de operá-lo. Nesse sentido não são ingênuos nem inocentes e atuam mesmo como co-criadores no processo criativo.

Dai a importância dos modelos interativos, desde que eles incluam possibilidades de interagir com outras estruturas de pensamento, principalmente aquelas em que os padrões do cravo-temperado ainda não deixaram suas marcas.

### BIBLIOGRAFIA

- Goody, J. (1988) *Domesticação do pensamento selvagem*. Lisboa. Editorial Presença.  
 Lagrou, Else M. (1992-93) *Caminhos, duplos e Corpos*, PPGAS-USP- São Paulo.  
 Lévi-Strauss, C. (1962) *La pensée sauvage*. Paris: Plon.  
 Lévi-Bruhl, L. *A mentalidade primitiva*. São Paulo. Zahar.  
 Peirce, C.S. (1977) *Semiótica*. São Paulo. Ed. Perspectiva S.A., Estudos.

## MODELOS MATEMÁTICOS E COMPOSIÇÃO ASSISTIDA POR COMPUTADOR, SISTEMAS ESTOCÁSTICOS E SISTEMAS CAÓTICOS

Mikhail Malt  
IRCAM  
mmalt@ircam.fr  
1, Place Igor Stravinsky  
75004 Paris  
France

### ABSTRACT

Este artigo propõe uma primeira reflexão sobre a noção de sistema, as relações entre modelos estocásticos, modelos caóticos e a escrita musical, em CAC, à partir da análise de dois exemplos compositionais: **Actrinou**<sup>1</sup> para piano solo e de **Lambda 3.99**<sup>2</sup> para violão e sintetizador controlado por computador. Estas peças foram formalizadas utilizando-se o paradigma 'sistema', dentro de um contexto de Composição Assistida por Computador. Sendo que, 'Actrinou' usa este paradigma numa visão estocástica, enquanto 'Lambda 3.99' foi composta num contexto de sistema caótico.

### 1. INTRODUÇÃO

No contexto da Composição Assistida por Computador a noção de sistema pode ser um instrumento precioso para a formalização de diversos processos musicais.

A grande maioria das práticas de composição assistida por computador (e de composição em geral) repousa sobre um protocolo<sup>3</sup> bastante regular:

- a) Geração de um material pré-composicional "a-temporal"  
("hors-temps" comme diria Iannis Xenakis)
- b) A articulação deste material inicial e a associação de uma estrutura ordenada,
- c) Definição de funções temporais para percorrer a estrutura "a-temporal" ordenada, de modo a ordená-la "temporalmente".

Este protocolo é nada mais, nada menos que a construção de um sistema à estados discretos.

De outro ponto de vista, inexistente atualmente uma reflexão profunda sobre as relações entre as características conceituais e técnicas de certos modelos e a escrita musical que se origina à partir do uso destes. Especialmente no que diz respeito ao uso de modelos estocásticos e caóticos, a visão de muitos compositores é ainda bastante ingênua, não indo além do uso destes modelos como algoritmos compositionais (**BIDLACK, Rick** (1992)) e da proposição metafórica sugerida pelos nomes dos modelos.

<sup>1</sup>Criada na "Academie d'été" do IRCAM em julho 1993

<sup>2</sup>criada na Tribuna Internacional de Compositores da UNESCO, PARIS 1994

<sup>3</sup>Este protocolo baseia-se na observação das práticas compositionais dos seguintes compositores: Tristan Murail, Gérard Grisey, Claudy Malherbe, Iannis Xenakis, Brian Ferneyhough, Alessandro Melchiorre, e de duas turmas de compositores participantes ao "Cursus d'informatique e Composition Musicale", do IRCAM, no qual participo dando seminários sobre a composição assistida por computador (Para uma familiarização com as práticas compositionais de alguns destes compositores aconselhamos a leitura de **GRISEY**,



## 2. Sistema dinâmico discreto

Definiremos  $S$  como sistema dinâmico à estados discretos como sendo uma entidade possuindo:

- Uma função de transição  $G$  que determina à cada instante o estado assumido pelo sistema, aonde  $G$  é uma função do tempo:

$$G \Rightarrow G(t)$$

e  
- um conjunto finito  $E$  dos estados possíveis que pode assumir o sistema, aonde  $e_t$  designa o estado assumido à cada instante "t" calculado à partir da função  $G$ .

$$E = \{e_0, e_1, e_2, e_3, \dots, e_n\}$$

Cada estado  $e_i$ , sendo uma classe de equivalência, pode por sua vez ser um sistema, com seus próprios estados internos e suas funções de transição. Um sistema finito à estados discretos  $S$  será então definido como sendo o conjunto formado por uma função de transição  $G(t)$  e um conjunto de estados  $E$ :

$$S = \{E, G(t)\}$$

## 3. Transição entre o formal e o musical

Musicalmente poderemos interpretar a formalização acima como sendo parte de um protocolo composicional<sup>4</sup>:

a) Inicialmente o compositor gera o seu material pré-composicional.

b) Como segunda fase o compositor organiza este material segundo vários parâmetros perceptivos e/ou lógicos. Nesta fase o compositor discretizou o seu material pré-composicional e associou-o à uma, ou várias estruturas ordenadas, criando o conjunto  $E$  dos estados possíveis que poderá assumir o sistema.

c) Finalmente o compositor vai construir caminhos através deste material para gerar um processo musical, o que significa que o compositor irá definir uma função de transição  $G$  que determinará como percorrer o conjunto dos estados criados.

O último item é de grande importância pois revela uma das grandes preocupações do compositor, que é o de definir caminhos, direções que serão tomadas pelo material. Esta preocupação se faz sentir ao nível da escrita musical em compositores como Tristan Murail, que calcula com muita precisão vários processos, altamente direcionais, que seguirá seu material; e de Brian Ferneyhough que constrói suas matrizes de estruturas altamente hierarquizadas, que ele percorre segundo algoritmos que lhe são próprios. Num caso como no outro estes processos nunca são deixados à vista, as pistas são apagadas, seja pela introdução de ruído (num sentido estocástico) ou seja pela quebra das regularidades calculadas.

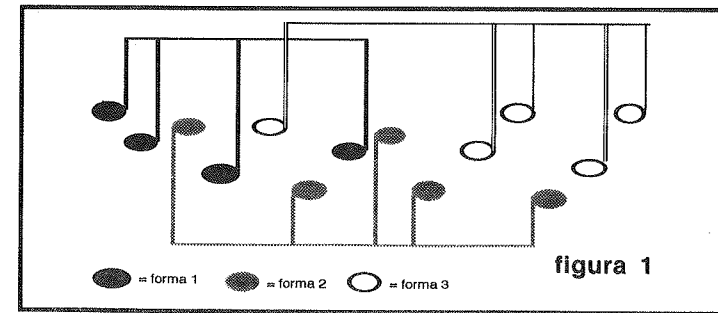
<sup>4</sup>Gostaria de lembrar que muitas destas reflexões podem ser aplicadas em casos particulares da composição mais tradicional, mas que estamos estudando este protocolo dentro de um contexto de C.A.C., isto é, composição assistida por computador, o que faz com que o roteiro da elaboração composicional tome rumos particulares. Além do que, deixamos de lado, voluntariamente, uma primeira fase que seria de elaboração conceitual, para nos concentrarmos sobre a fase diretamente ligada à manipulação do material com a ajuda do computador. Em vista destes aspectos gostaria de convidar o leitor a ler os artigos de DUFOUT, Hugues (1981) e BOULEZ, Pierre (1981) sobre a influência do computador no pensamento composicional.

## 4. Dois exemplos de aplicação

Ilustraremos à seguir os conceitos mais importantes na composição e formalização das duas peças. Mas é necessário lembrar que as indicações que daremos aqui não são uma análise detalhada de cada peça, mas indicações gerais que podem nos ajudar a compreender o uso do conceito abstrato de sistema em composição musical.

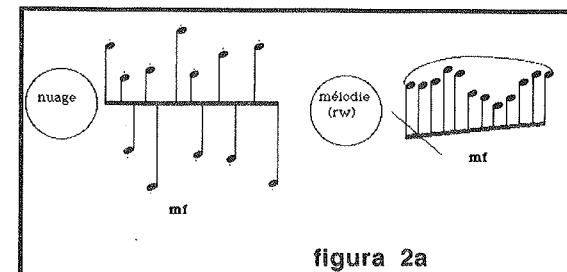
### 4.1. Os conceitos musicais

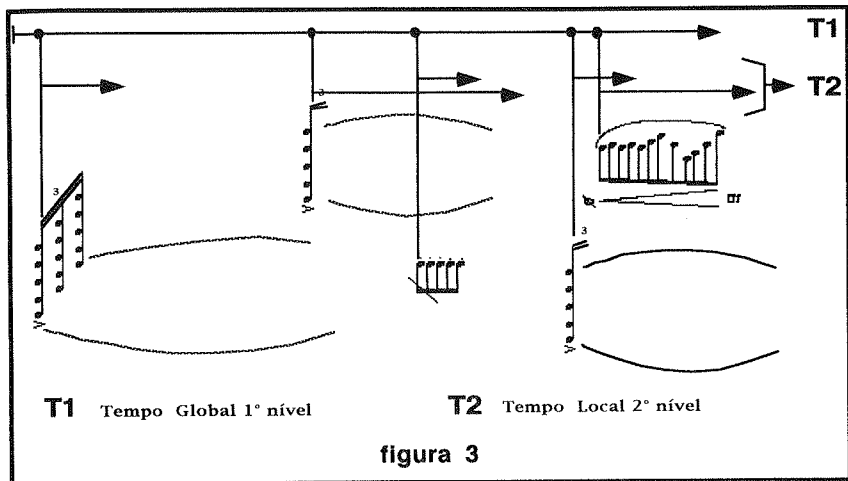
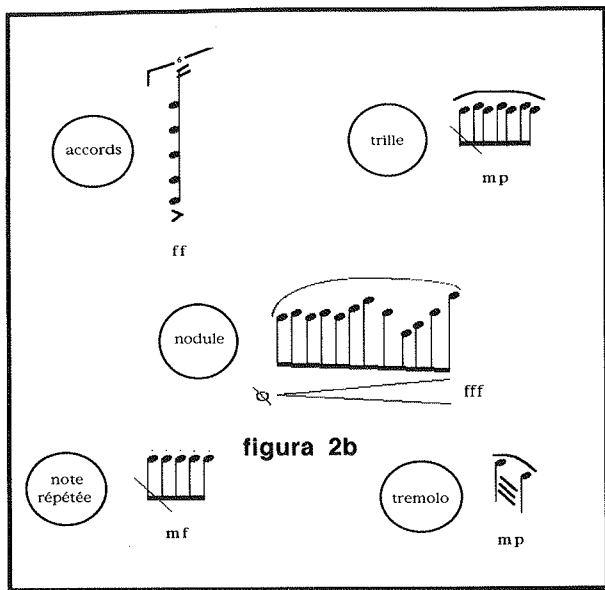
Um dos conceitos motores que guiou a composição de Actrinou e de Lambda 3.99 foi o uso da idéia de "polifonia virtual". Este conceito amplamente conhecido de todos os músicos é o que regeu a composição das suítes e sonatas de J. S. Bach. A idéia de base era de aplicar este conceito não somente à notas, mas à estruturas mais elaboradas, de modo que à partir de um jogo estritamente sequencial pudessemos dar a ilusão auditiva de uma polifonia de fluxos, ou seja uma polifonia de estruturas. Esta questão dos fluxos auditivos foi estudada exaustivamente por Mc ADAMS, Steve et A. Bregman (1987)-



Para podermos concretizar este conceito precisamos construir microformas (ou gestos) que pudessem ser reconhecidos como entidades autônomas e diferenciadas. Em função destas restrições cada gesto foi idealizado de maneira a ter suas características próprias (veja como exemplo as figuras 2a e 2b):

- uma morfologia de frequências,
- uma articulação,
- uma dinâmica,
- uma estrutura temporal local
- uma textura própria que decorre da conjunção dos parâmetros anteriores.





Desta primeira formalização poderemos deduzir a existência de duas camadas temporais superpostas (ver figura 3):

- a) Um tempo local, definido pelas características estruturais de cada gesto, e
- b) um tempo global que gera a evolução das nossas microformas

Além deste conceitos gerais, existem ainda outros que geram o detalhe dos campos harmônicos e do ritmo, cuja a análise não será abordada neste artigo.

4.2. Os conceitos formais

Duas idéias formam a base para a formalização destas duas peças:  
 a) É possível representar a evolução de um processo musical por um sistema dinâmico.  
 b) Todo processo musical pode ser analisado como sendo a evolução de várias formas, independentes, tendo cada uma destas uma evolução própria.

No nosso caso isso significa, definir um conjunto **E** dos estados possíveis e definir uma função **G (t)**.

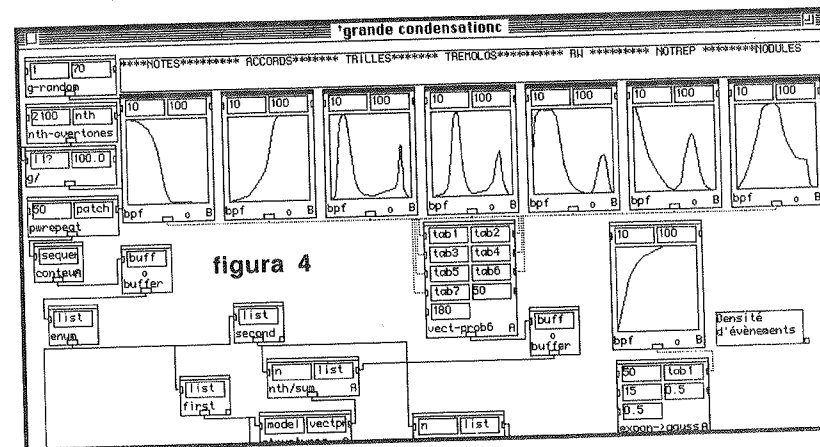
4.3. Actrinou

Para Actrinou nosso conjunto **E** era constituído de sete gestos de base (veja figura 2). A única transformação que cada gesto poderia sofrer era uma simples translação no espaço das frequências (transposição). A nossa função **G (t)** (estocástica) era função de um vetor de probabilidade calculado à partir de sete outras funções (determinadas graficamente) que indicavam a cada instante "t" a probabilidade do sistema assumir um estado em particular. Neste caso particular tínhamos:

$$G(t) = [f_{\text{notes}}(t), f_{\text{acordes}}(t), f_{\text{trillo}}(t), f_{\text{tremolo}}(t), f_{RW}(t), f_{\text{notrep}}(t), f_{\text{nodule}}(t)]$$

onde cada função  $f_{\text{gesto}}(t)$  é definida graficamente (veja as curvas gráficas na figura 4). A figura 4 representa uma janela do programa gráfico "Patchwork"<sup>5</sup>, na qual foram calculadas as diferentes secções da peça.

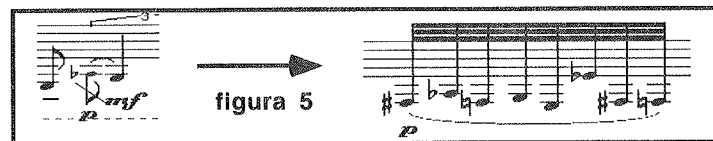
O uso destas funções auxiliares se mostrou inevitável, pois uma das propriedades de base dos modelos estocásticos mais simples (excluindo os modelos markovianos) é a ausência de memória. Do ponto de vista composicional esta "amnésia" era perturbadora, pois não permitia um controle do processo. A introdução destas funções auxiliares permitiu, pelo menos à um nível global um maior controle sobre a evolução de cada gesto, simulando uma memória (normalmente ausente) pela variação, no tempo, das densidades de probabilidade de cada gesto. Para gerar o tempo global da peça usamos uma interpolação entre um tempo gerado por uma distribuição exponencial de densidade baixa (aproximadamente .5 eventos por unidade de tempo) e uma distribuição gaussiana de média de 150 milésimos de segundo. Este tipo de interpolação entre modelos permitiu também de gerar o aspecto escrita musical de uma maneira mais flexível.



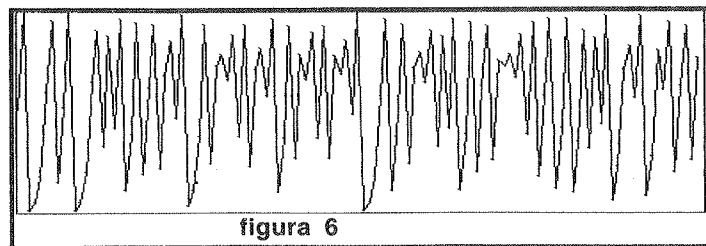
<sup>5</sup>O programma Patchork é um programa, desenvolvido no IRCAM, destinado à composição assistida por computador. Foi idealizado pelo finlandês Mikael Laurson e sua versão atual foi

## 4.4. lambda3.99

Basicamente, **lambda3.99** segue o mesmo processo de **Actrinou**. Inicialmente definimos o conjunto **E** à partir, não mais de gestos estáticos, mas à partir de classes de equivalência. Por exemplo um dos estados possíveis era o que chamamos de "apogiatura", este estado continha um conjunto de gestos ordenados evoluindo de uma "apogiatura" simples à uma nuvem estocástica em torno de uma nota central (ver figura 5):



Em seguida, um ponto importante era a escolha de uma função de transição que tivesse propriedades interessantes do ponto de vista musical. Uma propriedade que nos interessou nos sistemas caóticos são certos conjuntos de valores gerados por soluções numéricas de equações diferenciais, chamados atratores estranhos (ver Ruelle D. (1980) in CVITANOVIC, Predrag (1989)), como o gráfico da figura 6. Uma das propriedades destes conjuntos de valores é a de possuir simetrias internas (PEITGEN, H. O. ; JÜRGENS, Harmut; SAUPE, Dietmar (1993) & CVITANOVIC, Predrag (1989)-), mas de uma maneira que lembra a escrita musical: uma oscilação entre memória e informação. Não discutiremos aqui do porquê destas simetrias, visto que este sujeito se acha explicado detalhadamente na bibliografia citada.



Um conjunto de valores como os da figura 6 acima tem um certo interesse, mas se quisermos orientar o nosso processo deveremos transformá-lo, achar uma estratégia que possa dar-lhe uma direção; por esta razão utilizamos uma função da forma:

$$G(t) = \sigma(t)F(t) + \alpha(t)$$

aonde<sup>6</sup>,

$$F(t) = F(t-1)\lambda[1-F(t-1)]$$

$$F(0) = 0.512$$

$$\lambda = 3.99$$

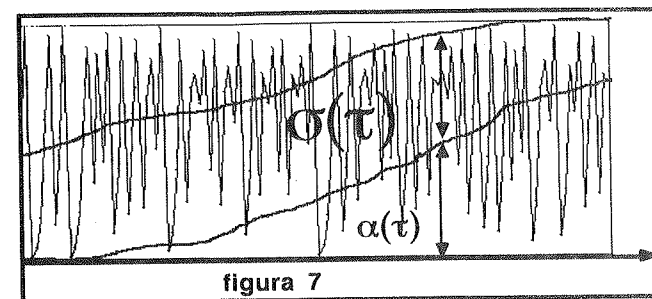
$\sigma(t)$  = fator de contração

$\alpha(t)$  = fator de deslocamento

<sup>6</sup>A equação recursiva  $F(t)$  é conhecida também como equação logística.

<sup>7</sup> $F(0)$ -valor inicial para as interações de  $F(t)$ , e  $\lambda$  é o coeficiente de turbulência da

isto-é aplicaremos uma transformação, localmente linear, sobre este conjunto de maneira a podermos dar uma direção à este processo e manter boa parte das simetrias de origem:



Este mesmo procedimento será então aplicado à cada classe de equivalência, de maneira a que durante o processo teremos estados que evoluirão não só por translação mas que assumirão diferentes morfologias dentro de uma lógica de proximidade perceptiva.

## 5. Conclusões

O paradigma sistema poderá ser um bom modelo para formalizar vários processos em composição musical assistida por computador, pois permite uma grande flexibilidade na escolha do material pré-composicional (construção do conjunto **E**), e na escolha da função de transição **G(t)**. A definição da função de transição é um dos pontos vitais deste paradigma, pois é esta função que determinará o desenvolvimento do processo modelizado. Como consequência imediata esta mesma função será portadora de informações sobre a forma do processo, na medida que entendermos que a evolução dinâmica dos elementos musicais participa naquilo que chamamos "forma". E, finalmente poderemos dizer que um uso consequente da composição musical assistida por computador deverá passar, obrigatoriamente, por um desenvolvimento de um solfejo de modelos, que permitirá de fazer a ligação entre as características dos modelos e as suas potencialidades musicais.

## 6. Referências

1. AMES, Charles (1991) - "A Catalog of Statistical Distributions: Techniques for Transforming Random, Determinate and Chaotic Sequences" in *Leonardo Music Journal*, vol 1, n° 1, pp. 55-70.
2. BARNESLEY, Michael (1988) - *Fractals Everywhere*, Academic Press, Inc.
3. BIDLACK, Rick (1992) - "Chaotic Systems as Simple (but complex) Compositional Algorithms", in *Computer Music Journal*, vol 16, n° 3, Fall 1992.
4. BOULEZ, Pierre (1981) - "L'in(dé)fini et l'instant", in *Le compositeur et L'ordinateur*, IRCAM, Paris.
5. CVITANOVIC, Predrag (1989) - *Universality in Chaos*, Adam Hilger, Grande -Bretagne.
6. DELATTRE P. (1985), *Systèmes de transformation*, in *Encyclopaedia Universalis*, France S.A.
7. DELATTRE, P. (1976), *Langage interdisciplinaire et Théorie des Systèmes*, in *Structure et*

8. DODGE, C. and JERSE, T. A. (1985), *Computer Music*, Schirmer Books N.Y. .
9. DUFOURT, Hugues (1981)- "Les difficultés d'une prise de conscience théorique", in *Le compositeur et L'ordinateur*, IRCAM, Paris.
10. GRISEY, G. (1989), "Tempus Ex Machina", in *Entretiens* n° 8, Paris, France.
11. LORRAIN, D. (1980), *Une Panoplie de Canons Stochastiques*, Rapport IRCAM-n° 30, Paris.
12. MALT, Mikhail (1991)- *Trois aspect de formalisation dans Achorrispsis de Iannis Xenakis* . Mémoire de D.E.A., sob a orientação de Hugues Dufourt, EHSS-Ecole des Hautes Études en Sciences Sociales et IRCAM-Institut de Recherches et Coordination Acoustique Musique.
13. MALT, Mikhail (1992)- *PW-Alca-librairie de Modèles stochastiques*, IRCAM, Paris, France.
14. MALT, Mikhail (1993)- *Introduction à Patchwork* , IRCAM, Paris, France.
15. MALT, Mikhail (1994)- *Chaos-librairie de modèles chaotiques et de fractales* , IRCAM, Paris, France.
16. Mc ADAMS, Steve et A. Bregman (1987)- "L'audition des flux musicaux", in *Marsyas*, Institut de pédagogie musicale et chorégraphique, La Villette, Paris (3-4) décembre 1987, PP 97-118.
17. MURAIL T. (1989) - "Questions de cible " in *Entretiens* n° 8, Paris, France.
18. PEITGEN, H. O. ; JÜRGENS, Harmut; SAUPE, Dietmar (1993)- *Chaos and Fractals*, *New Frontiers of Science*, Springer-Verlag, New York.
19. WALLISER, Bernard (1977)- *Systèmes et Modèles*, Editions du Seuil, Paris.
20. XENAKIS, I. (1976), *Musique Architecture*, Casterman, Paris
21. XENAKIS, Iannis (1981) - *Musiques Formelles*, Stock Musique, Paris.

## 7. Agradecimentos

Escrevo este artigo como bolsista do CNPq na EHSS-IRCAM-Paris, para a realização de um Doutorado em Música e musicologia do século XX sobre "Modelos Matemáticos e Composição Assistida por Computador" sob a direção de Hugues Dufourt e Jean Baptiste Barrière.

## Synthesizing Music with Sampled Sound

YEBON LO ([muse@leland.stanford.edu](mailto:muse@leland.stanford.edu)) and DAN HITT ([hitt@cs.stanford.edu](mailto:hitt@cs.stanford.edu))  
 Center for Computer Research in Music and Acoustics  
 Stanford University  
 Stanford, California 94305  
 USA

### Abstract

Sampled sounds are now an important resource for modern music-making and multi-media events. But except for certain classes of sampled sounds, digital synthesis methods have to be crafted to exploit them fully. A method involving a certain form of analysis is described here. Some results are presented and its musical applications discussed.

## 1 Emergence of Sampled Sounds

Sampled sounds are now an important resource for modern music-making and multi-media events. As magnetic storage costs less than \$1 (US) per megabyte and gets cheaper all the time and as flash memory becomes increasingly available, the advantage of sampled sounds in music computing is obvious. For example, there is much less compelling reason to use an algorithm to generate, or synthesize, plucked-string sounds when the latter are readily available from a CD-ROM or hard-disk memory.

Using samples not only speeds up the run-time process, thus making real-time performance attainable after a certain threshold in hardware speed is crossed, but also saves the user development cost, because the operation becomes as simple as file I/O management instead of coding and debugging a piece of numerical calculation in a typically larger and more complex music computing environment (which combines and manipulates samples)<sup>1</sup>. In other words, the space advantage of algorithmic synthesis is now overshadowed by time considerations.

So it seems that samples are replacing synthesis at least where acoustic instrument timbres are concerned. And indeed one might argue that any sound of nonacoustic origin may be similarly made available as samples at the factory, saving user cpu as well as development time as discussed above<sup>2</sup>.

## 2 Potential and Controversy

Now by means of widely available sequencing software, one can easily explore combining samples with control over choice of sound material, amplitude, timing, spatial movement and even reverberation. Therefore it is not surprising that some see samples to replace the orchestra soon if not already. Surprising, however, is that not everyone shares this bright outlook and there are those who are just as vehement in believing that even from a purely musical standpoint, sampled sounds (with all the help they can get from the computer) will never replace the orchestra!

Thus the questions are: Do we still need synthesis in a widely applicable sense? That is, do we still need synthesis if we are only interested in making music from available sampled sounds? If we do, why? And what form of synthesis do we need?

To answer these questions, it might be profitable to examine some of the issues pertaining to working with sampled sounds. Those who believe in the role of the orchestra or live acoustic ensemble more or less put their money where their ears are. Most trained musicians who rely on their ears to do their business will say the music written for an orchestra and realized by sampled sounds are "second-rate" at best. (So far we haven't examined the source of this less-than-second-ratedness.)

<sup>1</sup>To be sure, we might still want to filter the samples, detune them, shape them and "warp" them in all manner one can imagine, superpose them and sequence them, but these would be necessary additional operations anyway in the music synthesis paradigm whether samples are used or algorithm are invoked. The advantage of sampled sound in this instance illustrates the advantage of a whole class of sounds which are point-excited in origin—contributing to the popularity of drum machines.

<sup>2</sup>Here synthesis is used in the traditional sense of the word in sound synthesis by digital computer: the generation of a sequence of numbers that approximates the waveform to be heard (excluding a transformation for scaling—within the limits of linearity—which is nothing more than turning a dial on your amplifier in the analog domain).

On the other hand, those who are educated in the language of information theory and computer technology see a different perspective, based on a string of existential and counting arguments: Surely the Sampling Theorem of Nyquist and Shannon guarantees that every vibration in the air that our ears can catch the computer can duplicate with arbitrarily high fidelity. So if we sample as often as is required by the rate of the most stringent fluctuation in the acoustic signal and resolve each sample value into a number over the range required by the dynamic range of the signal and simultaneously minimize the noise from quantization (which, although it cannot be completely removed, is generally not a musical objection as far as concert music is concerned), then all the fine details in the orchestral performance will indeed be faithfully captured. Thus, at least at a single spatial point (or a finite collection of them), a piece of orchestral music is simply a finite sequence of numbers having finite resolution—i.e., equivalent to a finite sequence of integers. Bounding the length of the piece (say to require it to be less than one hour) bounds the number of such sequences, and gives a finite space of integer sequences to search through for any desired orchestral piece (of duration less than one hour).

It is, however, a leap of faith to suggest that a large enough cascade or assembly of oscillators will approximate arbitrarily closely the musical waveform if a few of them have been demonstrated to do so to some degree of success with a certain (sub-)class of sounds. In this case, there is no mathematical theorem concerning the procedure of *synthesis* (as opposed to counting). The Fourier theorem has stringent assumptions. When we stretch the domain of operation to suit these assumptions, i.e., take the Fourier Transform of the entire piece so as to avoid the periodicity restriction, we lose complete control over the procedure of manipulating the parts (the samples) that form the piece. This is a problem because our ears make use of time domain information as well as frequency domain information; in fact, music is traditionally written, performed, and listened to in the time domain. When we "violate" the Fourier premises by using time-varying approaches, we encounter all kinds of artifacts, as well as having to wrestle with tedious computations and grapple with precision of control. In short, there is *no* theorem that will *guarantee* the existence of a recipe which would produce a digital waveform for a score that would closely approximate the digitized copy of an orchestral rendition of that score.

### 3 A Fundamental Practical Constraint

So, the enumeration and existence arguments—that the set of all digital waveforms forms a superset of acoustic waveforms (and hence include every single orchestral piece ever written or that could be written or that are physically realizable)—ignore the intricate process where by the overall waveform is eventually arrived at. The enumeration arguments are simultaneously at odds with the combinatorial explosion problem. These arguments, after all, apply even more strongly to writing prose, for example, but authors have not been replaced. The moral is that a *finite* search space is not necessarily *small*, at least from a human perspective. On the other hand, we have a small alphabet from which to build a rich written vocabulary (and thus a literature). And we have a small set of phonemes to build a rich and flexible spoken vocabulary (and thus can communicate). So it is quite desirable, from a computer-science standpoint, to have a library of elemental objects (such as the sampled sounds) from which to build complex music from—if we ever hope to have a satisfactory solution to making music with sampled sounds.

It might be easy to dismiss our current lack of success in realizing an orchestral score via sampled sounds as a matter of incompatible paradigms—the methods of generating orchestral and computer music being too far apart. But one doubts this view satisfies most who are dedicated to maximizing the utility of a modern digital computer and who are well aware of the expressiveness and power possible from a computer (theoretically, and, in other areas, practically). The machine's current lack of eloquence reflects our own (current) limitations.

An alternative might be to explore the sources of obstacles that give rise to that second-rate quality when a score is realized with sampled sounds: identify the problems and search for solutions.

To do this, we will first take a step back to examine an important musical application of sampled sounds: interactive performance. Through it, we will soon find out some fundamental problems that hinders their utility for making superior-quality music. And then we will proceed to suggest solutions and present some preliminary results.

## 4 Interactive Performance: Can a machine imitate?

A classical approach to music making is the method of imitation: canon, ricercar, fugue, or almost any polyphonic writing, especially ensemble pieces.

Can a machine imitate? How well can it? Especially with sampled sounds.

Without loss of generality, let us use the model depicted in the diagram to discuss a situation in interactive performance—an important application of the computer to music. Let's suppose that the live part consists of a clarinet and a viola. Now suppose that the clarinet begins the music with a lyrical solo passage and we expect the machine to ease in with a polyphonic but homogeneous texture. (We are looking for a gentle build-up in the musical activity.) Such a development naturally calls for a similarly lyrical line from the machine, ideally voiced by a similar, i.e., clarinet, timbre. Now, if the lyricism of the live performer involves slurring of notes into long and short phrases as is most often expected, the polyphony would make the most sense if the machine counterpoint is also slurred.

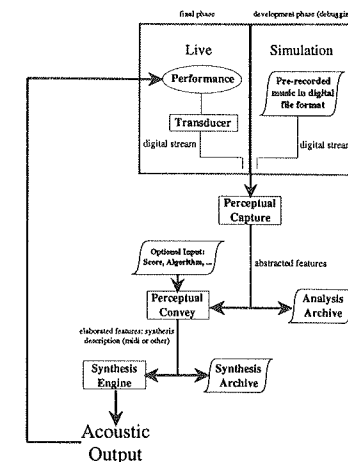
We will not consider the method of generating the counterpoint here (which is by no means trivial, although it can certainly be achieved to varying degrees of success). The issue to discuss here is articulation. Those of us who have worked with sampled sounds are aware that machines don't slur very well. And that is because despite the fact that a wealth of information is transmitted from the musicianship of a skilled performer to the acoustic signal, all a machine (a commercial synthesizer or otherwise) can do is cross-fade, which conveys practically zero information. This is an obstacle not only to using sampled sound to jam with the clarinet, but also to jamming with the viola, a bowed-string instrument.

This ability to slur, or do other kinds of articulation over a group of notes, which contributes so much to the music of any acoustic instrument playing, and which is such a measure of musicianship of the performers, is lacking in current applications of sampled sounds with computer [1]. The lack which is indeed a major stumbling block to replacing the orchestra (with sampled sounds) is due to a fundamental limitation in our understanding of signal behavior when notes are joined, i.e., when samples of different pitches but similar timbres, or different pitches and different timbres, or even different dynamics, are joined. Although one can avoid the issue and create another kind of work, the limited machine expressivity means the available "tricks" may be exhausted before long and adequate jamming or imitating using such classically versatile techniques as transposing, inverting, retrograding, or permuting as Bach might have done, is not practicable. (The emphasis is on the word *adequately* here; a preprogrammed or even spontaneous note list whose performance is not musical or bears only a poor relation to the performance of the live player is not really adequate.) Hence until workable methods for slurring are found (which may in the worst case differ from sample pair to sample pair), it is safe to say that the computer will not replace the orchestra (nor will sampled sounds).

## 5 A New Kind of Synthesis

The slurring or transition problem cannot be solved by recording and storing tables of transitions; this runs into the "combinatorial explosion" problem so familiar from computer science. And we still have the problem of joining the steady-state sampled regions with the transitions. This means we have to solve the problem of joining sampled sounds, and points to the need for a more sophisticated synthesis than we have commonly available.

Here, by synthesis, we mean the generation of the sequence of numbers which describes a waveform or a section of a waveform—such as the transitions between two samples. In this case, the samples near the transitions may also be modified in order that the connection be smooth.



In order for the synthesis to be capable of creating smooth transitions between sampled events, so as to achieve greater intimacy of interaction with live performers and to possess a greater degree of coherency, analysis must play an important role in this kind of synthesis. For example, if we need to create a transition between clarinet tones  $\alpha$  and  $\beta$ , we must somehow dig into the data present in  $\alpha$  and  $\beta$ , and elicit its essence.

The question now shifts to *what* kind of analysis is suitable for analysis-based synthesis.

## 6 Perceptual Computing, and "Capture and Convey"

Analysis has value in and of itself; it is, after all, the primary intellectual activity.

The analysis that we propose should be performed on sounds, however, has a particular goal: to enable a suitably flexible synthesis for musical purposes.

This suggests that our analysis should be "receiver-based": i.e., it should seek out attributes of the signal which receivers (humans) find significant. Further, it should guarantee some sort of continuity or smoothness condition in the other (auditory) perceptual attributes as a particular one is varied, in the synthesis of a class of sounds. In short, the analysis of a particular perceptual attribute should be able to express its essentials in the context of other perceptual attributes. For example, a computation which helps to enable us to take the pitch contour of a phrase with one timbre and re-perform it with another timbre would meet this criterion. A computation which purports to abstract a pitch contour but from which we cannot develop or reconstruct a phrase (e.g., to make a satisfactory slur) would not meet this criterion, and in fact would not be nearly as useful. A second example would be a process enabling one to take a pitch contour in a given timbre and transpose it to another pitch height without significantly distorting the timbre—we might refer to this as *capture and convey*, and we maintain that we haven't really captured anything if we cannot convey it (i.e., in this case, transpose it without distortion). We can formalize conveyability and how we intend to use it:

**Definition.** A perceptual structure  $\sigma$  is *conveyable* ( $\chi$ ) (over a sound set  $S$  with respect to pitch) if

$$\|P_{\sigma}(x) - P_{\sigma}(x')\| \ll \|P_{\pi}(x) - P_{\pi}(x')\| \quad x, x' \in S$$

Here,  $P_{\pi}$  is the projection onto the pitch dimension, i.e.,  $P_{\pi}(x)$  is the pitch of  $x$ , and  $P_{\sigma}$  is the projection onto the parameter space of  $\sigma$ .

**Remark.** We cast conveyability in these terms because often changing pitch necessarily changes  $\sigma$  (which might be one of the "qualities" of timbre, for example, to what extent and in what way it sounds like a violin); we only demand that the change in  $\sigma$  be small compared to the change in pitch.

**Definition.** A perceptual structure  $\sigma$  is *capturable* ( $\kappa$ ) weakly for a certain sound if that sound can be resynthesized perceptually identically with respect to  $\sigma$ . It is *capturable strongly* if as well it is conveyable.

**Remark.** The intention is that we've captured a particular timbre weakly if we can resynthesize it, and we've captured it strongly if we can resynthesize it along some range of pitches.

These considerations—in particular the notion of capture and convey—provide us with a broad measure of the success of our analysis and synthesis as well as measuring how resonant our methods are with sampled sounds.

## 7 A Particular Case: the Kinematic Method

Here, we report on a particular kind of analysis-based synthesis which we are using and trying to further develop to deal with the issues mentioned previously: joining sound events, capture and convey, etc.

We call the method *kinematic synthesis* because it is modeled on certain entities moving (hence "kinematic") through a suitable vector space (see [2], [3], and [6] for additional information on the kinematic method).

Briefly, the method models musical sounds as consisting of states and transitions between them, where the time scale of a state is comparable in some cases to the time scale of frames in a motion picture; in fact, we call the "periods" of a sound "frames". We note of course that no musically interesting sound is periodic, but the waveform of every musically interesting sound passes through stages with a great deal

of redundancy involved. One goal of kinematic synthesis is to remove the redundancy from the analysis data—but not remove the "content".

More exactly, the method postulates that a sound can be perceptually recaptured from a certain set of analysis data, a *triple*, consisting of a *frame trajectory*, an *amplitude envelope*, and a *period trajectory*. The use of the term "trajectory" is to suggest motion. The frame trajectory does not consist of all the frames (as we've used the term above) in the sound, but principle or key frames (*break frames*, by analogy with break points in a piecewise linear or piecewise smooth curve) from which much of the sound can be recovered (some might call it spectral data, which is approximately true). The amplitude envelope maps out the overall changes in amplitude, and can be cast in many forms (including as a pair of functions: a lower and upper envelope). There is actually a great deal to be said about the amplitude envelope (as there is about the other two members of a triple), but one heuristic guide to its construction is that it reflect the smoothness and evolution of the sound. The period trajectory maps how the "micropitch" varies (and in many cases is nearly flat) as measured by the changes in the motion of the frames through the space.

The analysis data (*triples*—frame trajectories, period trajectories, and amplitude envelopes) is directly interpretable from a perceptual view.

The method tries to exploit the mass of sampled data we find ourselves surrounded with, but tries to cut redundancy on a dynamic basis (so in a very "innovative" part of a sound, say the attack, more frames—more data—will be used, as well perhaps having a more densely specified amplitude envelope). If we cast the waveform as a series of frames—points in some high-dimensional vector space—it forms a curve, which we sample more densely where it is most dynamic (e.g., where the curvature is greater). Our data reduction is typically in excess of 90% just on resynthesis (capture) using the break frame notion—and we lose no phase spectrum data (on the break frames).

The method has had some success in capture and convey; we've conveyed flute tones across an octave, and a violin tone across three octaves with variable (and arbitrary) durations. We've also conveyed a slurred two-note phrase from the flute in the pitch dimension, and have created timbre melodies involving the violin and the flute. If we convey a sound over  $M$  pitches, then of course the data reduction is  $M$  times as large.

## 8 Applications

Kinematic synthesis, or other comparable methods, have at least four important uses in musical constructions involving sampled sounds:

1. Transposition, retrogression, inversion, and other continuous line formations. The first three of these are self explanatory (and of course, when we talk of, say, inversion, inverting the note list is the tiniest part of the effect we want to produce: we want to invert a line with articulation and phrasing suitably related to the original). By "other continuous line formations" we refer to effects on continuously changing sounds, such as slowing or other articulation.
2. Timbre substitution. By this we mean taking a line and holding its pitch trajectory constant while varying other components of the timbre (in the simplest case, this might refer to say, replacing a clarinet line with a trumpet line—musically).
3. Analogically reasoned transformations. In a simple case, this might refer to going through a curve, seeking a feature sequence  $\alpha\beta$  where  $\alpha$  stands in relation to some feature  $A$  in a template in the same way the  $\beta$  stands in relation to feature  $B$  in the template, and then replacing  $\beta$  by a  $\tau$  (from a  $T$  in the template). See [3] regarding this approach to sound modification-synthesis.
4. Restoration of data from corruption. A very good analysis-based synthesis paradigm should be able to go through corrupted data—say captured from an old vinyl record—and create a digital stream perceptually identical to the original. (We don't claim to have done this on any large scale.)

The greatest interest one has in these applications is of course in the case of continuously sustained sounds. An ideal computing environment to carry out this kind of analysis-based synthesis is an open-ended (extensible) object model with a "polyhedral" architecture where multi-window, multi-document

applications form the vertices of the polyhedron, and can communicate with each other across the edges, under user control; mixing, editing, synthesis, and analysis can be done concurrently in full view of the user ([5], [6]).

## 9 Concluding Remarks

1. Certain classes of sampled sounds are directly, immediately, and readily usable for the purposes of composition, e.g., point- (impulse-) excited sounds such as plucked string (pizzicato), drum, and piano (to some extent). Other sounds are useful for special occurrences but not for smooth melodic definition.
2. In order to widen the applicability of sampled sounds, the research community needs to direct its efforts to finding solutions to make the non-impulse-excited sounds readily usable for forming musical constructs. A synthesis model is not as useful if its demonstrable application is limited to plucked or percussive sounds. Likewise, a synthesis model which can take two sampled sounds and join them in a variety of expressive two-note phrases is more useful than a model which merely duplicates an existing sampled sound.
3. We must recognize that making a lyrical melody via digital means entails more than just cross-fading two sounds. It is essential to accept that in most cases, creating transitions requires knowledge, either in the form of data from the signal, or in equivalent algorithms to generate the data.
4. There is more than one possible transition between two tones, whether the timbres are the same or different. That is, a multiplicity of possible trajectories exists. These constitute a repertoire of articulations. The ability to display them digitally is a demonstrates the expressivity of a given composition/performance environment.
5. To choose a suitable transition trajectory (in order to maximize some local coherence criterion in composition), one needs suitable analytical tools. This is important in both a composition as well as a performance environment. The relevant tools might include some to decide which transition trajectory is being executed by the performer and provide the responding algorithm the best or most accurate information (regarding what actually happened) and allow it to make the best choice under a given composition strategy. They might also include some to analyze the pitch trajectory so as to perform a retrograde, transposition, inversion, etc., in a musically responsive way.
6. One synthesis candidate that is naturally suited for joining (or connecting) sampled sounds into articulated melodic constructs is kinematic synthesis, where the basic unit of the method is the triple: the amplitude envelope (possibly expressed as upper and lower sub-envelopes), the pitch trajectory or equivalent, and the trajectory of critical frames (which are periods in a steady state, and something more general in transient regions).

## References

- [1] Strawn, J. (1985) *Modeling Musical Transitions* (Ph.D. Thesis), Technical Report STAN-M-26.
- [2] Lo, Y. (1986) "A Technique for Timbre Interpolation" *Proceedings of the ICMC* 241-248.
- [3] Lo, Y. (1987) *Toward a Theory of Timbre* (Ph.D. Thesis), Technical Report STAN-M-42.
- [4] Hitt, D. & Lo, Y. (1990) "L: A Language for Composition", *Proceedings of the ICMC* 237-240.
- [5] Hitt, D & Lo, Y. (1992) "An Alternative Digital Environment for Music Synthesis" *Proceedings of the Delphi Computer Music Conference/Festival* [not paginated].
- [6] Lo, Y. & Hitt, D. (1992) "Uniform Treatment of Sounds and their Syntheses on Digital Computers" *1992 International Workshop on Models and Representations of Musical Signals, Capri, Italy* [not paginated].

## Um Ambiente de Auxílio a Composição Musical

ALEXANDRE JONATAN BERTOLI MARTINS  
ANDRÉ LUIZ COSTA BALLISTA

*Instituto de Informática - CPGCC*  
*Universidade Federal do Rio Grande do Sul*  
*Campus do Vale - Bloco IV*  
*Av. Bento Gonçalves, 15064, CEP 91501 - 970*  
*Porto Alegre - Rio Grande do Sul*  
*fone: +55 (051) 336-8399 - Ramal 6161*  
*e-mail: NATAN@INF.UFRGS.BR e BALLISTA@INF.UFRGS.BR*

MARCELO SOARES PIMENTA

*Departamento de Informática e Estatística*  
*Universidade Federal de Santa Catarina*  
*Campus Universitário, Trindade, CEP 88049-900*  
*Florianópolis - Santa Catarina*  
*fone: +55(0482) 319739*  
*e-mail: CEC1MSP@BRUFSC.BITNET*

### Abstract:

*Este artigo apresenta um ambiente de auxílio a composição musical desenvolvido em Smalltalk V/286. Ele provê ao compositor meios de criar e manipular objetos sonoros, os quais podem ser testados interativamente através da utilização de sintetizadores acoplados ao ambiente. Para este sistema não há distinção entre uma nota musical e uma melodia completa: ambos são objetos sonoros e são tratados da mesma forma. O sistema foi projetado e implementado de forma orientada a objetos e o ambiente utiliza o mesmo paradigma para sua interação com o usuário.*

### 1. Introdução

As aplicações músico-computacionais têm se destinado a atividades bastante diversas. Aquisição e reprodução de performances, ensino e treinamento prático e teórico da música, geração automática de melodias e auxílio a composição musical são alguns exemplos.

Neste artigo, trataremos especificamente da utilização de computadores na composição musical. Como compor é um processo criativo, é necessário um ambiente que proporcione ao usuário ferramentas que o auxiliem durante todo este processo. Por sua vez, estas ferramentas devem ser projetadas especificamente para o processo de composição, de forma que o compositor não seja obrigado a adotar uma metodologia de criação para que possa utilizar tais ferramentas.

Apresentaremos aqui um ambiente de auxílio a composição musical denominado CAMC [2], que é um ambiente orientado a objetos desenvolvido em Smalltalk V/286 para suportar o processo de composição musical. Ele provê ao compositor meios de criar e manipular trechos musicais, os quais podem ser testados interativamente através da utilização de sintetizadores acoplados ao ambiente. Para este sistema não há distinção entre uma nota musical e uma melodia completa: ambos são objetos sonoros e são manipulados da mesma forma.

Para a representação da música utilizamos o modelo Smallmusic [3], que incorpora conceitos musicais ao ambiente de programação Smalltalk [5]. Todos estes conceitos são automaticamente mapeados para objetos da linguagem Smalltalk que podem ser manipulados e reutilizados na implementação de novas ferramentas computacionais voltadas para a música.

A seção seguinte se concentra na utilização de objetos como forma de representação da música. A seção 3 faz uma breve descrição do modelo Smallmusic. Na seção 4, é apresentado o ambiente CAMC e sua abordagem de composição como prototipação. A seção 5 contém algumas conclusões sobre o trabalho desenvolvido.

**2. Representação da Música através de Objetos**

Reais ou abstratos, objetos estão presentes em todas as atividades humanas e a música não é exceção. Até os mais simples conceitos musicais podem ser descritos como objetos. Se observarmos atentamente uma partitura musical tradicional poderemos perceber facilmente a existência de inúmeros elementos cujo comportamento é variado porém com uma série de características em comum, como podemos verificar na figura 1.

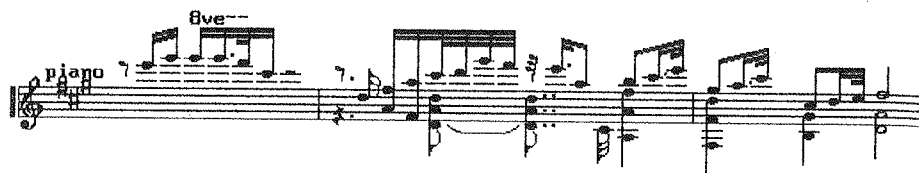


Figura 1

Todos estes objetos se interrelacionam agrupadamente de forma a constituir outros objetos mais complexos tais como melodias e trechos musicais ou até mesmo uma composição completa. A partitura anterior, por exemplo, poderia ser organizada como na figura 2.

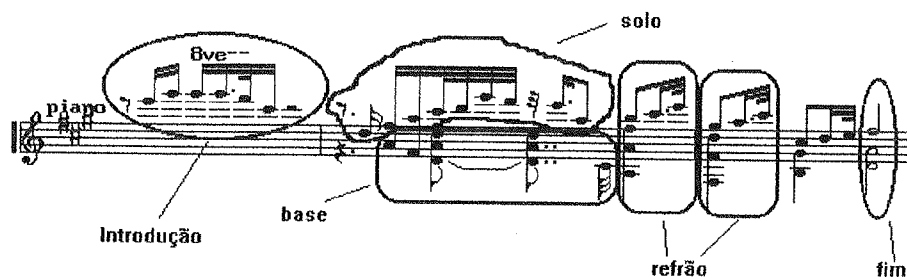


Figura 2

Esta figura apresenta de forma mais clara como está organizada a composição acima. Os objetos mais simples (figuras musicais) estão agrupados constituindo objetos mais complexos. Entretanto, embora de diferentes complexidades, todos estes objetos apresentam um comportamento similar e podem ser utilizados da mesma forma (uniformidade). Podemos verificar no exemplo dado que, do mesmo modo que figuras musicais tradicionais como as colcheias aparecem diversas vezes ao longo dos trechos, o objeto refrão aparece duas vezes na música (um tom abaixo na segunda vez).

Esta forma de representação facilita o processo de criação, pois o compositor é capaz de conceber a estrutura básica de uma música simplesmente compondo os objetos do sistema, mesmo que internamente estes objetos ainda não estejam completamente definidos ou adaptados para a composição

em questão. No pentagrama tradicional, podemos imaginar a mesma música descrita por objetos como na figura 3.

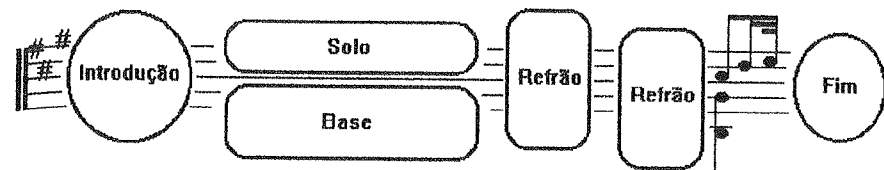


Figura 3

É importante ressaltar que uma música definida desta forma, mesmo que incompleta, poderá ser reutilizada na composição de novas músicas pois, na visão orientada a objetos, ela é um objeto como qualquer outro.

**3. O modelo Smallmusic**

O modelo Smallmusic é um modelo orientado a objetos para a representação da música. Objetos são estruturas de dados auto-descritivas cujo comportamento está definido em subrotinas (métodos) auto-contidas [5]. Como nosso objetivo está centrado em definir e interagir com conceitos musicais e sons, vamos tratar de objetos específicos para esta tarefa denominados *sound objects*.

Um *sound object* tem três características principais: *nome*, que identifica o objeto; *icone*, a representação gráfica do objeto e *parâmetros*, que descrevem o som ou evento produzido pelo objeto. Estes objetos estão organizados em uma hierarquia de classes como na figura 4.

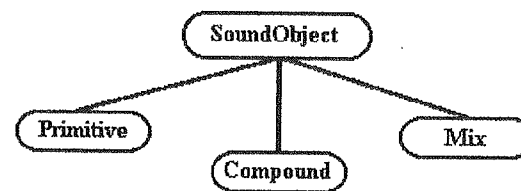


Figura 4

A classe *SoundObject* é a superclasse, e define o comportamento e as características básicas de todas as *sound objects*. Nesta classe estão definidos os parâmetros que são herdados por todas as suas subclasses.

A classe *Primitive* generaliza as classes primitivas do Smallmusic. Estas classes modelam os conceitos musicais mais simples e estão diretamente relacionadas com o protocolo MIDI [4], ou seja, para cada subclasse definida existe uma mensagem MIDI correspondente (*note On*, *note Off*, etc.). Estas primitivas são as únicas classes capazes de produzir sons ou eventos diretamente. Se desejarmos, por exemplo, enviar uma mensagem *Note On* para o sintetizador basta criarmos um instância da classe *NoteOn*, configurar seus parâmetros e executá-la, da seguinte forma:



```
| nOn |
```

```
nOn := NoteOn new.
nOn
  tick: 20.
  midiChannel: 10;
  key: 64;
  velocity: 100.
nOn perform.
```

Especializando a classe *Compound*, é possível incorporar novas classes a hierarquia do *Smallmusic*. Estas classes são ditas compostas, já que a sua estrutura é definida através da composição de conceitos musicais modelados em outras classes primitivas ou compostas. Os objetos que formam uma classe composta são chamados componentes. Por exemplo, dois componentes serão necessários na definição da classe *Nota*: um da classe *NoteOn* e outro da classe *NoteOff*. Já um acorde (tríade), pode ser definido como uma classe composta que possui três componentes da classe *Nota*.

Uma música (ou trecho) pode ser vista como uma seqüência de *sound objects* interrelacionados. A classe *Mix* (mistura) representa este conceito. Uma mistura é o agrupamento de instâncias que pode ser efetuada de forma seqüencial ou paralela. Estes dois tipos de mistura estão relacionados diretamente com a disposição das instâncias ao longo do tempo. Por exemplo, para criarmos um *Mix* de uma escala cromática podemos efetuar uma mistura seqüencial, como abaixo:

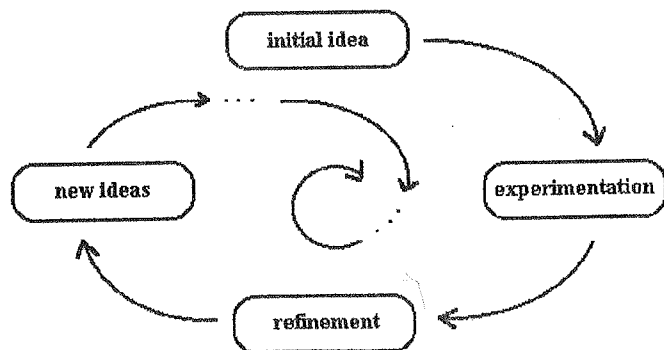
```
| escala |
```

```
escala := Mix new.
1 to: 12 do: [ :p | escala := escala,(Nota new pitch: p). ]
```

#### 4. O Ambiente CAMC

O ambiente CAMC foi projetado para auxiliar o compositor durante o processo de criação musical. Este processo não pode ser definido formalmente e tampouco delimitado por uma seqüência de passos enumeráveis. Além disso, diferentes pessoas podem ter motivações distintas para compor uma música e, como consequência, processos próprios de criação serão utilizados. Entretanto, processos de criação serão utilizados. Entretanto, processos de criação serão utilizados. Entretanto, processos de criação serão utilizados. Entretanto, processos de criação serão utilizados.

Desta forma, é possível considerar o processo de criação musical como uma atividade de prototipação (figura 5). A representação da música através de objetos apresentada na seção 2 está de acordo com esta idéia, pois neste enfoque a música pode ser descrita por uma seqüência de objetos já concluídos, como na figura 1, ou que serão refinados, como na figura 3.



Para permitir a aplicação dos conceitos relacionados a prototipação no processo de composição (figura 5), o ambiente CAMC dispõe de duas ferramentas: o Desktop (figura 6) e o Estúdio de Som (figura 7).

A principal característica do Desktop é suportar um modo experimental de trabalho, permitindo ao compositor avaliar rapidamente o resultado de suas experiências. Por sua vez, o Estúdio de Som é dedicado à atividade de performance, possibilitando a gravação e reprodução de objetos musicais. Estas duas ferramentas são complementares e manipulam os objetos existentes no ambiente de maneira integrada.

O ambiente foi desenvolvido na linguagem Smalltalk V/286 e utiliza o padrão MIDI para a comunicação com os instrumentos musicais. O controle de mensagens MIDI ocorre em tempo real, através de processos ativados por interrupção, o que confere maior flexibilidade na interação do usuário com o ambiente.

#### 5. Conclusões

Neste artigo apresentamos um ambiente de auxílio à composição musical, no qual foi utilizado um modelo orientado a objetos com o objetivo de suportar a idéia de composição como prototipação.

Existem outros tipos de ambientes para apoio à composição musical que se baseiam na notação tradicional. Concordamos que a notação musical tradicional seja necessária para o registro de melodias devidamente concluídas, já que é uma linguagem mundialmente consagrada. Entretanto, a sua representação rígida<sup>1</sup> (notas, tempos e compassos) não nos parece ideal para a atividade de criação musical. Por este motivo, pretendemos incorporar ao ambiente uma nova ferramenta que permita ao usuário editar os objetos numa espécie de pentagrama. Neste caso, as figuras musicais corresponderiam às classes compostas definidas no modelo.

Iremos continuar evoluindo a modelagem aqui apresentada. Estamos investigando a utilização do modelo delegativo de protótipos [1] na implementação dos objetos, com o objetivo de ampliar a uniformidade do modelo e a interatividade do ambiente.

#### 6. Referências

- [1] Ballista, A.L.C. (1993). Protótipos e Delegação - Uma Experiência de Desenvolvimento. *Relatório Técnico. Núcleo de Computação Sônica. Universidade Federal de Santa Catarina.*
- [2] Martins, A.J.B & Ballista, A.L.C. (1992). CAMC - Composição Musical Auxiliada por Computador. *Relatório Técnico. Departamento de Informática e Estatística. Universidade Federal de Santa Catarina.*
- [3] Martins, A.J.B & Pimenta, M.S. (1993). Smallmusic - Uma conversa musical em Smalltalk. *Anais da XIX Conferência Latino-Americana de Informática.* Volume 1. pp 593-610
- [4] Furia, S. & Scacciaferro, J. (1990). *The MIDI Programmer's Handbook.* M&T Books.
- [5] Goldberg, A. & Robson, D. (1983). *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

<sup>1</sup>Se desejarmos representar uma nota com duração de 4 semínimas (semibreve) em um compasso 2 por 4, serão necessárias duas mínimas em compassos sucessivos unidas por uma ligadura. Desta forma, o objeto que queríamos representar foi modificado para respeitar o tempo do compasso. Da mesma maneira, existem algumas situações difíceis de serem representadas pela notação tradicional.

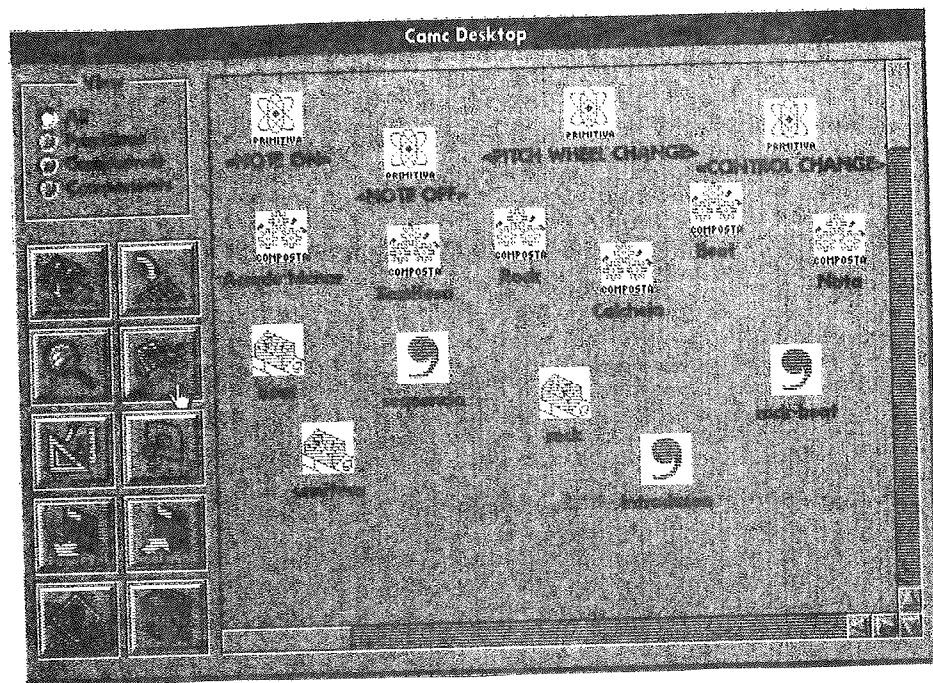


Figura 6 - Desktop

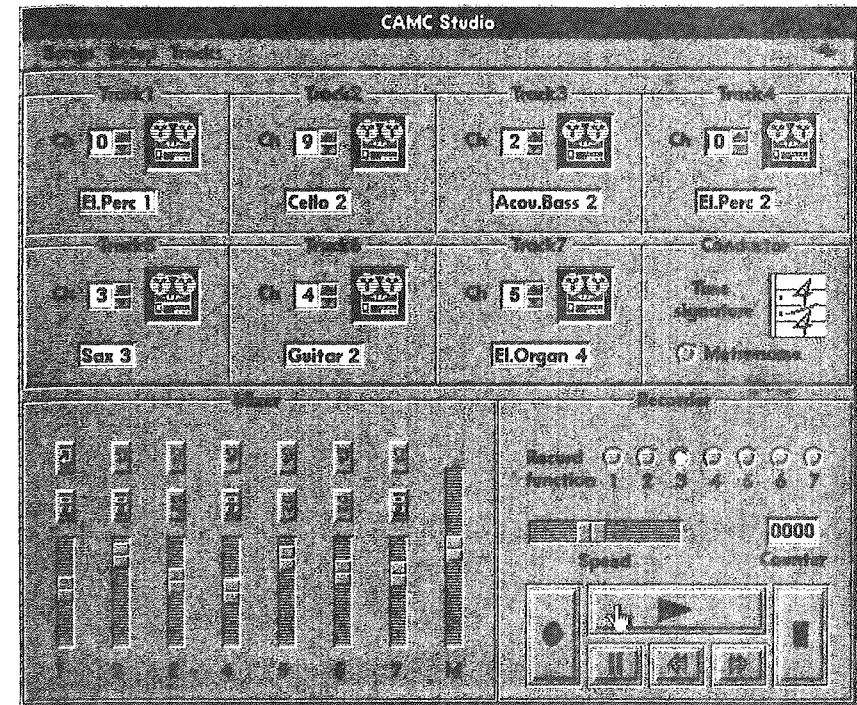


Figura 7 - Estudio de Som

## Orquestrador MIDI Sinfônico

OSMAN GIUSEPPE GIOIA  
*Laboratório de Processamento Espectral*  
*Departamento de Ciência da Computação*  
*Universidade de Brasília*  
*Brasília - D F - CEP 70.910-900*

### Resumo

A Orquestra Sinfônica moderna representa atualmente o estado da arte na execução musical em conjunto e podemos considerá-la como uma máquina sonora multitimbral constituída por várias unidades harmônicas espectralmente diversas - os instrumentistas e seus instrumentos - que, atuando em naipes e seções constituem uma estrutura espectralmente finita. Por outro lado, nas composições algorítmicas que utilizam a síntese aditiva como meio expressivo, são utilizadas unidades harmônicas senoidais que possibilitam a elaboração de estruturas espectrais potencialmente infinitas. O trabalho aqui apresentado tem como objetivo realizar um mapeamento entre os instrumentos algorítmicos e os sinfônicos que, utilizando preceitos definidos a partir de atributos psico-acústicos previamente levantados, gere uma orquestração para execução por orquestra sinfônica ou através de módulos de som digitais que estejam enquadrados no padrão General MIDI.

### Introdução

Os sistemas de composição algorítmica e síntese aditiva em geral, baseiam-se nos conceitos de "instrumentos" e "orquestra" na geração e reprodução dos timbres espectrais e portanto, um dos caminhos naturais de execução por outros meios de reprodução sonora constitui-se no mapeamento das características espectrais geradas pelos instrumentos da orquestra algorítmica para a sua contrapartida nos instrumentos de uma orquestra sinfônica ou na falta desta, em seu correlato digital utilizando-se as seções instrumentais sinfônicas do padrão General MIDI. O Orquestrador MIDI Sinfônico aqui apresentado foi portanto implementado com o intuito de mapear as cartas espectrais geradas pelo CARBON (Arcela, A., 1989) para o domínio dos instrumentos sinfônicos. Este mapeamento tornará viável não só a execução sinfônica através da impressão de partituras no sistema NOTACOR (Meireles, A., 1994), quanto digital, através de seqüências GM executadas pelo sistema MAESTRO (Pereira, A., 1994), programas estes que junto ao trabalho em questão compõem parte do Projeto DECIBÉIS (Arcela, A., 1993).

### Instrumentos Sinfônicos

Para que possamos realizar uma orquestração correta, precisamos de alguns dados básicos sobre cada um dos instrumentos com os quais trabalharemos. Entre eles podemos destacar a extensão, o tempo de ataque, o centro da banda passante espectral e a duração de sua envoltória de amplitude no caso dos instrumentos percussivos. Para o levantamento destes dados, além de utilizarmos pesquisas previamente realizadas. (Fonte principal: Luce, D. & Clark, M., 1965), estamos em fase de realização de experimentos paralelos tanto para a checagem final dos dados quanto para o preenchimento daqueles não disponíveis. Como o nosso objetivo é o de orquestrar também para módulos digitais, além de trabalharmos somente com os instrumentos comuns aos dois

sistemas, estamos utilizando para o levantamento dos dados a biblioteca de sons da Roland Corporation (Roland Sound Library) como fonte principal das amostras digitalizadas.

### Instrumentos de Corda

Através da comparativamente curta história da orquestração, o grupo de instrumentos de cordas - violinos, violas, violoncelos e contrabaixos - manteve a sua posição de elemento dominante na orquestra sinfônica e são os instrumentos com o maior tempo de ataque (Piston, Walter, 1955; Beauchamp, J.W., 1974; Carleen, M.H., 1973; Luce, D. & Clark, M., 1967).

Tabela 1 - Cordas

	Extensão	Centro	Transientes de Ataque:			Duração
			Solo	Grupo	Pizzicato	
Violinos	G3 - C7	E5	0.080 s	0.071 s	0.032 s	0.683 s
Violas	C3 - D6	G4	0.056 s	0.071 s	0.053 s	0.726 s
Violoncelos	C2 - C5	F#3	0.085 s	0.071 s	0.019 s	0.675 s
Contrabaixos	E1 - G3	F#2	0.062 s	0.071 s	0.039 s	0.725 s

### Instrumentos de Sopro

#### Sopros de Madeira

Um definição lógica e convincente da categoria conhecida como madeiras é difícil de ser proposta pois os corpos de seus instrumentos são construídos de diversos materiais. Porém na orquestra os instrumentos assim classificados são de embocadura livre, palheta simples ou dupla, e a flauta é, junto às cordas, o instrumento de ataque mais longo (Berlioz, H. & Strauss, R., 1948; Fletcher, N.H., 1975; Luce, D. & Clark, M., 1967).

Tabela 2 - Madeiras

	Extensão	Centro da Banda Passante	Transientes de Ataque
Flautas	C4 - C7	F#5	0.080 s
Flautim	D5 - C8	G6	0.024 s
Oboes	A#3 - G6	D#5	0.015 s
Corne Inglês	E3 - A5	F#4	0.014 s
Clarinetas	D3 - C5	A#4	0.030 s
Fagotes	A#1 - G6	F3	0.042 s

#### Sopros de Metal

Todos os instrumentos deste naipe são de metal e seu som é produzido através da vibração dos lábios em um bocal (Jachino, C., 1950; Luce, D. & Clark, M., 1967).

Tabela 3 - Metais

	Extensão	Centro da Banda Passante	Transientes de Ataque
Trompas	F2 - F5	B3	0.045 s
Trumpetas	A#3 - A#6	E5	0.030 s
Trombones	A#1 - D#5	F#3	0.033 s
Tuba	F1 - F5	F#2	0.062 s

#### Saxofones

Apesar de não serem parte integrante do naipe de sopros de uma orquestra sinfônica, foram incluídos no orquestrador como instrumentos opcionais. (Mancini, H., 1954).

Tabela 4 - Saxofones

	Extensão	Centro da Banda Passante	Transientes de Ataque
Soprano	F#3 - D#6	A#4	0.073 s
Alto	C#3 - G#5	E4	0.029 s
Tenor	E2 - D#5	B3	0.022 s
Barítono	C#2 - G#4	F#3	0.030 s

### Instrumentos de Percussão

Podemos definir os instrumentos de percussão como aqueles nos quais o som é produzido pelo choque entre dois objetos. (Berlioz, H. & Strauss, R., 1948)

#### Altura Definida

Com exceção dos Tímpanos que são membranophones todos os outros são idiophones. (Christian, R. S. & Davis, R. & Tubis, A. & Anderson, C. & Mills, R. & Rossing, T., 1984).

Tabela 5 - Percussão Definida

	Extensão	Centro da Banda	Ataque	Duração
Glockenspiel	C5 - C8	F#6	0.0006 s	0.8176 s
Vibrafone	F3 - F6	B4	0.0292 s	0.9432 s
Marimba	C3 - C6	F#4	0.0007 s	0.8163 s
Xilofone	F4 - C7	G#5	0.0025 s	0.3892 s
Carrilhão	C4 - F5	A4	0.0025 s	2.6340 s
Tímpanos	C2 - A3	A#2	0.0336 s	1.3312 s

#### Altura Indefinida

Por não possuírem uma frequência dominante auditivamente discernível consideramos como tal, a frequência predominante no ponto máximo de sua envoltória de amplitude e por serem muito diversificados, utilizaremos somente aqueles constantes da especificação General MIDI. (Aikin, J & Marans, M. & Rule, G., 1993; Fletcher, H. & Basset, L., 1978; Rossing, T. & Bork, I. & Zhao, H. & Fristom, D., 1992).

Tabela 6 - Percussão Indefinida

	Frequência	Transientes	Duração		Frequência	Transientes	Duração
Bumbo	40.000	0.011451 s	1.517891 s	Agogo Agd	1382.100	0.003288 s	0.076871 s
Caixa	233.101	0.006122 s	0.161383 s	Agogo Grv	1035.890	0.004376 s	0.102608 s
Castanholas	1612.570	0.001202 s	0.016939 s	Cabaça	10985.600	0.049297 s	0.065941 s
Pandeiro	8491.970	0.005351 s	0.151837 s	Maracas	12685.200	0.004580 s	0.052721 s
Cowbell	487.378	0.004807 s	0.120952 s	Apito Crto	2256.800	0.073447 s	0.094490 s
Pratos	5229.220	0.041814 s	1.941020 s	Apito Long	2005.300	0.281723 s	0.391361 s
Queixada	993.871	0.003946 s	0.950975 s	Reco Curto	1239.860	0.012472	0.030816 s
Prato	6091.350	0.076463 s	1.618798 s	Reco Long	2103.800	0.221134 s	0.252925 s
Bongô Agd	756.922	0.008389 s	0.059569 s	Claves	2221.100	0.001769 s	0.026145 s
Bongô Grv	373.488	0.002880 s	0.089864 s	Wblock Ag	822.152	0.001383 s	0.102041 s
Conga Mut	757.090	0.003129 s	0.030499 s	Wblock Gr	611.488	0.001088 s	0.076463 s
Conga Opn	276.168	0.009773 s	0.134286 s	Cuica Mute	720.458	0.059592 s	0.157800 s
Conga Grv	206.815	0.013039 s	0.179705 s	Cuica Opn	513.411	0.043605 s	0.198685 s
Timbal Ag	913.163	0.004671 s	0.237302 s	TriângMut	5494.230	0.000975 s	0.150635 s
Timbal Gr	691.270	0.006213 s	0.315147 s	TriângOpn	8542.810	0.000975 s	1.136304 s

### Orquestração MIDI Sinfônica de Cartas Espectrais

A orquestração sinfônica ou MIDI sinfônica das cartas espectrais é realizada observando-se uma série de preceitos que norteiam o mapeamento unívoco dos instrumentos algorítmicos em instrumentos sinfônicos. Após o levantamento e sistematização destas diretrizes de acordo com as características intrínsecas de cada um dos meios de expressão espectrais envolvidos, pudemos então desenvolver o Orquestrador MIDI sinfônico.

#### Instrumentos Sinfônicos

O primeiro e mais importante requisito para a realização de uma orquestração musicalmente correta, é o de respeitarmos a extensão de execução orquestral de cada instrumento, além de prevenir o aparecimento de notas mapeadas para os extremos da mesma concentrando-as em torno da faixa ideal de atuação de cada instrumento, ou seja, próximas do centro de sua banda passante espectral.

Outro fator para o qual precisamos atentar são as características espectrais de cada instrumento. A informação espectral pura não nos fornece subsídios suficientes para a realização de um mapeamento determinístico, pois os espectros sônicos gerados pela síntese aditiva não encontram paralelo nos timbres obtidos a partir dos instrumentos sinfônicos e suas combinações. Além disto, vários trabalhos nos mostram (Luce & Clark, 1965, 1967; Seashore, 1967; Pierce, 1983) que uma das características mais importantes no reconhecimento da maioria dos sons são os transientes de ataque e não a porção contínua do espectro harmônico de uma determinada fonte sonora.

#### Instrumentos Algorítmicos

Os instrumentos algorítmicos gerados através de síntese aditiva por serem totalmente manipuláveis não encontram contrapartida espectral no mundo dos instrumentos sinfônicos. Além disso, a finalidade de uma orquestração não é a de mapear timbres que poderiam eventualmente ser assemelhados, mas sim realizar uma "transposição espectral" utilizando as características intrínsecas de cada meio de reprodução. Outro fator que torna impossível o mapeamento *ipsis literis* de um instrumento algorítmico em sua contrapartida sinfônica é o da extensão de execução. Sabemos também que um instrumento algorítmico é constituído na realidade não de um único instrumento no sentido tradicional da palavra, mas sim, representa uma família de instrumentos geneticamente interligados (Arcela & Ramalho, 1991; Arcela, 1986).

A última consideração a respeito da construção dos instrumentos algorítmicos diz respeito à temporalidade relativa e não absoluta de suas envoltórias de amplitude. Na atual implementação do programa de síntese SOM-A (Arcela, 1989), as unidades-h tem a sua envoltória de amplitude implementada de tal forma que os tempos de cada segmento da envoltória variam com a duração da nota executada e os instrumentos musicais não sofrem variação significativa destes segmentos relativa à duração da mesma.

Com isto vemos que torna-se impraticável tentarmos mapeá-los diretamente e os principais parâmetros passíveis de mapeamento a nosso ver portanto são a extensão executável de cada instrumento sinfônico e o levantamento minucioso dos transientes de ataque dos mesmos, que então seriam classificados e escalonados temporalmente em relação à sua contrapartida algorítmica no âmbito da duração de cada nota executada mapeando-os pelo critério de maior ou menor percussividade. Como ponto de partida para este levantamento usamos a definição de transientes de ataque como sendo o período de tempo a partir do início do sinal até o ponto no qual a sua magnitude esteja 3 decibéis acima do estado contínuo (Luce & Clark, 1965). Falta-nos agora definir o seu correspondente na orquestra algorítmica.

•Definição: Podemos considerar que o correspondente algorítmico de transiente de ataque de uma unidade-h, corresponde ao lapso de tempo entre o instante inicial e o ponto de quebra da envoltória com a maior amplitude e por conseguinte um instrumento com mais de uma unidade-h possuirá tempo de ataque correspondente à unidade-h com predominância acústica, sobre as demais de acordo com o Teorema do Estado de Equilíbrio (Arcela, 1984).

A partir deste teorema, podemos então extrair duas informações importantes para o nosso mapeamento:

1. O Tempo de Ataque de um instrumento será o tempo de ataque da unidade-h predominante entre as ativas para uma determinada nota com relação à duração da mesma.
2. O som de altura de uma determinada nota será a frequência especificada na mesma, multiplicada pela ordem da unidade-h predominante entre as ativas para aquela nota.

Além disso os instrumentos são mapeados com a informação ortocostereofônica constante da carta e isto significa que na realidade teremos duas orquestras que idealmente deveriam executar suas respectivas partituras ortogonalmente posicionadas (Arcela, 1984, 1986).

Em relação à amplitude, esta será o resultado da multiplicação da amplitude da nota pela maior amplitude da unidade-h ativa a ela associada, mantendo assim o escalonamento dinâmico entre as diversas notas e instrumentos que as executarão.

Quanto aos tempos de início e duração de cada nota mapeada, estes são multiplicados por uma constante que depende tanto da figura adotada como unidade de tempo quanto da resolução do sequenciador MIDI utilizado. Na atual implementação o valor de duração 1 da carta espectral corresponde à fusa e a resolução empregada é de 120 divisões por semínima.

A partir destas conclusões podemos agora definir um algoritmo de orquestração sinfônica das cartas espectrais. Foram implementados três modelos de orquestração, em um crescendo de complexidade interpretativa que acreditamos poderão gerar resultados matemática e musicalmente acabados.

#### Modelo 1

A partir da análise dos dados contidos em cada nota individualmente relacionados ao instrumento que a executa, extraímos as informações necessárias ao mapeamento da mesma para um e somente um instrumento sinfônico da seguinte forma:

1. Som de altura: Temperamento da frequência da nota multiplicada pela ordem da unidade-h com predominância acústica sobre as demais ativas para aquela nota multiplicada pela transposição constante do cabeçalho da carta espectral.

2. Mapeamento: A partir do som de altura são selecionados entre os instrumentos que possuam extensão compatível aqueles com o tempo de ataque mais próximo do tempo de ataque do instrumento algorítmico, caso o tempo de ataque da nota seja maior do que o maior tempo de ataque dos instrumentos não-percussivos, a nota é mapeada para flauta ou cordas pois são os instrumentos com o maior tempo de ataque como nos mostram as tabelas 1 e 2. Caso o tempo de ataque da nota seja menor do que o menor tempo de ataque dos instrumentos percussivos a nota é mapeada para um instrumento de percussão com altura indefinida de acordo com a tabela 6. Finalmente é mapeado aquele que tenha o seu centro de banda passante espectral mais próximo do som de altura. No caso dos instrumentos percussivos ainda levamos em conta a proximidade entre a duração da nota e as durações totais de suas envoltórias de amplitude de acordo com a tabela 6.

Conclusão: Este mapeamento prevê que cada uma das notas executadas originalmente seja carregada para o instrumento mais apto a executá-la, tanto em relação a suas características originais de maior ou menor percussividade quanto à sua compatibilidade em altura, extensão e velocidade de articulação pois as notas mais longas naturalmente tenderão para instrumentos mais lentos de ataque e vice-versa.

Característica: A orquestra resultante torna-se viável de execução por uma orquestra tradicional ou pelo sistema General MIDI padrão, isto é, multitímbral a 16 partes e polifônico a 24 vozes no mínimo, com uma complexidade de execução orquestral moderada.

#### Modelo 2

A partir da análise dos dados contidos em cada nota individualmente relacionados ao instrumento que a executa, extraímos as informações necessárias ao mapeamento da mesma para um ou mais instrumentos sinfônicos passíveis de executá-la da seguinte forma:

1. Som de altura: Idem ao Modelo 1.

2. Mapeamento: Idem ao Modelo 1, só que agora, não apenas um, mas todos os instrumentos selecionados são mapeados para aquela determinada nota.

Conclusão: Este mapeamento prevê que cada uma das notas executadas originalmente seja carregada para um ou mais instrumentos aptos a executá-la, tanto em relação a suas características originais de maior ou menor percussividade quanto à sua compatibilidade em altura, extensão e velocidade de articulação.

Característica: Este método introduz a utilização da combinação instrumental em cada naipe, de maneira que agora não teremos apenas um instrumento, mas sim um grupo de instrumentos executando uma determinada nota em uníssono. Através deste sistema obteremos resultados tímbricos muito mais elaborados do

que através do primeiro, mantendo ainda uma certa portabilidade tanto em relação à execução MIDI quanto sinfônica.

### Modelo 3

A partir da análise dos dados contidos em cada nota individualmente relacionados ao instrumento que a executa, extraímos as informações necessárias ao mapeamento da mesma para tantos instrumentos sinfônicos quantas sejam as unidades-h que a executam.

1. Som de altura: Temperamento da frequência da nota multiplicada pela ordem de cada unidade-h ativa multiplicada pela transposição constante do cabeçalho da carta espectral gerando assim um número de notas correspondente ao número de unidades-h ativas cada uma com sua respectiva frequência.

2. Mapeamento: Idem ao Modelo 2, só que agora ao invés de um ou mais instrumentos executando uma nota em uníssono, teremos como resultado o mapeamento de um instrumento para cada unidade-h ativa.

Conclusão: Neste Modelo utilizamos um princípio de combinação orquestral que encontra o seu máximo expoente no compositor impressionista francês Maurice Ravel, que em sua obra prima "Bolero", utilizou os instrumentos da orquestra não como entidades espectrais isoladas mas sim como se fôssem "unidades-h" de um instrumento maior, cada um executando o que seria um harmônico de um som espectral mais amplo, em uma espécie de síntese aditiva orquestral. Deste modo podemos utilizar os mais variados recursos de combinação instrumental possibilitando a obtenção de efeitos combinatórios os mais diversificados possível e levando ao extremo o que seria a essência da orquestração: a técnica da combinação instrumental.

Característica: Utilização plena dos recursos de combinação instrumental de uma orquestra sinfônica possibilitando a geração das mais diversas configurações instrumentais. Capacidade de obtenção de timbres orquestrais inéditos e instigantes perceptualmente.

### O Programa

Implementado em estação SUN SparcStation na linguagem C com interface gráfica gerada pelo DevGuide, utiliza como entrada arquivos no formato \*.car gerados pelo CARBON e gera um arquivo de saída no formato \*.ces no qual os rótulos de instrumentos algorítmicos e/ou suas unidades-h são substituídos ao nível de cada nota pelo instrumento sinfônico e mudança de programa MIDI correspondente além de conter os outros dados necessários à execução MIDI ou sinfônica. Ao término da execução do Modelo selecionado, é aberta uma janela na qual pode-se visualizar instantaneamente quais foram os instrumentos sinfônicos mapeados. Além disso o programa conta com um visualizador gráfico das envoltórias de amplitude e gráfico de barras das ordens e amplitudes máximas de cada unidade-h correspondente aos instrumentos constantes da carta sendo trabalhada, além de possibilitar a escolha da instrumentação desejada, permitindo a definição de quais instrumentos estarão ativos para mapeamento, possibilitando assim a geração desde partituras para grande orquestra até conjuntos menores.

### Considerações Finais

O Orquestrador MIDI Sinfônico fornecerá ao LPE os meios de transcrição e conversão necessários a uma execução MIDI ou sinfônica das cartas espectrais geradas algorítmicamente e neste sentido, é um projeto voltado exclusivamente para as aplicações diretamente relacionadas com o ambiente composicional do LPE.

### Agradecimentos

Sou especialmente grato ao meu orientador de mestrado Aluizio Arcela, agradecimentos esses extensivos à minha instituição de origem, Departamento de Música da UFPE e ao Departamento de Ciência da Computação da UnB por acreditarem na interdisciplinaridade acadêmica.

### Referências

- Aikin, J. & Marans M. & Rule, G. (march 1993). General MIDI. *Keyboard*.
- Arcela, A. & Ramalho, G. (1991). A formal composition system based on the theory of Time-trees. *Proceedings of the ICMC*, Montreal.
- Arcela, A. (1984). As árvores de Tempos e a configuração Genética dos Intervalos Musicais. *Tese de Doutorado*, PUC, Rio de Janeiro.
- Arcela, A., (1986). Time-Trees: the inner organization of intervals. *Proceedings of the 12th International Computer Music Conference (ICMC)*, pp 87-89, Haia.
- Arcela, Aluizio (1989). CARBON, Relatório Técnico LPE8901
- Arcela, Aluizio (1994). A Linguagem SOM-A para Síntese Aditiva, *Anais do 1º Congresso Brasileiro de Computação e Música, Caxambú, MG*.
- Arcela, Aluizio (1993). DECIBÉIS, DPP - Projetos Gerais
- Beauchamp, J.W. (1974). Time-variant spectra of violin tones. *J.Acoust.Soc.Am.*, Vol.56, N.3, September pp.995-1004.
- Berlioz, H. & Strauss, R. (1948). Treatise on Instrumentation. *Kalmus*, New York, N.Y.
- Carleen, M.H. (1973). Instrumentation and Methods for Violin Testing. *Journal of the Audio Engineering Society*, September X, Volume 21, Number 7, pp.563-570.
- Christian, Richard S. & Davis, Robert E. & Tubis, Arnold & Anderson, Craig A. & Mills, Ronald I. & Rossing, Thomas D. (1984). *J. Acoust. Soc. Am.* November 76(5)
- Fletcher, H. & Basset I. (1978). Some experiments with the bass drum. *J. Acoust. Soc. Am.*, 64(6), Dec.
- Fletcher, N.H. (1975) Acoustical correlates of flute performance technique. *J.Acoust.Soc.Am.*, Vol.57, No.1, January, pp.233-237.
- Jachino, C. (1950). *Gli Strumenti D'Orchestra*, Curci, Milano.
- Luce, D. & Clark, M. (1965). Durations of Attack of Nonpercussive Orchestral Instruments. *Journal of the Audio Engineering Society*, July, volume 13, number 3, pp. 194-199.
- Luce, D. & Clark, M. (1967). Physical Correlates of Brass-Instrument Tones. *Journal of the Acoustic Society of America*, Volume 42, Number 6, pp. 1232-1243.
- Mancini, Henry (1973). *Sounds and Scores*. Northridge Music, inc. USA.
- Meireles, Alex (1994). NOTACOR, Impressão de Partituras em Cores, *Anais do 1º Congresso Brasileiro de Computação e Música, Caxambú, MG*.
- Pereira, Antônio (1994). MAESTRO, Tese de Mestrado em andamento LPE-CIC-UnB.
- Pierce, J.R. (1983). *The Science of Musical Sound*. American books, Inc., New York.
- Piston, W. (1955). *Orchestration*. W.W.Norton & Co., Inc., New York.
- Rossing, T. & Bork, I. & Zhao, H. & Fystrom, D. (1992) Acoustics os snare drums. *J. Acoust. Soc. Am.* 92(1), July.
- Seashore, C.E. (1967). *Psychology of music*. Dover Publications, Inc., New York.

## EDITOR DE PRECEITOS INTERVALARES

RICARDO RIBEIRO DE FARIA CASTRO  
*Laboratório de Processamento Espectral*  
*Departamento de Ciência da Computação*  
*Universidade de Brasília*  
*Brasília, DF, CEP. 70910-900*

### RESUMO

A análise do Intervalo musical, composto pela simultaneidade de dois tons puros, revela sob a ótica do Objeto Intervalar, um sistema de informações de natureza musical, organizado de forma hierárquica pelas Árvores de Tempos. O Objeto Intervalar pode ser representado por um Preceito, ou um conjunto de parâmetros que o definem como entidade euclidiana, e que possibilitam a construção da Árvore de Tempos de forma algorítmica, e a extração de seu interior de informações de natureza geométrica e temporal. As Árvores, por apresentarem uma natureza de incompletude, podem ser acasaladas de forma geométrica. Assim se caracteriza o método de composição algorítmica calcado no acoplamento de objetos intervalares. Tal acoplamento pode ocorrer sob diversas formas, dependendo do posicionamento no espaço euclidiano do Objeto Intervalar, bem como da escolha dos vértices a serem usados no acoplamento. A necessidade de um ambiente computacional que suporte a modelagem de Objetos Intervalares, e que organize de forma coerente as suas informações, possibilitando a manipulação de forma interativa e dinâmica dos parâmetros relativos ao acoplamento geométrico, motivou o projeto e implementação de um sistema Editor de Preceitos Intervalares, objeto deste trabalho.

### O OBJETO INTERVALAR

O intervalo musical é constituído por um par de frequências que distam uma da outra numa razão fixa, e pode ser visto como a menor unidade de um espectro sonoro musical. A necessidade de se compreender com maior profundidade as características do fenômeno intervalar e um melhor entendimento da forma auditiva é que motivou a ampliação dos conceitos existentes através da análise mais profunda do fenômeno intervalar. Assim buscou-se através da observação da geometria da composição ortogonal de sinais senoidais harmônicos, uma representação geométrica tridimensional para o intervalo musical.

O Objeto Intervalar pode ser descrito como o modelamento do fenômeno Intervalar através da composição ortogonal de um MHS (Movimento Harmônico Simples) e de um MCU (Movimento Circular Uniforme), dado pelas seguintes equações:

$$y = a \sin((2\pi St)/\tau + \phi_0) \quad (\text{MHS})$$

$$\gamma = \gamma_0 + (2\pi st)/\tau, \quad r=b \quad (\text{MCU}),$$

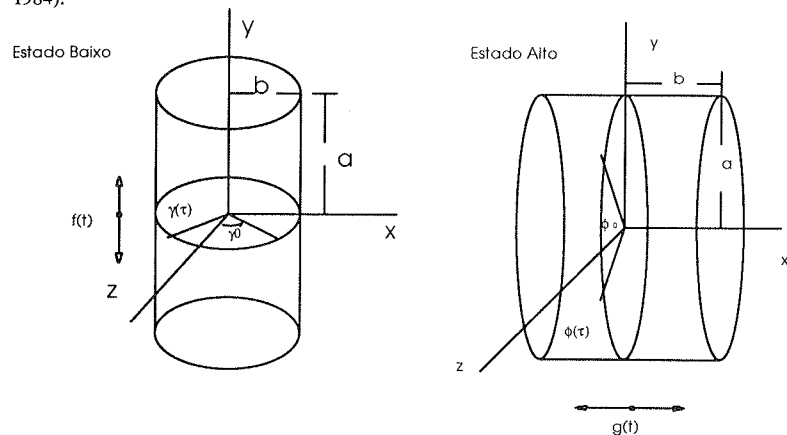
onde S e s são inteiros primos entre si correspondentes à razão entre as frequências;  $\phi_0$  a fase inicial de y e  $\gamma_0$  a fase inicial de  $\gamma$ , e  $\tau$  é o período do movimento intervalar. Verifica-se também que o movimento vibratório é caracterizado pela ordem de frequência associada ao MHS, tendo o MCU a função de apoio estrutural na unidade intervalar.

A trajetória descrita por um ponto cartesiano, submetido ortogonalmente a esses dois movimentos, é inscrita num cilindro, cuja altura e raio dependem das amplitudes das componentes de frequência do intervalo. Ao longo desta trajetória, vão ocorrendo auto-interseções, que dão origem a um conjunto de pontos, denominados Momentos, que demarcam os lapsos de tempo ao longo da sequência de auto-interseções (Arcela, 1984).

### ESTADO E NÃO-ESTADO DO OBJETO INTERVALAR

A observação dos Objetos Intervalares demonstra que, às frequências do intervalo podem estar associados dois Objetos distintos, cujas projeções nos planos xz e yz são idênticas, implicando na necessidade de se introduzir novos conceitos. A existência simultânea de duas componentes de frequência é marcada pela

predominância de uma sobre a outra, e a que predomina impõe o seu padrão espectral. O Estado (S) de um Objeto Intervalar é caracterizado pela componente de frequência predominante, determinando o MHS do movimento intervalar. O Não-Estado (s) é dado pela ordem de frequência mascarada, e que determina o Movimento Circular. Um Objeto Intervalar se diz no Estado Baixo se a Ordem de Frequência do MHS for menor que a ordem de frequência do MCU ( $S < s$ ). A um ciclo completo do MHS executa-se mais de um volta do MCU; e estará no Estado Alto se a ordem de frequência do MHS for maior que a ordem de frequência do MCU ( $S > s$ ). A um ciclo completo do MCU corresponde mais de um ciclo do MHS (Arcela, 1984).

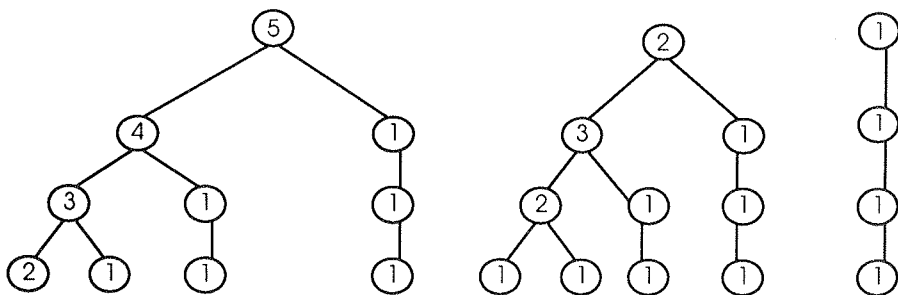


SEGMENTAÇÃO DA LISTA DE DURAÇÕES EM CICLOS DO MHS

A sequência de tempos determinada pelas auto-interseções resultantes do movimento intervalar depende fundamentalmente da razão entre as frequências do intervalo (m:n) do Estado do Intervalo, e da duração inicial do movimento dada pelo tempo gasto até a ocorrência do primeiro momento.

Mas o Objeto Intervalar encerra em seu interior mais informações do que apenas uma sequência de durações. A partir da formação do primeiro ponto, a cada ciclo completo do MHS, um novo conjunto de durações é produzido. Assim, a segmentação das durações em ciclos do MHS, desvenda uma estrutura arborescente, que apresenta na mudança de um nível para outro, a ocorrência de repetições de números ou desdobramentos em dois ou mais.

A organização do intervalo através da segmentação das sequências dá origem a seis espécies de árvores (Primordial, Telescópica, Residual, Especular, Complexa), que podem ser computadas algoritmicamente, prescindindo-se do fenômeno intervalar. As árvores são organizadas em Florestas de acordo com o Estado do objeto. Ao Estado Alto está associada a Floresta Ômega, e ao Estado Baixo a Floresta Alfa (Arcela, 1984). Por exemplo, o intervalo 4:5, no estado baixo, e primeira sequencia, possui a seguinte Floresta de Tempos:



COORDENADAS DE UM MOMENTO DO OBJETO INTERVALAR

A determinação das coordenadas, no espaço tridimensional, de um momento  $PQ$  do objeto intervalar, cujo centro está na origem do sistema de coordenadas, se baseia nas seguintes equações do movimento, para o Estado Alto e Baixo, respectivamente:

$$\begin{aligned} \text{MHS} \quad f(t) &= a \sin((2\pi mt)/\tau + \phi_0) & g(t) &= b \sin((2\pi nt)/\tau + \gamma_0) \\ \text{MCU} \quad \gamma(t) &= \gamma_0 + (2\pi nt)/\tau, R=b & \phi(t) &= \phi_0 + (2\pi mt)/\tau, R=a \end{aligned}$$

onde,  $t$  é o período do movimento,  $\phi_0$  e  $\gamma_0$  fases iniciais. Portanto, as equações que definem as projeções de  $PQ$  sobre os eixos  $x$ ,  $y$  e  $z$  são dadas por:

Estado Baixo	$P0x = b \sin g(t)$
	$P0y = f(t)$
	$P0z = b \cos g(t)$
Estado Alto	$P0x = g(t)$
	$P0y = a \sin f(t)$
	$P0z = a \cos f(t)$

O caso geral, em que um objeto intervalar pode se situar em qualquer lugar do espaço tridimensional requer a aplicação de operações de translação e rotação (através de operação matricial) sobre o objeto na origem do sistema de coordenadas, conduzindo-o à sua posição destino.

CARTAS ESPECTRAIS

Uma nota musical é caracterizada por sua frequência, amplitude, início e duração. Os nodos da Floresta de Tempos possuem esta informação, resultado do próprio movimento intervalar (e que pode ser computada algoritmicamente). O Início da execução da nota pode ser calculado pela soma das durações dos nodos anteriores. A frequência e a amplitude dependem das coordenadas do nodo, e são calculadas por um método denominado de MHS Medidor; que produz a demarcação de dois MCU's pela projeção do vetor velocidade do momento sobre os planos XZ e YZ do sistema de coordenadas. A cada MCU está associado um MHS, cuja frequência é função da velocidade escalar, e cuja amplitude é dada pelo raio do MCU. Portanto, as informações necessárias à construção de uma partitura, composta por uma lista de notas, cada qual demarcando o seu início, duração, frequência e amplitude, são extraídas da Floresta de Tempos. A execução de cada nota é realizada por instrumentos algorítmicos também extraídos da Floresta, e cujo conteúdo espectral está diretamente relacionado com a hierarquia dos nodos. Atualmente esta operação é realizada pelo sistema CARBON, que se constitui no instrumental base para a produção de cartas espectrais

SOM-A

As Cartas Espectrais, por sua vez, servem de base para o cômputo do sinal digital, que enviado a um conversor Digital Analógico, completa o processo de composição algorítmica. Atualmente esta operação é realizada pelo sistema SOM-A, que implementa o método de Síntese Aditiva para produção de sinais amostrados. Utiliza dispositivos espectrais do tipo Oscilador a tabela e Acumulador Estéreo.

FLORESTA DE TEMPOS E O PRECEITO

As informações contidas nas Florestas de tempos são de natureza rítmica e geométrica. Rítmica, pois cada momento (que está associado a um nodo da floresta) possui uma cinemática decorrente do fenômeno intervalar, a partir da qual podem ser computadas a amplitude e frequência de cada momento. Geométrica, pois a floresta define uma hierarquia entre os nodos, que são conectados através de arestas, dando origem a estruturas poliédricas. A extração de partituras naturais computáveis do seu interior, caracteriza o Objeto Intervalar como uma unidade contendo informações de natureza musical.



Os dados necessários ao mapeamento da floresta de tempos em um conjunto de notas e instrumentos musicais são aqueles que caracterizam e unificam o Objeto Intervalar. Tais informações, que recebem o nome de Preceito, norteiam o processo de construção algorítmica do Objeto, e são as seguintes: as ordens de frequência (m,n), as amplitudes (a,b), as fases iniciais do MHS e do MCU, o Estado (alto ou baixo), os deslocamentos e as rotações do Objeto em relação aos eixos X,Y e Z (Arcela, 1984).

#### FORMA INACABADA E A NATUREZA GEOMÉTRICA DAS FLORESTAS

Dependendo do posicionamento da Floresta de Tempos no espaço, pode-se obter diferentes melodias, porém todas com o mesmo padrão rítmico, e a primeira impressão é que constituem melodias inacabadas, necessitando um complemento, ou acabamento da forma melódica. Por outro lado, a observação das estruturas geométricas obtidas pela conexão dos momentos do objeto intervalar por arestas de acordo com a hierarquia pai-filho, também indica uma forma com partes inacabadas, formando uma estrutura poliédrica aberta.

É através da relação hierárquica dos nodos da floresta de tempos que surge a sua natureza geométrica / poliédrica, pois os nodos da floresta que apresentam a relação pai-filho são conectados por uma aresta. Tal relação indica que podem haver diferentes tipos de vértices na floresta em função do número de arestas que a ele convergem. Uma vez que os nodos das árvores que formam a floresta podem ser desdobrados no máximo em três outros nodos, verifica-se que o número máximo de arestas que podem convergir a um vértice é três. Como consequência, pode-se classificar os vértices em completos (contendo 3 arestas não-coincidentes e não coplanares) e vértices incompletos (contendo duas ou menos arestas não coincidentes) (Ramalho, 1992).

#### ACABAMENTO DE UM VÉRTICE / ACOPLAMENTO GEOMÉTRICO

Baseando-se no fato de que toda floresta de tempos constitui uma estrutura poliédrica aberta (contendo vértices incompletos), define-se o acabamento de um vértice como sendo a sua passagem para a condição de vértice completo, através do acoplamento geométrico entre florestas, cujo objetivo é produzir o compartilhamento no espaço de vértices das florestas acopladas. A mudança de um vértice incompleto para a condição de vértice completo, depende da sua valência, que é o número de vértices necessários para o vértice se tornar completo.

A possibilidade de um vértice incompleto passar à condição de vértice completo decorre do fato de que posicionados no mesmo lugar do espaço passam a compartilhar as suas arestas, implicando na mudança da valência de ambos. Através de operações algébricas de translação, rotação e escala (Foley, 1984) sobre uma das florestas a serem conectadas (floresta móvel), consegue-se promover conexões. A aceitação do acasalamento depende da não violação do critério do estado (as dimensões finais da floresta móvel não podem implicar em mudança do estado do objeto) e da submissão das florestas envolvidas na regra de crescimento (que tenta garantir um grau de convergência da abertura/fechamento da estrutura poliédrica resultante) (Ramalho, 1992).

Uma vez que cada Objeto Intervalar é caracterizado pelo seu Preceito, após o acoplamento geométrico entre Objeto distintos, o Preceito da Forma resultante é composto pelos Preceitos das Florestas que foram conectadas à forma resultante. Portanto, através do acoplamento geométrico é possível a construção de formas sônicas de rara complexidade. As diversas possibilidades de posicionamento do Objeto intervalar no espaço euclidiano abrem uma infinidade de possibilidades de padrões rítmicos, bem como diversas possibilidades de acoplamento. Assim, neste processo, a identificação da posição mais adequada para a floresta fixa, além da seleção dos vértices que farão parte das conexões, indicam a necessidade de um ambiente computacional capaz de dar suporte tanto para a construção de Objetos Intervalares a partir de Preceitos, bem como monitorar e oferecer informações que possam nortear o processo de composição algorítmica baseado no acoplamento geométrico de Objeto Intervalares.

#### EDITOR DE PRECEITOS INTERVALARES

O Editor de Preceitos Intervalares é um ambiente de computação gráfica projetado para manter de forma coerente e organizada o universo de informações relativas a Objetos Intervalares, bem como manter estruturas de dados capazes de suportar mais de um Objeto compartilhando o espaço euclidiano, passíveis de serem acoplados geometricamente. O sistema se propõe a dar suporte ao cômputo de informações que possam nortear a escolha do melhor posicionamento da floresta fixa, bem como orientar a escolha dos vértices que farão parte do acoplamento, possibilitando ao pesquisador investigar as possibilidades de conexão de forma ágil e interativa. Todo a funcionalidade oferecida neste processo tem como objetivo, em última análise, a

extração do Preceito resultante dos acoplamentos realizados, que constitui atualmente a entrada do sistema CARBON, que implementa o mapeamento das Florestas de Tempos em Carta Espectral.

Uma vez que o Editor realiza o cômputo da quase totalidade das informações relativas aos Objetos Intervalares, seu segundo objetivo é servir como um ambiente de apoio aos alunos de Pós-graduação, com especialização em processamento de sinais e computação sônica. A construção passo a passo de Objetos Intervalares, através da montagem das suas partes constituintes (cilindro base do Objeto, traçado da curva relativa ao movimento intervalar e suas projeções nos planos XZ e YZ, a determinação dos momentos associados às auto-interseções, a conexão dos momentos por arestas de acordo com estrutura hierárquica dada pela Floresta de Tempos; a representação gráfica da própria Floresta, e a matriz cilíndrica com as durações) possibilitará uma dinâmica maior ao processo de aprendizado da teoria das Árvores de Tempos.

Um terceiro aspecto do sistema é a capacidade de se verificar a representação auditiva dos Objetos Intervalares selecionados, uma vez que o sistema SOM-A será parte integrante do Editor, funcionando como um subsistema gerador de sinais amostrados. Esta integração poderá ser realizada pelo acoplamento de algumas das funções de mapeamento de Florestas de Tempos em cartas espectrais contempladas pelo sistema CARBON. Desta forma, o editor se constituirá num ambiente integrado capaz de oferecer suporte às diversas operações sobre Objetos Intervalares (construção, visualização da representação tridimensional, simulação do movimento intervalar, verificação do sinal correspondente, acoplamento geométrico monitorado e extração do Preceito relativo à forma resultante).

O ambiente operacional selecionado para a versão atual é a plataforma Windows, uma vez que o sistema necessita de suporte a interfaces gráficas para usuário (GUI). Além disso, o ambiente orientado a janelas possibilita a organização mais racional das diversas visões que se pretende oferecer, bem como permitir um grau de liberdade na criação das janelas pelo usuário. E um fator relevante na escolha deste ambiente está associado à característica de orientação a objetos da programação sobre o Windows, propício à implementação de problemas de alta complexidade.

#### REFERÊNCIAS BIBLIOGRÁFICAS

- ARCELA, A. As Árvores de Tempos e a Configuração Genética dos Intervalos Musicais. *Tese de Doutorado*, PUC, Rio de Janeiro, 1984.
- RAMALHO, G. L. Um Sistema de Geração de Teoremas Musicais baseado na Geometria das Árvores de Tempos. *Tese de Mestrado*, CIC-UnB, Brasília, 1992.
- FOLEY, J. D. *Fundamentals of interactive computer graphics*, New York, 1984.

## The SmOKe music representation, description language, and interchange format

Stephen Travis Pope  
 The Nomad Group, *Computer Music Journal*, CCRMA  
 P. O. Box 60632, Palo Alto, California 94306 USA  
 Electronic Mail: stp@CCRMA.Stanford.edu

### ABSTRACT

The Smallmusic Object Kernel (*SmOKe*) is an object-oriented representation, description language and interchange format for musical parameters, events, and structures. The author believes this representation, and its proposed linear ASCII description, to be well-suited as a basis for: (1) concrete description interfaces in other languages, (2) specially-designed binary storage and interchange formats, and (3) use within and between interactive multimedia, hypermedia applications in several application domains.

The textual versions of SmOKe share the terseness of note-list-oriented music input languages, the flexibility and extensibility of "real" music programming languages, and the non-sequential description and annotation features of hypermedia description formats. This description presents the requirements and motivations for the design of the representation language, defines its basic concepts and constructs, and presents examples of the music magnitudes and event structures. The intended audience for this discussion is programmers and musicians working with digital-technology-based multimedia tools who are interested in the design issues related to music representations, and are familiar with the basic concepts of software engineering. Two other documents ([Smallmusic 1992] and [Pope 1992]), describe the SmOKe language, and the MODE environment within which it has been implemented, in more detail.

### 1: INTRODUCTION

The desire has been voiced (ANSI 1992; Dannenberg et al. 1989; MusRep 1987; MusRep 1990; Smallmusic 1992), for an expressive, flexible, abstract, and portable structured music description and composition language. The goal is to develop a kernel description that can be used for structured composition, real-time performance, processing of performance data, and analysis. It should support the text input or programmatic generation and manipulation of complex musical surfaces and structures, and their capture and performance in real time via diverse media. The required language have a simple, consistent syntax that provides for readable complex nested expressions with a minimum number of different constructs. The test of the language and its underlying representation will be the facility with which applications can be ported to it.

Smallmusic Object Kernel consists of primitives for describing the basic scalar magnitudes of musical objects, abstractions for musical events and event lists, and standard messages for building event list hierarchies and networks. Structures in the underlying music representation can be described in a text-based linear format, and in terms of the of in-memory data structures that might be used to hold them. It is intended that independent parties be able to implement compatible abstract data structures, concrete interchange formats, and description languages based on the formal definition of SmOKe (Smallmusic 1992).

The naming conventions and the code description examples use the Smalltalk-80 programming language (Goldberg and Robson 1989), but the representation should be easily manipulable in any object-oriented programming language. For readers unfamiliar with Smalltalk-80, another document (available via InterNet ftp from the file named "reading.smalltalk.t" in the directory "anonymous@ccrma-ftp.Stanford.edu/pub/st80"), introduces the language's concepts and syntax to facilitate the reading of the code examples in the text. In this document, SmOKe examples are written between square brackets in *sans-serif italic* typeface.

### 2: REQUIREMENTS AND MOTIVATIONS

Several of the groups that have worked on developing music representations have started by drawing up lists of requirements on such a design, and separating out which items are truly determined by the underlying representation, and which are interface or application issues. The Smallmusic group developed the following list, using the results of several previous attempts (see citations above) as input. SmOKe shall provide or support:

- abstract models of the basic musical quantities (scalar magnitudes such as pitch, loudness or duration);
- sound functions, granular description, or other (non-note-oriented) description abstractions;
- flexible grain-size of "events" in terms of "notes," "grains," "elements," or "textures";
- description/manipulation levels including event, control, and sampled function;
- hierarchical event-tree (nested lists) for "parts," "tracks," or other parallel or sequential structures;
- separation of "data" from "interpretation" (what vs. how in terms of having interpretation objects such as the instru-

- ment/note, voice/event, or performer/part abstractions);
- abstractions for the description of "middle-level" musical structures (e.g., chords, clusters, or trills);
- annotation of events supporting the creation of heterarchies (lattices) and hypermedia networks;
- annotation including common-practise notation possible (application issue);
- description of sampled sound synthesis and processing models such as sound file mixing or DSP;
- possibility of building convertors for many common formats, such as MIDI data, Adagio, note lists, HyTime, DSP code, instrument definitions, mixing scripts; and
- possibility of parsing live performance into some rendition in the representation, and of interpreting it (in some rendition) in real-time (application issue related to simplicity, terseness, etc.).

### 3: EXECUTIVE SUMMARY

The SmOke representation can be summarized as follows. Music (i.e., a musical surface or structure), can be represented as a series of "events" (which generally last from tens of msec to tens of sec). Events are simply property lists or dictionaries; they can have named properties whose values are arbitrary. These properties may be music-specific objects (such as pitches or spatial positions), and models of many common musical magnitudes are provided. Events are grouped into event lists as records consisting of relative start times and events. Event lists are events themselves and can therefore be nested into trees (i.e., an event list can have another event list as one of its events, etc.); they can also map their properties onto their component events. This means that an event can be "shared" by being in more than one event list at different relative start times and with different properties mapped onto it. Events and event lists are "performed" by the action of a scheduler passing them to an interpretation object or voice. Voice objects and applications determine the interpretation of events' properties, and may use "standard" property names such as pitch, loudness, voice, duration, or position. Voices map event properties onto parameters of I/O devices; there can be a rich hierarchy of them. A scheduler expands and/or maps event lists and sends their events to their voices. Stored data functions can be defined and manipulated as breakpoint or summation parameters, "raw" data elements, or functions of the above. Sampled sounds are also describable, by means of synthesis "patches," or signal processing scripts involving a vocabulary of sound manipulation messages.

### 4: THE SmOke LANGUAGE

#### 4.1 Linear Description Language

The SmOke music representation can be linearized easily in the form of immediate object descriptions and message expressions. These descriptions can be thought of as being declarative (in the sense of static data definitions), or procedural (in the sense of messages sent to class "factory" objects). A text file can be freely edited as a data structure, but one can compile it with the Smalltalk-80 compiler to "instantiate" the objects (rather than needing a special formatted reading function). The post-fix expression format taken from Smalltalk-80 (*receiverObject keyword: optionalArgument*) is easily parseable in C++, Lisp, Forth, and other languages.

#### 4.2 Language Requirements

The basic representation itself is language-independent, but assumes that the following immediate types are representable as ASCII/ISO character strings in the host language:

- arbitrary precision integers (at least very large),
- integer fractions (i.e., stored as numerator/denominator, rather than the resulting whole or real number),
- 32- (and 64-bit) (7-, 12-place precision) floating-point numbers,
- arbitrary-length ASCII/ISO strings,
- unique symbols (i.e., strings managed with a hash table),
- 2- and 3-dimensional points (or n-dimensional complex numbers) (axial or polar representation), and
- functions of one or more variables described as breakpoints for linear, exponential or spline interpolation, Fourier sums, series, sample spaces, and probability distributions.

The support of block context objects (in Smalltalk), or closures (in LISP), is defined as being optional, though it is considered important for complex scores, which will often need to be stored with interesting behavioral information. (It is beyond the scope of the present design to propose a metalanguage for the interchange of algorithms.) Dictionaries or property association lists must also either be available in the host language or be implemented in a support library (as must unique symbols and even associations in some cases [e.g., std. C]).

#### 4.3 Naming and Persistency

The names of abstract classes are known and are treated as special globals. The names of abstract classes are used wherever possible, and instances of concrete subclasses are returned, as in [*Pitch value: 'c3'*] or [*'c3' pitch*] both returning a Symbol-

pitch instance (strings are written between single-quotes in SmOke [and Smalltalk]). All central classes are assumed to support "persistency through naming" whereby any object that is explicitly named gets stored in a global dictionary under that name until explicitly released. What the exact (temporal) scope of the persistency is, is not defined here. The (lexical) extent is assumed to be as SmOke "document" or "module."

#### 4.4 Score Format

A SmOke score consists of one or more parallel or sequential event lists whose events may have interesting properties and links. Magnitudes, events, and event lists are described using class messages that create instances, or using immediate objects and these post-fix operators. These can be named, used in one or more event lists, and their properties can change over time. There is no pre-defined "level" or "grain-size" of events; they can be used at the level of notes or envelope components, patterns, notes, etc. The same applies to event lists, which can be used in parallel or sequentially to manipulate the sub-sounds of a complex "note," or as "motives," "tracks," or "parts." Viewed as a document, a score consists of declarations of, or messages to, events, event lists and other SmOke structures. It can resemble a note list file or a DSP program. It is structured as executable Smalltalk-80 expressions, and can define one or more "root-level" event lists. There is no "section" or "wait" primitive; sections that are supposed to be sequential must be included in some higher-level event list to declare that sequence. A typical score will define and name a top-level event list, and then add sections and parts to it in different segments of the document.

### 5: SmOke MUSIC MAGNITUDES

Abstract descriptive models for the basic music-specific magnitude types such as pitch, loudness or duration are the foundation of SmOke. These are similar to Smalltalk-80 magnitude objects in that they represent partially- or fully-ordered scalar or vector quantities with (e.g.) numerical or symbolic values. Some of their behavior depends on what they stand for, and some of it on how they're stored. These two aspects are the objects' "species" and their "class." The pitch-species objects *440.0 Hz* and *'c#3'*, for example, share some behavior, and can be mixed in arithmetic with (ideally) no loss of "precision." The expression *(261.26 Hz + MIDI key number 8)* should be handled differently than *(1/4 beat + 80 msec)*. The class of a music magnitude depends on the "type" of their values (e.g., floating-point numbers or strings), while their species denote what they represent. Only the species is visible to the user.

Music magnitudes can be described using prefix class names or post-fix type operators, e.g., [*Pitch value: 440.0*] or [*440.0 Hz*] or [*'a3' pitch*], [*Amplitude value: 0.7071*] or [*-3 dB*]. The representation and interchange formats should support all manner of weird mixed-mode music magnitude expressions (e.g., [*'c4' pitch*] + (*78 cents*) + (*12 Hz*)), with "reasonable" assumptions as to the semantics of the operation (coerce to Hz or cents?). Applications for this representation should support interactive editors for music magnitude objects that support the manipulation of the basic hierarchy described below, as well as its extension via "light-weight" programming.

The Figure on the next page shows the class hierarchy of the model classes—those used for the species (i.e., representation)—on the left side, and the partial hierarchy of the concrete (implementation) classes on the right. The class inheritance hierarchy is denoted by the order and indentation of the list. The lines indicate the species relationships of several of the common music magnitudes. The examples below demonstrate the verbose (class message) and terse (value + post-operator) forms of music magnitude description. Note that comments are delineated by double quotes (or curly braces) in SmOke.

#### Music Magnitude Examples

(Duration value: 1/16) asMS	"Same as [1/16 beat]; answers 62."
(Pitch value: 36) asHertz	"Same as [36 pitch]; answers 261.623."
(Amplitude value: 'ff') asMidi	"Or [#ff ampl]; answers 106."
('f#4' pitch), ('pp' dynamic)	"Terse examples; value + post-op."
(261 Hz), (1/4 beat), (-38 dB), (56 velocity), (2381 msec), (0@0 position)	

### 6: SmOke EVENT OBJECTS

The simplest view of events is as Lisp-esque property lists, dictionaries of property names and values, the relevance and interpretation of whom is left up to others (voices and applications). Events need not be thought of as mapping one-to-one to "notes," though they should be able to faithfully represent note-level objects. There may be one-to-many or many-to-one relationships between events and "notes." Events may have arbitrary properties, some of whom will be common to most musical note-level events (such as duration, pitch or loudness), while others may be used more rarely or only for non-musical events.

Events' properties can be accessed as keyed dictionary items (i.e., events can be treated as record data structures), or as direct behaviors (i.e., events can be thought of as purely programmatic). One can set an event to be "blue" (for example), by saying [*anEvent at: #color put: #blue* 'dictionary-style accessing'] or more simply [*anEvent color: #blue* "behavioral access-PT"] (whereby #string means unique symbol whose name is string). Events can be linked together by having properties that are associations to other events or event lists, (as in [*anEvent #soundsLike: otherEvent*]), enabling the creation of annotated hy-

permedia networks of events. Event properties can also be active blocks or procedures (in cases where the system supports compilation at run-time as in Smalltalk-80 or Lisp), blurring the differentiation between events and "active agents." Events are created either by messages sent to the class Event (which may be a macro or binding to another class), or more tersely, simply by the concatenation of music magnitudes using the message "," (comma for concatenation), as shown in the examples below. Applications should enable users to interactively edit the property lists of objects, and to browse event networks via their time or their links using flexible link description and filtering editors. Standard properties such as pitch, duration, position, amplitude, and voice are manipulated according to "standard" semantics by many applications.

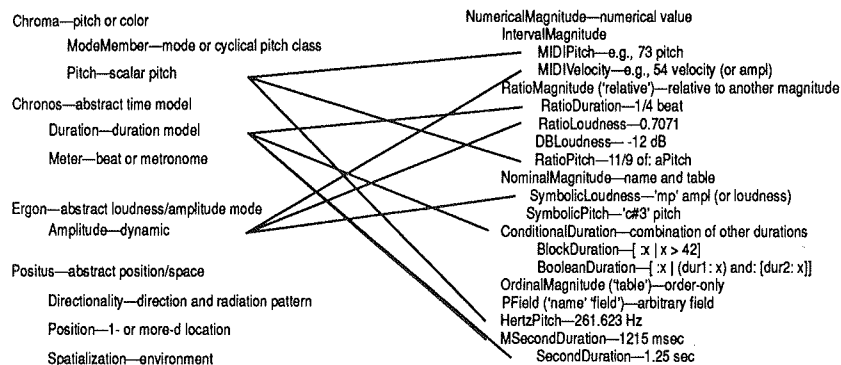


Figure 1: SmOke Music Magnitude Model Abstractions and Implementation Classes

Event Examples

```

"Event creation examples--the verbose way (class messages)."
[event := (Event newNamed: #flash) color: #white; place: #there]
[(Event duration: (Duration value: 1/2) pitch: (Pitch value: #c2)
loudness: (Loudness value: #mf) playOn: aVoice)
"Create three events with mixed properties--the terse way"
[(440 Hz, (1/4 beat), (-12 dB), Voice default)
[38 key, 280 ticks, 56 vel, (#voice -> 4)]
[(#c4 pitch, 0.21 sec, 0.37 ampl, (Voice named: #oboe)]
"Create a named link between two events."
[event1 isLouderThan: event2]
    
```

7: EVENT LISTS

Events are grouped into collections—event lists—where a list is composed of associations between start times (durations starting at the start time of the event list) and events or sub-lists (nested to any depth). Schematically, this looks like: (EventList = (dur1 => event1), (dur2 => event2), ...) where (x => y) means association with key x and value y. Event lists can also have their own properties, and can map these onto their events eagerly (at definition time) or lazily (at "performance" time); they have all the property and link behavior, and special behaviors for mapping with voices and event modifiers. Event lists can be named, and when they are, they become persistent (until explicitly erased within a document or session).

The messages [anEventList add: anAssociation] and [anEventList add: anEventOrEventList at: aDuration], along with the corresponding event removal messages, can be used for manipulating event lists in the static representation or in applications. If the key of the argument to the add: message is a number (rather than a duration), it is assumed to be the value of a duration in seconds or milliseconds, as "appropriate." Event lists also respond to Smalltalk-80 collection-style control structure messages such as [anEventList collect: aSelectionBlock] or [anEventList select: aSelectionBlock], though this requires the representation of contexts/closures. The behaviors for applying functions (see below) to the components of event lists can look applicative (e.g., [anEventList apply: aFunction to: aPropertyName]), or one can use event modifier objects to have a stateful representation of the mapping. Applications will use event list hierarchies for browsing and annotation as well as for score following and performance control. The use of standard link types for such applications as version control (with such link types as #usedToBe or #viaScript11b5i4), is defined by applications and voices.

A named event list is created (and stored) in the first example below, and two event associations are added to it, one starting at 0 (seconds by default), and the second at 1 sec. Note that the two events can have different types of properties, and the handy

creation messages such as [dur: durationValue pitch: pitchValue amp: ampValue]. The second example is the terse concatenation of event list declaration using the behavior of (duration => event) associations such that [(aMagnitude) => (anImmeasurableDictionary)] returns the association [(duration with value aMagnitude) => (Event with given property dictionary)]. One can use dictionary-style shorthand with event associations to create event lists, as in the very terse way of creating an anonymous (non-persistent) list with two events in the second example. The third example shows the first few notes from the c-minor fugue from The Well-Tempered Clavichord in which the first note begins after a rest (that could also be represented explicitly as an event with a duration and no other properties). Note that there is one extra level of parentheses for readability.

Event List Examples

```

Event Lists the verbose way"
[eventList newNamed: #test1] add: (0 => (Event dur: 1/4 pitch: 'c3' ampl: 'mf'));
[eventList newNamed: #test1] add: (1 => ((Event new) dur: 6.0 ampl: 0.3772 sound: #s73bw))

Terse Lists--concatenation of events or (dur => event) associations."
[(440 Hz, (1/1 beat), 44.7 dB), (1 => ((1.396 sec, 0.714 ampl) sound: #s73bw; phoneme: #xu))]

c-minor fugue theme."
"start time duration pitch voice"
( (0.5 beat => ((1/4 beat), ('c3' pitch), (voice: 'harpsichord')),
((1/4 beat), ('b2' pitch)), ((1/2 beat), ('c3' pitch)),
((1/2 beat), ('g2' pitch)), ((1/2 beat), ('a-flat2' pitch)))
    
```

8: EVENT GENERATORS AND MODIFIERS

The EventGenerator and EventModifier packages provide for music description and performance using generic or composition-specific middle-level objects. Event generators are used to represent the common structures of the musical vocabulary such as chords, ostinati, or compositional algorithms. Each event generator subclass knows how it is described—e.g., a chord with a rest and an inversion, or an ostinato with an event list and repeat rate—and can perform itself once or repeatedly, acting like a control structure, or a process, as appropriate. EventModifier objects hold onto a function and a property name; they can be told to apply their functions to any property of an event list lazily or eagerly. Event generators and modifiers are described elsewhere.

9: FUNCTIONS, PROBABILITY DISTRIBUTIONS AND SOUNDS

SmOke also defines functions of one or more variables, several types of discrete or continuous probability distributions, and granular and sampled sounds. The description of these facilities is, however, outside the scope of this paper, and the reader is referred to (Smallmusic 1992).

10: VOICES AND STRUCTURE ACCESSORS

The "performance" of events takes place via Voice objects. Event properties are assumed to be independent of the parameters of any synthesis instrument or algorithm. A voice object is a "property-to-parameter mapper" that knows about one or more output or input formats for SmOke data (e.g., MIDI, note list files, or DSP commands). A StructureAccessor is an object that acts as a translator or protocol convertor. An example might be an accessor that responds to the typical messages of a tree node member of a hierarchy (e.g., What's your name? Do you have any children/sub-nodes? Who are they? Add this child to the tree) and that knows how to apply that language to navigate through a hierarchical event list (by querying the event list's hierarchy). SmOke supports the description of voices and structure accessors in scores so that performance information or alternative interfaces can be embedded. The goal is to be able to annotate a score with possibly complex real-time control objects that manipulate its structure or interpretation. Voices and event interpretation are described in (Pope 1992).

12: SmOke SCORE EXAMPLE

In SmOke scores, sections with declarations of variables, naming of event lists, event definition, functions and event modifiers, and annotation, can be freely mixed. Note that one tries to avoid actually typing SmOke at all anyway, leaving that to interactive graphical editors, algorithmic generation or manipulation programs, or read/write interfaces to other media, such as MIDI. The example below shows the components of a SmOke score for a composition with several sections declared in different styles. Variable name declarations are placed between vertical bars.

```

"declarations of variable names and top-level event list."
| piece section1 section2 | "name declarations--optional but advised."
piece := EventList newNamed: #piece.
"section 1--verbose, add events using add:at: message."
section1 := EventList newNamed: #section1.
section1 add: (...first event (may have many properties)...) at: 0.
section1 add: (...second event...) at: 0. "starts with a chord."
"...section 1 events, in parallel or sequentially..."
    
```

```

"section 2--terse, add event assoc. using ',' concatenation operator."
section2 := ((0 beat) => (...event1...)), ((1/4 beat) => (event2)),
"...section 2 events...",
((2109/4 beats) => (event3308)).
"Event list composition (may be placed anywhere)"
piece add: section1; add: section2. "add the sections in sequence."
piece add: (Event duration: (4/1 beat)). "add one measure of rest after section 2."
"Add a section from data arrays."
piece add: (EventListwithProperties: #(duration: loudness: pitch:)
values: (Array with: #(250 270 230 120 260 260 ... ) "duration"
values: (Array with: #('mp') "loudness"
values: (Array with: #('c3' 'd' 'e' 'g' ... ))). "pitch")
"Add an event with the given samples (you want low-level? we got low-level!)"
piece add: (Event rate: 44100 channels: 1 samples: #(0 121 184 327 441 ... )).
"Declare global (named) event modifiers, functions, etc."
(Rubato newNamed: #tempo) function: (...tempo spline function...) property: #startTime.
piece tempo: (Rubato named: #tempo).
"Optionally declare voices, accessors, other modifiers, etc."

```

### 13: CONCLUSIONS

The Smallmusic Object Kernel (SmOke) is a representation, description language and interchange format for musical data that eases the creation of concrete description interfaces, the definition of storage and interchange formats, and is suitable for use in multimedia, hypermedia applications. The SmOke description format has several versions, ranging from very readable to very terse, and covering a wide range of signal, event, and structure types from sampled sounds to compositional algorithms. SmOke can be viewed as a procedural or a declarative description; it has been designed and implemented using an object-oriented methodology and is being tested in several applications. More explicit documents describing SmOke, and the Smalltalk-80 implementation of the system in the MODE system, are freely available via Internet file transfer.

### 14: ACKNOWLEDGEMENTS

SmOke, and the MODE of which it is a part, is the work of many people. Craig Latta and Daniel Oppenheim came up with the names Smallmusic and SmOke. These two, and Guy Garnett and Jeff Goms, were part of the team that discussed the design of SmOke, and commented on its design documents (Smallmusic 1992).

### REFERENCES

- ANSI 1992 *Journal of Technical Development*, ANSI Working Group X3V1.8MSD-7 (now ISO/IEC DIS 10744).
- R. B. Dannenberg, L. Dyer, G. E. Garnett, S. T. Pope, and C. Roads, "Position Papers for a Panel on Music Representation," *Proc. of the ICMC*, San Francisco: ICMA, 1989.
- A. Goldberg and D. Robson, *Smalltalk-80: The Language*, (revised and updated from 1983 edition). Menlo Park: Addison-Wesley, 1989.
- MusRep 1986, R. Dannenberg, J. Maloney, et al., "Network electronic discussion on music representation" USENET
- MusRep 1990, G. Diener, L. Dyer, G. E. Garnett, D. Oppenheim, S. T. Pope et al., "Notes of meetings on music representation at CCRMA", Fall, 1990.
- Newcomb, N. A. Kipp, and V. T. Newcomb. "The HyTime Hypermedia/Time-based Document Structuring Language," *Communications of the ACM*, vol. 34, no. 11, pp. 67-83
- S. T. Pope, "Interim DynaPiano: An Integrated Computer Tool and Instrument for Composers," *Computer Music Journal* 16:3, Fall, 1992.
- Smallmusic 1991. G. E. Garnett, J. Goms, C. Latta, D. Oppenheim, S. T. Pope et al., Smallmusic discussion group notes, Credo 1-6 documents (from which this document was derived), and MODE User Primitive specification available from Smallmusic@XCF.Berkeley.edu as email or via anonymous InterNet ftp from the server ccrma-ftp.Stanford.edu in the directory pub/st80 (see the README file there).

## Análise Musical, Educação

## Learning Counterpoint Rules for Analysis and Generation

EDUARDO MORALES M.

*ITESM - Campus Morelos, Apto. Postal C-99,  
Cuernavaca, Morelos, 62050, México  
email: emorales@rs970.mor.itesm.mx*

ROBERTO MORALES-MANZANARES

*Laboratorio de Informática Musical (LIM)  
Escuela de Música de la Universidad de Guanajuato  
Universidad de Guanajuato  
Centro de Investigaciones en Matemáticas (CIMAT)  
Paseo de la Presa # 152  
Guanajuato, Gto., México  
email: roberto@kaliman.cimat.conacyt.mx*

### Abstract

History in composition and analysis have shown that composers using the same patterns in structure and harmony get different results depending on the way these patterns are resolved. In terms of musical analysis, a particular piece can be described by a sequence of states and transitions between states representing the personal criteria that each composer pursues when solving a musical structure. A first-order learning system, called Pal, is used to learn transition criteria for counterpoint analysis, in the form of Horn clauses from pairs of musical states (given as sets of notes) and general purpose musical knowledge. It is shown how the rules learned by Pal can be used for musical analysis of simple two-voice counterpoint pieces. Similarly, a counterpoint voice can be generated from a single voice (*cantus firmus*) using the learned rules. Conclusions and future research directions are given.

## 1 Introduction

Musical composition generates symbolic representations (i.e., musical scores) of musical ideas. Such ideas are based on subjective temporal interpretations of auditive events. The events are characterized by their frequency, amplitude and its envelope (which determines the quality of tone or pitch). Such elements, which define the musical characteristics of the musical instruments, are part of the material which a composer uses to propose its aesthetic solutions. During this process, a composer can follow a set of implicit or explicit rules to guide his/her preferences and express his/her ideas. Our goal is to induce musical criteria rules that can be used for musical analysis and generation. As a first step, we looked at counterpoint analysis, which is well understood and defined with a finite set of known rules (Fux, 1971). Counterpoint rules can be expressed in a compact and understandable way using first-order logic. In general, musical rules express relations between notes. In order to learn such rules we used Pal (Morales, 1991; Morales, 1992a) an Inductive Logic Programming (ILP) system (Muggleton, 1991) capable of learning a subset of Horn clauses from examples and background knowledge expressed as logic programs. It is shown how Pal can learn counterpoint rules of the first specie (to be defined below) and used them for musical analysis and generation of counterpoint notes from *cantus firmus* (a sequence of single notes to which counterpoint rules are applied to generate harmonic notes). The constraints used

by Pal to guide its inductive search for hypothesis are discussed, in the context of music and in general to other areas where Pal has been used.

Section 2 describes some musical concepts required for counterpoint analysis. Section 3 provides some definitions from logic. The concepts and notation will be used in the sections to follow. Section 4 briefly describes an ILP inductive framework and Pal. Section 5 shows the main results and finally conclusions and future work are given in section 6.

## 2 Musical background

The concept of musical counterpoint emerge in the 14th. century and evolve up to *Gradus ad Parnassum* by Johann Joseph Fux published in 1725 (Fux, 1971). This is the first book which synthesizes in form of rules the art of polyphony considered to be correct by that time. Those rules can be considered as the culmination of musical analysis from the 14th. until 18th. century.

Counterpoint rules can be classified between two voices (sequences of notes) into several species, according to the number of notes involved at the same time on each voice:

- 1st: one note on one voice against one note on the other
- 2nd: two notes on one voice against one note on the other
- 3rd: four notes on one voice against one note on the other
- 4th: a whole note (of four times) on one voice against half notes (of two times) on the other
- 5th or florid: three or more notes in combination with the previous species

Our goal is to obtain similar rules as those described by Fux from examples of musical pieces and basic musical knowledge from traditional music. Such knowledge includes the classification of intervals (distances in height between two notes) into: *consonances* and *dissonances*. *Unison*, *fifth* and *octave* are *perfect consonances* while *third* (major and minor) and *sixth* (major and minor) are *imperfect consonance*. *Second* (major and minor), *fourth*, *augmented fourth*, *diminished fifth* and *seventh* (major and minor) are *dissonances*.

These are the elements which account for all harmony in music. The purpose of harmony is to give pleasure by variety of sounds through progressions from one interval to another. Progression is achieved by motion, denoting the distance covered in passing from one interval to another in either direction, up or down. This can occur in three ways: direct, contrary and oblique:

- *direct motion*: results when two or more parts ascend or descend in the same direction
- *contrary motion*: results when one part ascends and the other descends, or vice versa.
- *oblique motion*: results when one part moves while the other remains stationary

With these concepts the counterpoint rules of 1st. specie are defined as follows:

**First rule:** from one perfect consonance to perfect consonance one must proceed in contrary or oblique motion

**Second rule:** from a perfect consonance to an imperfect consonance one may proceed in any of the three motions

**Third rule:** from an imperfect consonance to a perfect consonance one must proceed in contrary or oblique motion

**Fourth rule:** from one imperfect consonance to another imperfect consonance one may proceed in any of the three motions

In section 4 it is shown how these rules are learned from a small set of examples, represented as pairs of notes in two voices, and general musical knowledge about musical intervals. First we give a short description of Pal.

## 3 Preliminaries

A *variable* is represented by a string of letters and digits starting with an upper case letter. A *function symbol* is a lower case letter followed by a string of letters and digits. A *predicate symbol* is a lower case letter followed by a string of letters and digits. A *term* is a constant, variable or the application of a function symbol to the appropriate number of terms. An *atom* or *atomic formula* is the application of a predicate symbol to the appropriate number of terms. A *literal* is an atom or the negation of an atom. Two literals are *compatible* if they have the same symbol, name and number of arguments. The negation symbol is  $\neg$ . A *clause* is a disjunction of a finite set of literals, which can be represented as  $\{A_1, A_2, \dots, A_n, \neg B_1, \dots, \neg B_m\}$ . The following notation is equivalent:

$$A_1, A_2, \dots, A_n \leftarrow B_1, B_2, \dots, B_m.$$

A *Horn clause* is a clause with at most one positive literal (e.g.,  $H \leftarrow B_1, \dots, B_m$ ). The positive literal ( $H$ ) is called the *head*, the negative literals (all  $B_i$ s) the *body*. A clause with empty body is a *unit clause*. A set of Horn clauses is a *logic program*.  $F_1$  *syntactically entails*  $F_2$  (or  $F_1 \vdash F_2$ ) iff  $F_2$  can be derived from  $F_1$  using the deductive inference rules. A *substitution*  $\Theta = \{V_1/t_1, V_2/t_2, \dots, V_n/t_n\}$  consists of a finite sequence of distinct variables paired with terms. An *instance* of a clause  $C$  with substitution  $\Theta$ , represented by  $C\Theta$ , is obtained by simultaneously replacing each occurrence of a component variable of  $\Theta$  in  $C$  by its corresponding term. A *model* of a logic program is an interpretation for which the clauses express true statements. We say that  $F_1$  *semantically entails*  $F_2$  (or  $F_1 \models F_2$ , also  $F_1$  logically implies or entails  $F_2$ , or  $F_2$  is a logical consequence of  $F_1$ ), iff every model of  $F_1$  is a model of  $F_2$ .

## 4 Pal

Inductive Logic Programming (ILP) is a fast growing research area which combines Logic Programming and Machine Learning (Muggleton, 1991). The general setting for ILP is, given a background knowledge  $\mathcal{K}$  (in the form of first-order clauses) and sets of positive ( $\mathcal{E}^+$ ) and negative ( $\mathcal{E}^-$ ) examples, find a hypothesis  $\mathcal{H}$  (another set of clauses) for which  $\mathcal{K} \wedge \mathcal{H} \vdash \mathcal{E}^+$  and  $\mathcal{K} \wedge \mathcal{H} \not\vdash \mathcal{E}^-$ . That is, find a hypothesis which can explain the data in the sense that all the positive ( $\mathcal{E}^+$ ) but none of the negative ( $\mathcal{E}^-$ ) examples can be deduced from the hypothesis and the background knowledge. This inductive process can be seen as a search for logic programs over the hypothesis space and several constraints have been imposed to limit this space and guide the search. For learning to take place efficiently, it is often crucial to structure the hypothesis space. This can be done with a model of generalization. Searching for hypothesis can then be seen as searching for more general clauses given a known specialized clause.

Plotkin (Plotkin, 1969; Plotkin 1971a; Plotkin, 1971b) was the first to study in a rigorous manner the notion of generalization based on  $\Theta$ -subsumption. Clause  $C\Theta$  subsumes clause  $D$  iff there exists a substitution  $\sigma$  such that  $C\sigma \subseteq D$ . Clause  $C_1$  is more general than clause  $C_2$  if  $C_1 \Theta$ -subsumes  $C_2$ . Plotkin investigated the existence and properties of least general generalizations or *lgg* between clauses and the *lgg* of clauses relative to some background knowledge or *rlgg*. That is, generalizations which are less general, in terms of  $\Theta$ -subsumption, than any other generalization.

More recently, Buntine (Buntine, 1988) defined a model-theoretic characterization of  $\Theta$ -subsumption, called *generalized subsumption* for Horn clauses (see (Buntine, 1988) for more details). Buntine also suggested a method for constructing *rlggs* using Plotkin's *lgg* algorithm between clauses. The general idea of the *rlgg* algorithm is to augment the body of the example clauses with facts derived from the background knowledge definitions and the current body of the example clauses, and then generalized these "saturated" clauses using the *lgg* algorithm. Pal's learning algorithm is based on this framework which is more formally described in Table 1.

A direct implementation of it is impractical for all but the simplest cases, as it essentially involves the deduction of all ground atoms logically implied by the theory (see (Niblett, 1988) for a more thorough discussion on generalization). However, *rlgg* exists for theories without variables (as in Golem (Muggleton & Feng, 1990)), theories without function symbols (as in Clint (deRaedt & Bruynooghe, 1988)), and when only a finite number of facts are deducible from the theory, either by limiting the depth of the resolution steps taken to derive facts and/or by constraining the theory, as in Pal. Even with a finite set of facts, the *lgg* of two clauses can generate a very large number of literals and some additional constraints are required

Table 1: A plausible *rlgg* algorithm for a set of example clauses

- **given:**
  - a logic program ( $\mathcal{K}$ )
  - a set of example clauses ( $SC$ )
- Take an example clause ( $C_1$ ) from  $SC$ . Let  $\theta_{1,1}$  be a substitution grounding the variables in the head of  $C_1$  to new constants and  $\theta_{1,2}$  grounding the remaining variables to new constants
- Construct a new clause ( $NC$ ) defined as:  
 $NC \equiv C_1\theta_{1,1} \cup \{\neg A_{1,1}, \neg A_{1,2}, \dots\}$  where  
 $\mathcal{K} \wedge C_{1body}\theta_{1,1}\theta_{1,2} \models A_{1,i}$ , and  $A_{1,i}$  is a ground atom
- Set  $SC = SC - \{C_1\}$
- **while**  $SC \neq \{\emptyset\}$ 
  - Take a new example clause ( $C_j$ ) from  $SC$ . Let  $\theta_{j,1}$  be a substitution grounding the variables in the head of  $C_j$  to new constants, and  $\theta_{j,2}$  grounding the remaining variables to new constants
  - Construct a new clause ( $C'_j$ ) defined as:  
 $C'_j \equiv C_j\theta_{j,1} \cup \{\neg A_{j,1}, \neg A_{j,2}, \dots\}$  where  
 $\mathcal{K} \wedge C_{jbody}\theta_{j,1}\theta_{j,2} \models A_{j,k}$  and  $A_{j,k}$  is a ground atom
  - Set  $NC = lgg(C'_j, NC)$
  - Set  $SC = SC - \{C_j\}$
- **output**  $NC$

to achieve practical results. PAL (i) uses a pattern-based background knowledge representation to derive a finite set of facts and (ii) applies a novel constraint which identifies the role of the components in different example descriptions to reduce the complexity of the *lgg* algorithm (these are explained below).

Examples in Pal are given as sets of ground atoms (e.g., descriptions of musical scores stating the notes involved on each voice). In general, a musical score can be completely described by the tone and height of each note involved, its time interval and the voice where it belongs. For counterpoint rules of the first specie, time intervals can be ignored (we are beginning to investigate how to include time intervals in the descriptions of scores) and the examples were described by two-place atoms (*note/2*) stating the tone and height of each note and its voice. For instance, *note(c/4, voice1)* states that a "c" note in the center of the piano scale (4) belongs to voice one. Other notes of the same or different voices can be described in the same way. Other examples descriptions have been used in chess (Morales, 1991; Morales, 1992a) and qualitative modelling (Morales, 1992b). Each example description is added to the background knowledge from which a finite set of facts are derived.

In the musical context Pal induces pattern definitions with the following format:

$$Head \leftarrow D_1, D_2, \dots, D_i, F_1, F_2, \dots$$

where,

- *Head* is the head of the musical rule (pattern definition). Instantiations of the head are regarded as musical patterns recognized by the system.
- The  $D_i$ s are "input" predicates used to describe scores (e.g., *note/2*) and represent the components which are involved in the pattern.

- The  $F_i$ s are instances of definitions which are either provided as background knowledge or learned by Pal, and represent the conditions (e.g., relations between notes and voices) to be satisfied by the pattern.

Pal starts with some pattern definitions as background knowledge and use them to learn new patterns. For instance, the definition of *inter\_class2/3* was given to Pal as follows:

```
inter_class2(Note1, Note2, Type) ←
  note(Note1, Voice1),
  note(Note2, Voice2),
  interval(Inter, Note1, Note2),
  int.class(Valid, Inter, Type).
```

where *interval/3* is a background knowledge definition that returns the musical interval between two notes, while *int.class/3* returns if an interval is valid/invalid with its type for valid intervals (i.e., perfect consonance, imperfect consonance, or dissonance). This definition gets instantiated only with example descriptions with two notes of different voices and returns if they form a perfect/imperfect consonance or a dissonance.

Given an example description, Pal "collects" instantiations of its pattern-based background knowledge definitions to construct an initial hypothesis clause. The head of the clause is initially constructed with the arguments used to describe the first example description. The initial head, in conjunction with the facts derived from the pattern definitions and the example description, constitutes an initial concept clause. This clause is generalized by taking the *lgg* of it and clauses constructed from other example descriptions.

Even with a finite theory for music, the large number of plausible facts derivable from it, makes the finiteness irrelevant in practice (e.g., consider all the possible intervals between notes in music). In Pal a fact  $F$  is *relevant* to example description  $D$  if at least one of the ground atoms of  $D$  occurs in the derivation of  $F$ . Since PAL constructs its clauses using pattern-based definitions, only a finite set of relevant facts are considered.

The size of the generalized clauses is limited by requiring all the variable arguments to appear at least twice in the clause. In addition, PAL uses a novel constraint based on labelling the different components which are used to describe examples to guide and constrained the *lgg* algorithm. For instance, notes in the following example are assigned unique labels as follows:

```
note(c/4, voice1) → note(cα/4β, voice1)
note(c/5, voice2) → note(cγ/5δ, voice2)
```

The labels are kept during the derivation process, so the system can distinguish which component(s) is(are) "responsible" for which facts derived from the background knowledge, by following the labels. For example, instances of *inter\_class2/3* will use the same labels:

```
inter_class2(c/4, c/5, perf.cons) → inter_class2(cα/4β, cγ/5δ, perf.cons)
```

The *lgg* between compatible literals is guided by the associated labels to produce a smaller number of literals, as *lgs* are produced only between compatible literals with common labels (a simple matching procedure is used for this purpose). In music this constraint identifies corresponding notes (i.e., notes which are involved in the same relation in different examples). The labels used in one example for the first note are associated with the first note of another example. Thus Pal requires that the corresponding components are presented in the same order.

## 5 Experiments and results

The following musical knowledge was provided to Pal:

- *inter\_class1(Notel, Note2, Valid)*: describes if two notes from the same voice have a valid/invalid interval. Where valid intervals can be consonances or dissonances which follow the same modality of the *cantus firmus* (i.e., a 7th. or augmented 4th. would be invalid)



- *inter\_class2(Nota1, Nota2, Conso)*: describes if two notes of different voices form a perfect or imperfect consonance or a dissonance

Pal was given manually the examples for each rule. It should be noted that the second author suggested all the examples and musical knowledge without knowing the exact functioning of the system. The number of examples required to learn each rule is given below:

Rules	Rule1	Rule2	Rule3	Rule4
Number of examples	6	4	5	5

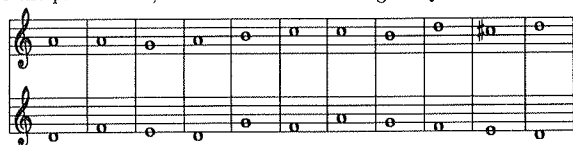
The first rule induced by Pal is shown below (the other three rules are very similar changing only in the different combinations of *perf\_cons* and *imperf\_cons*).

```
rule(Note1, Note2, voice1, Note3, Note4, voice2) ←
  note(Note1, voice1),
  note(Note2, voice1),
  note(Note3, voice2),
  note(Note4, voice2),
  inter_class1(Note1, Note2, valid),
  inter_class1(Note3, Note4, valid),
  inter_class2(Note1, Note2, perf_cons),
  inter_class2(Note3, Note4, perf_cons).
```

The rules learned by Pal were tested for analysis on simple counterpoint pieces. We add an extra argument to each rule to distinguish them from the rest. The analysis was made with the following program:

```
analysis([Note1, Note2|RVoice1], [Note3, Note4|RVoice2],
  [NumRule|Rules]) ←
  rule(Note1, Note2, Note3, Note4, NumRule),
  analysis([Note2|RVoice1], [Note4|RVoice2], Rules).
analysis([ - ], [ - ], [ ]).
```

For instance, for the piece below, we obtained the following analysis:



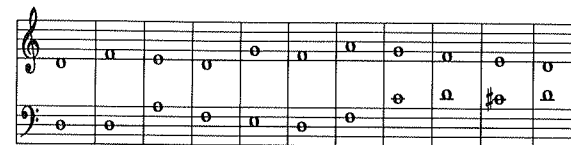
```
?- analysis([d4,f4,e4,d4,g4,f4,a4,g4,f4,e4,d4],
  [a4,a4,g4,a4,b4,c5,c5,b4,d5,cs5,d5],
  Rules).
```

Rules = [r1,r4,r3,r2,r3,r2,r4,r4,r4,r2].

The same program can be used for musical generation. For example, given the *cantus firmus* below, we can generate the required counterpoint notes:

```
?- analysis(Notes,[d4,f4,e4,d4,g4,a4,g4,f4,e4,d4],[r1,r3,r3,...]).
```

Notes = [d3,d3,a3,f3,e3,d3,f3,c4,d4,cs4,d4].



## 6 Conclusions and future work

In (Widmer, 1992), Widmer describes a system capable of learning counterpoint rules using an Explanation-based learning approach (de Jong, & Mooney, 1986). Unlike Pal, a generalization of the target counterpoint rules is required as background knowledge, from which the more specific counterpoint rules are derived. By contrast, Pal uses a much simpler background knowledge to induce equivalent rules.

In this paper, it is shown how Pal, an ILP system, can effectively learn simple counterpoint rules from general purpose musical knowledge and simple example descriptions. The learned rules can be used for musical analysis and generation. This is an initial step towards learning more complicated musical rules expressing personal criteria follow by composers. If we succeed in our goal, the learned rules would provide explicit knowledge of preference criteria follow by composers. This knowledge could be used for analysis and provide suggestions for musical compositions.

## References

- W. Buntine (1988). Generalised subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2):149-176.
- De Raedt, L. & Bruynooghe M. (1988). On Interactive Concept-Learning and Assimilation. In D. Sleeman & J. Richmond (Eds.), *Proc. of the third european working session on learning*, EWSL-88, pp. 167-176, London, Pittman.
- Fux, J.J. (1971). *Gradus ad Parnassum*, 1725. Translated and edited by Alfred Mann, W.W. Norton & Company.
- Morales, E. (1991). Learning Features by Experimentation in Chess. In Y. Kodratoff (Eds.) *Proceedings of the European Working Session on Learning*, (EWSL-91), pp. 494-511, Berlin, Springer-Verlag.
- Morales E. (1992a). Learning Chess Patterns. In S. Muggleton (Eds.), *Inductive Logic Programming*, pp. 517-537, London, Academic Press, The Apic Series.
- Morales, E. (1992b). First-Order Induction of Patterns in Chess, *Ph.D. Thesis*, The Turing Institute - University of Strathclyde, Glasgow.
- Muggleton S. & Feng, C. (1990). Efficient induction of logic programs. In S. Arikawa, S. Soto, S. Oh-suya, & T. Yokomari (Eds.), *Proceedings of First International Workshop on Algorithmic Learning Theory*, (ALT90), pp. 368-381, Tokyo, Japan, Ohmsha.
- Muggleton, S. (1991). Inductive Logic Programming, *New Generation Computing* 8: 295-318.
- Niblett, T. (1988). A study of generalisation in logic programs. In D. Sleeman & J. Richmond (Eds.), *Proc. of the Third European Working Session on Learning* (EWSL-88), pp. 131-138, Glasgow, Pittman.
- Plotkin, G.D. (1969) A note on inductive generalisation. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence* 5, pp. 153-163, Edinburgh, Edinburgh University Press.
- Plotkin, G.D. (1971a). A further note on inductive generalisation. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence* 6, pages 101-124, Edinburgh, Edinburgh University Press.
- Plotkin, G.D. (1971b). Automatic Methods of Inductive Inference. *Ph.D. Thesis*, Edinburgh, Edinburgh University.
- Widmer, G. (1992). The importance of basic musical knowledge for effective learning, In M. Balaban, J. Ebcioğlu & O. Laske (Eds.), *Understanding Musica with AI: Perspectives on Music Cognition*, Cambridge - Menlo Park: AAAI Press/MIT Press.

## A System for Aiding Discovery in Musical Analysis

EDILSON FERNEDA \*, CARLOS ALAN PERES DA SILVA \*\*,  
LUCIÊNIO DE MACÊDO TEIXEIRA \*\*, HÉLIO DE MENEZES SILVA \*

*Universidade Federal da Paraíba*

*\* Departamento de Sistemas e Computação / \*\* Departamento de Artes*

*Av. Aprígio Veloso, s/n — 58.109-970 Campina Grande - PB - Brazil*

*e\_mail: edilson@dsc.ufpb.br / peres@brufpb2.bitnet*

### Abstract

We start by proposing a computer aided scientific discovery system. This system may be seen as a knowledge acquisition environment. We present a knowledge representation (the Semi-Empirical Theories) usable to formulate, to experiment, and to divulge that knowledge, and a protocol (MOSCA) for cooperation between *rational* agents. The protocol is geared to the acquisition and evolution of knowledge. The objective of this system, rather than producing an exact knowledge, is yielding a knowledge which may present a high level argumentation on its validity and may also be improved via a dialog protocol. As an application, we aim at making the machine to behave rationally when performing Musical Analysis, which involves the four technical fields of Music: *composition, execution, theory, and sound digital processing.*

### 1. Introduction

In this paper we propose the specification of a Rational-Agent machine and its application to the field of Musical Analysis. A Rational Agent is an autonomous intelligent system that appears to the user as having reasoning abilities, because it is capable of common sense reasoning (such as those that we exert in our daily lives) and of handling intentions, beliefs, and knowledge that is tolerated to be, to some extent, evolutionary, incomplete, imprecise, and erroneous. This project is characterized mainly by making use of expertise in varied domains of the Cognitive Sciences: Artificial Intelligence, Informatics, Music, Psychology, Didactics, etc.

From an Artificial Intelligence vantage point, this work may be seen as lying in the confluence of the streams of Knowledge Acquisition and Machine Learning. According to the definition proposed in (Aussenac-Gilles, Krivine & Sallantin, 1992):

*The domain of Knowledge Acquisition for Knowledge Based Systems (KBS's) is characterized by the identification and management of the processes necessary to the elaboration (conception, evaluation, and evolution) of a KBS from heterogeneous sources of knowledge (documented, human, and experimental). The result expected from our approach is to furnish the future system with the knowledge that will be the foundation of its expertise. The conductor of the process of knowledge acquisition is the knowledge engineer: he orchestrates the intervention of different processes, actors, and agents.*

While Knowledge Acquisition uses the machine as a mere tool for helping the knowledge engineer to elicit the expert's knowledge, Machine Learning studies the set of mechanisms that gives the machine the faculty of building the knowledge base by analyzing data, explanations, criticisms to solutions, etc. Several

works (e.g. Barboux, 1990) have shown the necessity of making Machine Learning and Knowledge Acquisition to synergetically work together for modeling the control component of the learning process.

Learning proficiently is not enough if the expert is left without proper ammunition to efficiently check and validate the information acquired by the machine. This information has a too large volume and the expert has no tool capable of helping him to efficiently criticize the choices done, particularly the choice of the description language (which implies the choice of the learning tool) and of the selected sample of examples.

A System for Aided Discovery (SAID) is a synergetic combination of Machine Learning and Knowledge Acquisition. It follows principles (summarized in Wielinga, Boose, Gaines Sreiber & van Someren, 1990), such as those for data acquisition, for abstraction based on information about a conceptual model, for particularization of this model, etc. The study of expert systems has shown a pervasive dichotomy between deep knowledge (characterized by having theoretical justification and by being found in scientific books, articles, etc.) and shallow knowledge (which is characterized by being situational, empirical, and not found in conventional scientific writings, though massively used by true experts, being the very cause of their expertise). As it was well put in (Sallantin & Haiech, 1993), a SAID discovers this shallow knowledge by taking advantage of both some deep (theoretical) knowledge made available and a set of incomplete, partially erroneous data. The knowledge base is assumed to be revisable (by error correction) and evolutionary (by making the knowledge more precise, more broad, more deep, more structured, more understandable,... or, in short, by improving the knowledge, in any sense).

In this paper we see scientific discovery as being the result of examining and revising a modeling process over which both theoretic models and experimental data intervene. During the modeling process, discovery is seen as that which was not yet learned by the on going modeling.

A first effort for conceptualizing an artificial apprentice generated a conceptual framework: the Semi-Empirical Theories (SET), introduced in (Sallantin, Szczeciniarz, Barboux, Lagrange & Renaud, 1991; Sallantin, Quinqueton, Barboux & Aubert, 1991). That effort established the elementary concepts which allow building (modeling) an apprentice's knowledge and studying its evolution. SET, however, being focused on the structures and mechanisms of an apprentice, neglected a fundamental learning aspect: the environment for the interaction between the apprentice and the external world. Therefore, a learning environment was proposed, as well as a description of a learning protocol. (Ferneda, 1992) shows how this protocol can be merged with the SET framework.

Since concepts formulated by an apprentice can be erroneous, an intervening agent should be able to determine counter-examples (and also examples likely to be in the frontier or beyond the frontier of the apprentice's current knowledge), testing and exercising him to the limits of his capabilities, hopefully embarrassing him by exposing his deficiencies, therefore stimulating him to revise and improve his knowledge. The goal is not to have an apprentice capable of acquiring a perfect (exact and complete) knowledge, but rather to have an apprentice capable of acquiring a knowledge which will be considered as quite plausible (because the apprentice can yield a high-level argumentation of its plausibility) and may also be corrected/improved via a dialog protocol.

The objective of a SAID applied to Musical Analysis is the analysis of Music's horizontality (melody, theme, scale, ...) and verticality (harmonic structure, instrumental coloring). An immediate application would be a study comparing the works of a same composer, or the works of a set of composers.

## 2. Rational agents

The scientific community, in spite of having tried really hard, has not yet come to a consensual definition of *intelligence*. There are, however, active entities which display behavior conventionally considered as being intelligent. These entities will hereafter be named *agents*.

Researches for the conceptualization and conception of artificial systems (or agents) capable of exhibiting behavior accepted as intelligent must, therefore, take into consideration the several characteristics presumed as necessary for a conduct to be classified as being intelligent. Among these attributes, we are here particularly

interested in the one of rationality (Newell, 1982). The notion of rationality, more specific than the one of intelligence, is related to the treatment of a well delineated class of problems.

A *Rational Agent* is defined as being any (human or artificial) system capable of producing and controlling its own knowledge in a certain domain, in such way that the system will be able to proficiently perform some classes of complex tasks (such as deciding, classifying, diagnosing, predicting, simulating, restricting, conceiving, and planning) conventionally considered as requiring intelligence for being well accomplished.

J. P. Müller (Müller, 1987) showed the possibility of constructing systems that: (i) are able to interpret symbolic structures; (ii) are conscious of their limitations; (iii) act in logical accordance with their beliefs; (iv) are able to adapt their actions to the changes in their knowledge. These systems, therefore, are capable of improving their representation of the external world and of better interacting with this world. This capability of constructing and evolving their representation of the world may be added to the learning aptitude of an intelligent agent.

Next, we will describe the behavior of an apprentice agent (*apprentice*, for short) whose knowledge is the result of communicating with other agents. This agent builds and controls the evolution of its knowledge. It has reasoning mechanisms such as those of the Semi-Empirical Theories (section 3), and its learning environment is based on the MOSCA protocol (section 4).

## 3. Semi-Empirical Theories

Semi-Empirical Theories (SET) are a language-independent knowledge conceptualization introduced by J. Sallantin. SET defines how the knowledge is formulated, experimented, and divulged.

Figure 1 depicts a taxonomy of the terms used for expressing knowledge in SET. This taxonomy is based on the work of T. Addis (Addis, 1988), which revised C. S. Pierce's work (Pierce, 1934) on modeling knowledge. The taxonomy includes: (i) *data* representing the knowledge; (ii) *mechanisms* for creating the data (by *abduction*), organizing them (by *induction*) and propagating restrictions on them (by *deductions*); (iii) *methods* related to the interactions with an external agent that plays the role of criticizing or the role of proposing a statement to be proved. The methods examine the adequacy of information such as *being a lemma*, *being an objection*, *being a proof*, *being a conjecture*, etc.

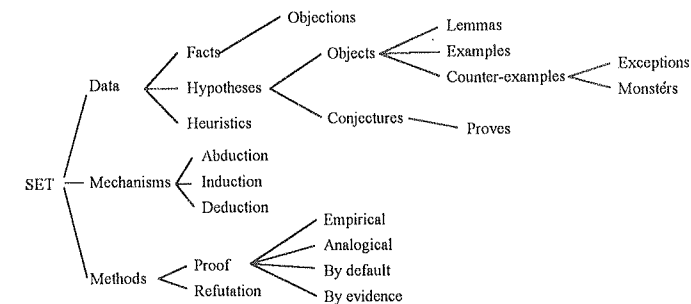


Figure 1: Terms intervening in the knowledge formalization and evolution via SET.

## 4. A Protocol for learning

Formal theories for learning (Boucheron, 1992) define a minimal learning environment which is made of an *apprentice* communicating with an *oracle*. From the point of view of problem solving, the protocol controlling the dialog between these two actors may be summarized as follows: the oracle sends pairs <problem, solution> to the apprentice, each pair being named a *sample*, the problems having been already solved and their solutions known by the oracle; upon receiving each pair, the apprentice stores the information

received and, if it does not perfectly match the *currently learned hypothesis* (the knowledge base), the apprentice searches the *hypothesis space* (the set of all hypothesis which may be formed in the light of all the information, old and new) looking for a hypothesis that, when measured by a *learning criterion*, will be considered better than the previous hypothesis and all the other candidate ones. Therefore, the presented model sees a learning problem as composed of (i) a hypothesis space, (ii) a learning criterion measuring how a hypothesis fits the set of samples, and (iii) a strategy for traversing the hypothesis space.

Two kinds of noise add to the inherent complexity of searching for a hypothesis: (i) the pair <problem, solution> may have been erroneously described, and (ii) the language adopted for describing those pairs may be too coarse, leading to the consequence that it, not perceiving-and-representing the difference between two problem specifications, may present the apprentice with a unique problem having two distinct solutions. Real world applications can not completely escape the existence and negative effects of noise.

This minimal learning environment is implemented as follows: While the machine plays just the role of the apprentice, the expert plays the role of the oracle and may also play some other roles, as we will see. The expert, by choosing the way of structuring and representing knowledge, determines the type of apprentice deemed more adequate to the problem at hand. Well, determining the type of the apprentice is determining the hypothesis space on which the apprentice can operate. This way, the expert is the one who takes all the crucial decisions: (i) He, by choosing the type of the apprentice (and, therefore, the format of the hypothesis, of the samples, and of the problems to be solved), decides the underlying theoretical framework for learning in the application domain; and (ii) The expert selects the examples to be offered to the apprentice. When playing his role as an *oracle*, the expert has available a first manner of pressing the apprentice, imposing a knowledge on him.

After having received a first set of examples and counter-examples and built a learned hypothesis, the apprentice leaves his learning mode and enters his probation mode, ready for solving problems proposed by the expert. Three difficulties appear: (i) examining merely the solutions produced by the apprentice is not enough for evaluating the learned hypothesis; (ii) we are trying to achieve scientific discovery, therefore the expert does not exactly know how to characterize whether or not a hypothesis is a good one, deserving to be maintained; and (iii) hypotheses is usually too large, too unstructured and too complex to be directly utilized by the expert. For these reasons, we provided the apprentice with a high-level argumentation mechanism whose importance has acknowledged by some researchers (Fisher, Lemke, Mastaglio & Morch, 1991). The expert will judge the goodness of a hypothesis by judging the argumentations presented by the apprentice as an justification of the solutions he found for the posed problems. When playing his role as an examiner, the expert has available a second manner of pressing the apprentice, making him to revise the learned hypothesis.

This informal presentation of the MOSCA<sup>1</sup> (Reitz, 1992) depicts five distinct roles: (i) the *apprentice*, yielding a learned hypothesis which fits well the sample of examples and counter-examples previously made available to him; (ii) the *oracle*, yielding unrefutable <problem,solution> pairs; (iii) the *Client*, which submits problems to the apprentice and expects to receive solutions from him; (iv) the *probe*, yielding refutable <problem,solution> pairs, making him to present the due argumentations; and (v) the *master*, who analyses the apprentice's argumentations and then offers useful criticisms to him. The learning environment is summarized in Figure 2. Additional explanations follow:

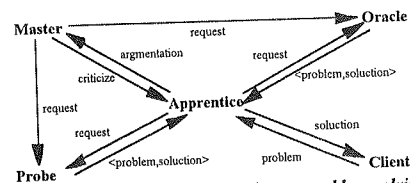


Figure 2: The MOSCA learning protocol, seen from a problem solving vantage point.

<sup>1</sup> MOSCA = Master + Oracle + Sonda (Probe, in Portuguese) + Client + Apprentice.

The apprentice asks a set of <problem,solution> pairs from the oracle. The pairs are then stored (it should be defined whether or not data may be eventually erased/changed) by the apprentice and compose the *sample* to be used for learning. Every change in the sample makes the apprentice to revise the hypothesis so far learned. This hypothesis is extracted from a hypothesis space and satisfies a learning criterion. Of course, any learned hypothesis finds the right solution for all the problems in the sample, and finds none of the wrong solutions for the counter-examples in the sample.

The apprentice may request a <problem,solution> pair, or even a set of those pairs, from the oracle. There two ways for making this request: (i) When the choice of the problem is left to the oracle (who may do it following or not may plan, such a previously defined teaching plan), the request is made just by sending a signal; (ii) when the apprentice desires to learn to solve a certain problem, he makes his request by precisely specifying this problem (of course this must be allowed only in a controlled way, or else the apprentice would take the time just unproductively interrogating the oracle).

Similarly, the apprentice receives <problem,solution> pairs from the probe. These pairs may be intentionally erroneous. The apprentice compares his solution against the one proposed and displays to the master an argumentation justifying the solution. There are two forms of argumentations: explanations (when the solutions agree one with the other) and objection (otherwise).

For each argumentation presented, the apprentice receives a criticism from the master. Whenever possible, every negative criticism makes the apprentice to present an alternative argumentation. When no alternative argumentation is possible, then the apprentice either weakens the learned hypothesis, in such way that it no more produces the pair <problem, solution>, or he consults the oracle, aiming at revising the learned hypothesis and, consequently, revising the argumentation.

The master, by either sending a signal or a problem specification, controls the probe's production of pairs <problem, solution>.

Whenever the apprentice weakens his learned hypothesis, thus becoming silent about certain problems, the master finds a way of forcing the apprentice to re-strengthen his learned hypothesis: the master sends (again either via a signal or via a problem specification) to the oracle a request for generating an adequate set of pairs.

A *Client* submits a problem to the apprentice and expects a solution from him. Whether or not an argumentation on the solution is sent to the master, depends on how the apprentice was defined.

## 5. Discovery in musical analysis

The musical thinking, when generating a certain work, spans aspects pertaining to a knowledge branch difficult to approach. This has lead us to conceptualizations emphasizing terms, such as "gift" and "inspiration", which were quite used, in the past, to impose an end to any discussions on music creation. Our days, however, such extremist position fully satisfies neither the artist, nor the scientist, as both of them, knowing how sounds effect the men, have been studying the organization of sounds, encompassing aspects spanning from its atomic form, pitch up to its final manifestation, the composition.

Perception of this organization in the composition, and perception of the observance of clearly defined forms and rules regulating the creative process, indicate the existence of a systematization in the musical thinking. This systematization is common to all composers in a given context (style-and-school, local-and-time, etc.) and may be represented by means of a knowledge base constructed by a musician and which can constantly have its rules corrected, added, modified, substituted, or eliminated (in short, enriched in any form) at the discretion of the musician.

As it was shown by R. B. Dannenberg (1993), the musical thinking does not follow a linear pattern as it does in exact sciences. The reason for this is that many complex factors interact in the musical thinking: creativity, emotions, intuition, and the proper vibratory nature of sound. However, during the creative process the musical thinking maintains its relationship with form, structure, and harmony, determining a musical logic which allows classifying a work as belonging to a style. Those characteristics strongly associate themselves with those peculiar to given composers.

According to H. J. Koellreutter (Zagonel & la Chiamulera, 1985), the composition process follows four steps: consciousness of the idea, formal conception, choice of the musical signs, and structuring. From the above, (Kugel, 1990; Roads, 1985; Widmer, 1992) perceived that the reasoning of a composer may be simulated by a knowledge based system zeroing not in the musical work proper, but in the process which he used for generating his work. This system, besides the computational aspects involved in analyzing a musical work in terms of information processing, should handle knowledge on domains influencing musical conception: (i) *Aesthetics*, involving concepts from Physics (study of the vibratory nature of the sound), Psychology (study of sound as a psychic phenomenon), and Sociology (study of the ideological aspects of the creator), and Statistics; (ii) *Music*, with its rules for harmony, melody, etc.; and (iii) *History of Arts*, approaching the peculiarities of each style in a given period of History. All these knowledge areas interact for analyzing a composer's work and also for further understanding men's reasoning process.

The study of musical analysis has several facets and is intimately bound to Aesthetics (the study of the conditions and the effects of the artistic activity), which can not be dissociated from History of Arts. We can say that the analysis of a musical work is really much more than a study for deciphering a language or a simple reading of the formal aspects of the work. We adopted the view that knowledge on Aesthetics and on History of Arts should come to the aid of knowledge on Musical Theory. This way, we define a musical work as being the result of knowledge on the domains of Aesthetics, of Music, and of History of Arts.

We aim at making the machine to behave rationally when performing Musical Analysis, which involves the four technical fields of Music: *composition, execution, theory, and sound digital processing*. We investigate, therefore, the conception and development of a system capable of aiding musical analysis.

## 6. Conclusions

We presented a learning environment whose protocol, identifying the set of communications needed for controlling an agent, is an extension of the classic protocol and permits analyzing the process of revising the knowledge acquired by the apprentice. This environment was studied within an conceptual framework, the Semi-Empirical Theories, supporting the expression of both the reasoning and the structure of the apprentice. Under the light of both theoretical and practical Machine Learning current results, integrating high level argumentation techniques into the learning system was deemed necessary to get the expert's validation, approval, and confidence in the acquired knowledge.

Our experience has shown that most of the currently available learning tools do not fully satisfy the expert's expectations and needs. More than interested in a system that merely has the capability of learning correctly, experts are looking for systems able to go beyond that by understandably and convincingly explaining what and how they have learned. One of the major reasons for this is that the explanations may hint what should be changed in order to improve the knowledge base. All of this grows in importance when the expert uses the machine as an aid for scientific discovery (of course this involves modeling a phenomenon).

If the user wants to teach the system, than he, while playing the role of the oracle, he should select the relevant problems. While playing the role of the master, he should use his deep knowledge to criticize the currently learned knowledge in order to identify lemmas already validated the proofs. While playing the role of the probe, he should produce relevant examples and counter-examples. Finally, while playing the role of the apprentice, he should choose methods and heuristics to be used for advancing a new learning step in the light of the last information stimulus received from any of the other agents.

Our approach was corroborated just by few and microscopic experiments (Ferneda, 1992), thus it still calls for more numerous and larger scale experiments in order to fully establish itself as a really useful learning framework. In spite of this, the way our proposed system solves learning problems may be seen as an advancement at least as a methodologically and didactically relevant concept, in as much as it sees the learning problem in a way understandable and profitable to both the application domain expert and the Artificial Intelligence researcher. It should be noticed that we assured the possibility of refuting the learned knowledge. For accomplishing that, it is necessary that all heuristics involved in the knowledge acquisition may be

reevaluated. This justifies our approach in adopting the Semi-Empirical Theories and the MOSCA protocol for representing and evolving the knowledge, respectively.

Musical Analysis seeks to explain a musical work by making use of bodies of knowledge such as Aesthetics, Music, and History of Arts. It is our firm and well-founded belief that the environment here presented will show itself to be a satisfactory aid for the task of performing Musical Analysis, since the environment makes room for those bodies of knowledge (what is fundamental for the conception of a musical work) and allows the construction of a theory that permits the characterization of the work through dialogues with a human, application-domain expert agent.

## References

- T. R. Addis (1988). Knowledge organization for abduction, *Interdisciplinary Information Technology Conference*, Bradford University (England).
- N. Aussenac-Gilles, J.-P. Krivine, J. Sallantin (1992). L'acquisition des connaissances pour les systèmes à base de connaissance, *Revue d'Intelligence Artificielle*, Vol. 6, n° 1-2, Hermès, Paris.
- C. Barboux (1990). "Contrôle par objection d'une théorie incomplète", *Doctorate Thesis*, Université de Montpellier (France).
- S. Boucheron (1992). *Théorie de l'apprentissage: de l'approche formelle aux enjeux cognitifs*, Hermès, Paris.
- R. B. Dannenberg (1993). Music representation issues, techniques, and systems, *Computer Music Journal*, Vol. 17, n° 3, pp. 20-30, MIT Press.
- E. Ferneda (1992). *Conception d'un Agent Rationnel et examen de son raisonnement en géométrie*, Doctorate Thesis, Université de Montpellier (France).
- G. Fisher, A. C. Lemke, T. Mastaglio, A. I. Morch (1991) The role of critiquing in cooperative problem solving, *ACM Transaction on Information System*, Vol. 9, n° 3, pp. 123-151.
- P. Kugel (1990). Myhill's Thesis: There's more than computing in musical thinking, *Computer Music Journal*, Vol. 14, n° 3, pp. 12-25, MIT Press.
- J.-P. Müller (1987). *Contribution à l'étude d'un agent rationnel : spécification en logique intensionnelle et implantation*, Doctorate Thesis, Institut National Polytechnique de Grenoble (France).
- A. Newell (1982). The knowledge level, *Artificial Intelligence*, n° 18, pp. 87-127.
- C. S. Pierce (1934) Scientific method, in *Collected Papers of Charles Saunders Pierce*, P. Weiss (Ed), Harvard University Press.
- Ph. Reitz (1992). *Contribution à l'étude des environnements d'apprentissage. Conceptualisation, Spécifications et Prototypage*, Doctorate Thesis, Université de Montpellier (France).
- C. Roads (1985). "Research in Music and Artificial Intelligence", *ACM Computing Surveys*, Vol. 17, n. 2, pp. 163-190.
- J. Sallantin, J.-J. Szczeciniarz, C. Barboux, M.-S. Lagrange, M. Renaud (1991). Théories semi-empiriques: conceptualisation et illustrations, *Revue d'Intelligence Artificielle*, Vol. 5, n° 1, pp. 9-67, Hermès, Paris.
- J. Sallantin, J. Quinqueton, C. Barboux, J.-P. Aubert (1991). Théories semiempiriques: éléments de formalisation, *Revue d'Intelligence Artificielle*, Vol. 5, n° 1, pp. 69-92, Hermès, Paris.
- J. Sallantin, J. Haiech (1993). L'aide à la découverte scientifique: évaluation sur l'investigation des séquences génériques, *Revue d'Intelligence Artificielle*, Hermès, Paris.
- G. Widmer (1992). Qualitative perception modeling and intelligent musical learning, *Computer Music Journal*, Vol. 16, n° 2, pp. 51-68, MIT Press.
- B. Wielinga, J. Boose, B. Gaines, G. Scriber, M. van Someren (Eds) (1990). Current trends on knowledge acquisition, *Proceedings of the European Knowledge Acquisition Workshop*.
- B. Zagonel, S. M. la Chiamulera (orgs) (1985). *Introdução à Estética e à Composição Musical Contemporânea - H. J. Koellreutter*, Editora Movimento, Porto Alegre (Brazil).

### An integral educational project: Musical Production

Ricardo Dal Farra

*Escuela Técnica ORT  
Producción Musical*

*Yatay 240  
1184 Buenos Aires  
Argentina*

*e-mail: dalfarra@clacso.edu.ar  
or rqdalfar@arriba.edu.ar*

#### Abstract

The ORT Technical School has developed and started recently an educational project that integrates scientific, artistic and technological aspects related to musical production and creation. The formation of the future "Technicians on Musical Production" aims to cover a wide space existing amongst those who develop their work in the creative field and those who work in a specifically technical area.

#### Introduction

The ORT Technical School has developed and started recently a project that integrates scientific, artistic and technological aspects related to musical production and creation. After completing the first three years of secondary school, students can choose among several specialities to complete their studies, such as: Electronics, Industrial Design, Mass Media, Computers, Construction, Chemistry and lately, Musical Production. The speciality Musical Production extends for three years (a total of 6 years of secondary school) and students are graduated as "Technicians on Musical Production".

#### Musical Production

The syllabus of this speciality is formed of different subjects aimed to an integral education: Anatomy and Physiology, Oral and Written Expression, State and Society in Argentina, Gymnastics, Swimming, English, Economics, altogether with other subjects whose contents furthers the integration of scientific knowledge with the development of the creativity and the capacity to use efficiently the available technology: Musical Technology Workshop, Electronics, Physics, Electroacoustics Workshop, Mathematics, History of Musical Culture, Ear Training, Musical Structures, Keyboard Workshop, Musical Production Seminars, Multimedia Laboratories, Applied Computers and Final Project.

The permanent coordination among the areas is the fundamental aspect that make this plan successful. Therefore, the contents of Mathematics are related to the proper problems of the sound and music areas. The same is applied to Physics, both in the generation, transmission and reception of sounds, as on recording and playback. Even in English as well as Oral and Written Expression, students undertake topics of their speciality without forgetting their general education.

Being that the integration among different subjects is the key of the speciality, the idea is more evident in the subjects directly connected with music and sound. Subjects such as Musical Structures, Ear Training, History of the Musical Culture and Keyboard Workshop are functionally coordinated and connected among themselves; and the different topics studied in the Musical Technology Workshop (main core of the speciality in the first year) is thoroughly related to Ear Training, Electronics, Musical Structures, Physics, Keyboard Workshop, English and Mathematics.

As I pointed out before, the Musical Technology Workshop is not - during the first year - one more subject. It is the "heart" of this speciality, an experimentation, investigation and creation center. Students work in pairs, sharing workstations formed of hardware and software that let them develop (during a prefixed time) different

contents such as: additive synthesis, sampling, subtractive synthesis, digital recording, musical notation, sound processing, analog recording, FM synthesis, and construction of musical structures, amongst others. The different workstations are adapted to the necessities of each one of the subjects, and students "rotate" from one to another, completing the experiences proposed. The equipment of each workstation is of the latest generation and includes a computer (Macintosh or PC) provided with the software to elaborate, process, control, record and reproduce sound (Digidesign's Sound Designer II, and Turbosynth; Opcode's Studio Vision, Galaxy Plus and Max; Passport's Master Tracks, Encore and Alchemy; Farallon Computing's SoundEdit; Twelve Tones' Cakewalk Pro for Windows; Mark of the Unicorn's Digital Performer, Mosaic and Unisyn; Coda's Finale; OSC's Deck; ...). At the same time, there are synthesizers and samplers (Korg's Wavestation A/D, 01/W and M1; E-mu's Emax, Emax II, and MPS Plus; Yamaha's DX7II, TX81Z and SY35; Kawai's K5; Oberheim's OB-1; Kurzweil's K2000; ...), analog and digital recording systems (Digidesign's Audiomedia II; Revox's A77; analog multitrack cassette decks; ...), sound processors (Roland's RSP-550; Boss' SE-50; graphic equalizers; ...), one drum machine (Roland's R8 MKII), an spectrum analyzer, and a keyboard controller (Roland's A-80; ...), among other elements.

The infrastructure of the Musical Technology Workshop is very important to form creative professionals who should know how to take profit out of the latest technology when available, and - at the same time - to be able to accomplish a good task even when the work conditions are not the best (an essential aspect to face, taking into account the present Latin American reality).

In 1993 the first year of this new course was completed. Now a small recording studio is being finished to be used by the students on their second and third year, and a multimedia lab is planned for their last year at the school. The recording studio have both analog and digital recording systems (Digidesign's Sound Tools II; 8 tracks digital tape recorder; DAT recorder; 8 tracks analog tape recorder; 2 tracks analog tape recorder; ...), a 32 channel mixing board, professional monitor speakers (Tannoy and JBL), different kind of microphones (Shure's SM 81, SM98A, SM57 and SM58, Crown's PZM 30F, Sennheiser's MD421 and Beyer's M500TG; ...), several sound processors (Lexicon, Yamaha, Ensoniq, Roland, Behringer, ...), plus a Macintosh Quadra 840AV and a PC486 with softwares for different applications (Digidesign's, Digital Intelligent Noise Reduction; Opcode's Cue; Passport's Producer; Soundtrek's The Jammer; Emagic's Logic Audio; Cool Shoes' Sound Globes; Jupiter Systems' Multiband Dynamic Processor and Infinity; Steinberg's Cubase Audio;...), a sampler and synthesizer keyboard (Kurzweil's K2000S), a drum controller, and some acoustic instruments. At the multimedia lab, students will work with multimedia and hypermedia, and also will experiment with the creation and editing of sound effects and music for video and films.

During the last year at the school, the students will receive a balanced program combining art and science: theoretical scientific principles of electroacoustics; practice with the technology of recording, electronic musical instruments and live sound amplification; computer programming; study and creation of musical structures both from the contemporary academic and experimental, as well as traditional folk (ethnic / world music) and non-academic (pop, rock, jazz, ...) point of view will be worked. An annual final (and advanced) project will be developed by the students putting in practice the knowledge acquired.

### Conclusions

The formation of the future "Technicians on Musical Production" aims to cover a wide space existing between those who develop their work in the creative field and those who work in a specifically technical area. This educational project aims to form individuals in the scientific, artistic and technological realms with a careful balance, so that the Technicians on Musical Production will be ready to work in different technological-musical fields as soon as they graduate or, on the other hand, to continue their mastering at the university. In this last case, and taking into account that superior studies are usually directed towards scientific and/or technological aspects or, on the contrary, to purely artistic ones (loosing multiplicity of perspectives and interfering on the coordination of efforts between professionals highly specialized that works on multidisciplinary teams), the Technicians on Musical Production will find a firm background in his secondary school formation which would complement the aspects forgotten in the university.

### Aknowledgements

This educational project is being realized thanks to the efforts of the directive staff of ORT Argentina and the ORT Technical School, the director of the speciality, and all the teachers involved on the project, convinced of working in a different proposal that supports a plan of real change for our environment.

## Inteligência Artificial, Psicoacústica e Modelos Cognitivos

## A Connectionist Model for Chord Classification

FABIO GHIGNATTI BECKENKAMP\*  
PAULO MARTINS ENGEL\*\*

CPGCC, INSTITUTO DE INFORMATICA, UFRGS  
Caixa Postal 15064  
91501-970 Porto Alegre - RS- Brazil

### ABSTRACT

This work is a contribution to the unification of the worlds of computer science and music. This is a first step in the direction of opportunities offered by connectionism on music research. This paper studies the application of a connectionist model on an actual problem of music domain and tries to demonstrate the efficiency of using connectionism in the chord classification problem. This work specifies a neural network model based on Backpropagation architecture to recognize four types of chords: major, minor, diminished and augmented. Simulations results have shown that the network recognized the four chord types for twelve major and twelve minor tonalities.

### 1 Introduction

Music is a human expression that is difficult to study and to model because it evolves great creativity and esthetic concepts. The problems encountered on such study have challenged researchers interested in different aspects, reaching from those that investigate music applying traditional methods to others that apply artificial intelligence (Todd, 1991).

The connectionist paradigm offers a new and unified method of music investigation due to its inherent aspects such as learning, generalization and feature abstraction. Neural networks offer powerful tools to be used by researchers to solve actual music problems such as: human voice and instruments sound perception, performance interpretation, composition process understanding, musical education, etc. Neural networks and music research might provide benefits to musicians by providing new tools to help their work on music.

The chord classification is a relatively easy task for a musician but it is not easy to model the musician knowledge used to do this task. Some efforts to model the chord classification task were done. Pitts and McCulloch in 1947 (Pitts & McCulloch, 1947) created a model based on the cortex tonotopic structure, where equal intervals on the frequency axis were assumed to map into equal cortical distances. Berenice Laden (Laden, 1989) used this interval approach to create two artificial neural models for chord classification. The first model was based on cognitive representation of musical pitch and the second on psychoacoustic representation.

### 2 Goals

This work has two main goals:

- The proposal of a chord classification neural model, and
- the experimentation and validation of neural network applied to music cognition domain.

\* Student of Master Degree in Computer Science at Universidade Federal do Rio Grande do Sul - UFRGS, E-mail: fgb@inf.ufrgs.br

\*\* Professor at Universidade Federal do Rio Grande do Sul - UFRGS E-mail: engel@inf.ufrgs.br



Chords are more than two simultaneously played notes. The chords considered on this work are triads, i.e., chords of three notes. The proposed neural network must classify four chord types: major, minor, augmented and diminished. Those types were chosen because they are the basic triads for more sophisticated chords that evolve other intervals as seven, ninth, etc.

The chord classification problem consists of presenting a chord to the input of the network. This input will activate the network producing a network output. The output signals indicate what kind of chord was presented. The network used to model this problem was the Backpropagation network.

### 3 Methodology

Chord classification has been investigated by two points of view: psychoacoustic and cognitive. Psychoacoustic researchers have been interested in low-level pitch representations like some physic parameters, as frequency. Cognitive researchers may be interested in more abstract levels such as rhythm and tonality. The neural network researchers might be interested in both levels (Laden 1989). In this work only the cognitive approach will be considered.

In this work we implemented the Backpropagation algorithm based on Rumelhart (Rumelhart & McClelland, 1986). The Backpropagation network is used to solve problems that require complex pattern recognition and that need to map nontrivial functions. The proposed neural network shown on figure 1 was modeled with three layers: the input -, the hidden -, and the output layer. The input layer represents the twelve notes on chromatic scale. A hidden layer maps the input patterns into interval patterns. The output layer indicates what kind of chord was presented.

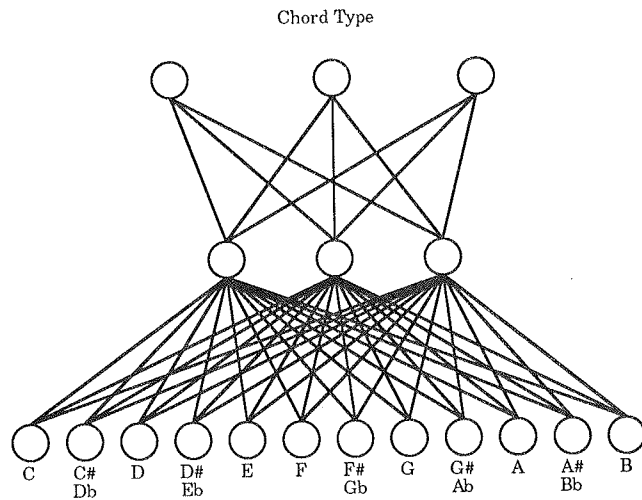


Figure 1 - Neural Network Architecture

The number of neurons on input layer is determined by the pitch representation. The chosen pitch representation is simple: 12 input neurons were stipulated where each neuron is associated to one note of the natural scale. Each input neuron is activated when the respective note is played. The octave of the played note is not considered. All notes are transposed to only one octave. If the same note is played at the same time in different octaves, only one note will be considered. The activation values of the inputs can be 0 or 1, where 0 means a not played note and 1 a played one.

The ideal number of hidden neurons will be determined by inspection during the neural net training. The higher the number of hidden neurons has the neural network, the better will be the convergence possibility. In

this case, a large computational effort will be necessary due to the increase of connections number among network layers.

The output layer has three units that were modeled according to the existent intervals among triad notes. The chord classification can be determined by verification of those intervals. The first output neuron represents the interval between the root and the third of the chord. The second output neuron represents the interval between the third and the fifth of the chord and the third output neuron represents the interval between the root and the fifth of the chord.

In the first and in the second neuron the output must be 1 if the interval is minor and it must be 0 if it is major. In the third neuron, the output must be 0 if the interval is diminished, 0.5 if it is perfect and 1 if it is augmented. The expected output patterns for the four types of chords are presented on table 1. The utilization of the second neuron might be unnecessary because it is not important to know the interval between the third and the fifth of the chord, but it was considered to model all existing chord intervals.

Table 1 - Output Patterns

1 <sup>a</sup> neuron	2 <sup>a</sup> neuron	3 <sup>a</sup> neuron	Chord
0	1	0.5	major
1	0	0.5	minor
1	1	0.0	diminished
0	0	1	augmented

For network training, 48 chords were used; twelve of each type. One example chord is presented to the network at each training cycle. One different chord is selected at each interaction from the chord example set. To assure that the network training will not privilege one chord type, two chords of the same type are never successively presented to the network.

### 4 Result Analysis

In spite of the fact that the network weights were initialized with random values, each simulation of a BPN can have different performances. This initialization affects directly the number of interactions that the network needs to learn all training examples. Due to this fact a great number of simulations were realized with each network configuration and on next tables only best results are shown.

The learning rate parameter ( $\eta$ ), is very important for the training process. With a large  $\eta$  the network training will need a small number of interactions to learn the examples but it can cause the training error to go toward a local minimum. If it happens the network can not find an acceptable solution. On the other hand, a small  $\eta$  ensures that the network will settle on an acceptable solution but it will need more training interactions.

Table 2 shows initial simulation results. These simulations used a small  $\eta = 0.25$  so that exhaustive training was performed to allow to the network to settle on a solution. Another objective of these simulations was to find an appropriate number of hidden units. The network outputs were observed during the learning phase to verify if they had been modified to settle on a solution.

Table 2 - Short  $\eta$  Training

Network	Number of Hidden Units	$\eta$	$\alpha$	Correct Chords	Interactions
1	36	0.25	0.9	48	41520
2	25	0.25	0.9	48	35760
3	9	0.25	0.9	48	56352
4	3	0.25	0.9	0	300000

The simulations of table 2 correctly classified 100 percent of trained chords but the number of interactions needed for training was larger than expected. The network that had the best performance was number 2, that was

trained with 25 hidden units. The network 4, that had only three hidden units, did not recognize any chord after 300000 interactions.

Another way to try to reduce the interaction number is to control the momentum ( $\alpha$ ). Setting it to 0.2 for the first 40 interactions and to 0.9 for the other ones will reduce the network probability of reaching a local minimum (Franzini, 1988). Table 3 shows a comparative performance between network training using one and two  $\alpha$  values. It shows the difference of performance between small and large  $\eta$  too. Network 2 is repeated to an easier comparison.

Table 3 - Two  $\alpha$  Training

Network	Number of Hidden Units	$\eta$	$\alpha$	Correct Chords	Interactions
2	25	0.25	0.9	48	35760
5	25	0.25	0.2/0.9	48	37584
6	25	0.9	0.9	48	18096
7	25	0.9	0.2/0.9	48	36336

The use of two  $\alpha$  during the training did not increase the network performance as expected. The network 6, which was trained with large  $\eta$  and large  $\alpha$ , got the best training result.

The interaction number needed for training is still very large. This fact causes training set adjusts. In the selection of the example set it is important to cover all the problem domain and not to favour one pattern. The number of examples used for training is the same for all patterns but the problem domain is not covered. The input of the network was modeled with 12 binary inputs that means  $2^{12} = 4096$  possible input examples.

Considering only triads, it is a combination of  $C_{12}^3 = 220$  possible input triads.

The domain of the problem (48 chords), is smaller than the input domain so that the network is not mapping all possible input patterns. Besides this, it is easy to verify that the 48 chords cover all possible examples for the 4 types of chords. Simulations were realized with less than 48 examples but the network was not able to generalize in the way to classify not trained patterns.

To solve this input modeling problem the network was trained with the 48 known triads plus some examples from the rest possible triads. Those triads were presented to the network as unknown patterns and were called "random triads" because they were randomly searched from the rest of possible input triads. These random triads does not necessarily have musical meaning. During the training, one random triad is searched and presented to the network repeatedly after a certain number of training cycles ( $n$ ). The trained output pattern for these random triads has zeros for the three output neurons.

Simulations were done to verify if the use of random triads helps to decrease the number of training interactions and to find the best value for  $n$ .

Table 4 - Training with Random Chords

Network	Number of Hidden Units	$\eta$	$\alpha$	$n$	Correct Chords	Interactions
8	25	0.9	0.9	3	48	19536
9	25	0.9	0.9	4	48	21312
10	25	0.9	0.9	8	48	13968
11	25	0.9	0.9	12	48	15360

The results on table 4 show that the use of random examples really decreases the number of network training interactions. The network 10 had the best performance, where 48 chords were correctly recognized after 13968 interactions.

## 5 Conclusions

This paper demonstrated the use of connectionism in the solution of a musical problem and shows the capacity of using such unconventional musical investigation technique. This work presented a cognitive connectionist model that solved the chord classification problem. It also made investigated network learning parameters, training examples and network architectures that provided an increase of the network performance.

The same model has been tested to classify the 4 triads types plus seven chords. The unique model modification is the increase of one output neuron to indicate the presence of a seven interval in the input chord. The new model has a very similar performance to the older one, indicating that other chord classes may be considered in the future. In further studies this model can be used for more complex chord classification, where tonality and chord inversions may be considered.

## Bibliography

- Franzini, M. A. Learning to recognize Spoken Words: a study in connectionist speech recognition. In: *Connectionist Models Summer School*. Proceedings... San Mateo: Morgan Kaufmann, 1988, p. 407-416.
- Laden, Berenice; Keefe, Douglas H. The Representation of Pitch in a Neural Net of Chord Classification. *Computer Music Journal*, v.13, n. 4, Winter 1989.
- Pitts, W.; McCulloch, M.C. How We know Universals: the perception of auditory and visual forms. *Bull. Math. Biophysics*, v.9, p. 127-147, 1947.
- Rumelhart, David; McClelland, James. *Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1986, v.1, p. 318-334.
- Todd, Peter; LOY D. Gareth. *Music and Connectionism*. MIT Press, Cambridge, MA, 1991, p. ix-xi, 3-19, 31-33, 39-41, 64-85.

## The MusES system : an environment for experimenting with knowledge representation techniques in tonal harmony

FRANCOIS PACHET,

LAFORIA, Institut Blaise Pascal, 4, Place Jussieu, 75252 Paris Cedex 05, France

### Abstract

We report here on current works on the MusES environment, designed for experimenting with various object-oriented knowledge representation techniques in the field of tonal harmony. The first layer of MusES is a repository of consensus knowledge about tonal harmony, including an explicit representation of enharmonic spelling, as well as representation of intervals, scales and chords that support standard computations. We give an overview of several systems built on top of MusES: a system for the analysis of jazz chord sequences, a system for the automatic generation of harmonizations, and a system that generates real-time jazz improvisations. We give an overview of MusES and its extensions and discuss several representation issues and their solutions in MusES.

### 1. Introduction: yet an other Smalltalk music representation system

Music analysis has long been a favorite domain for researchers in Artificial Intelligence. Within AI, Object-Oriented Programming (OOP) has traditionally been a favorite paradigm to build complex musical systems, especially to oriented towards *synthesis* (from the *Forme* system (Cointe&Rodet 1991) to the *MODE* system Pope (1991), the *Kyma* system (Scaletti (1987)), and *ImprovisationBuilder* (Walker and al., 1992)). Of course, object-oriented programming has been - and is used - for almost any kind of complex software, but there is hardly no mention in the literature of attempts to use specifically OOP techniques to implement analysis systems in tonal music. Following this tradition, we are interested in developing intelligent systems specialized specifically in tonal harmony, using object-oriented techniques. This paper is a report on our results so far, embodied in the MusES system.

From a totally different point of view, our main object of study is the construction of *large knowledge bases*, using OO techniques, and Smalltalk in particular. In this context, tonal harmony is seen as an ideal field for obvious reasons: it is complex yet understandable, involves complex structures which call for non-trivial representations (e.g. intervals, chords), requires adequate representations of time, involves abstract notions (such as analysis, degrees), and so forth. In this respect, we are interested in the integration of OOP with various inference mechanisms to produce truly reusable knowledge-based components.

We will first briefly describe the foundation of MusES, dealing with pitch classes and basic concepts of tonal harmony, then give an overview of three systems built on top of MusES, and end up with a discussion of the results.

### 2. Representation of pitch classes

#### 2.1. The problem ?

One of the foundations of the MusES system is the representation of pitch classes (hereafter referred to as PC), that respects enharmonic spelling (i.e. the difference between notes that spell differently but sound the same, such as Eb and D#). Enharmonic spelling is as vital to music analysis, as orthography is to grammar and semantics. Although it may seem a remarkably simple problem, it has, to our knowledge, yet never been fully addressed. For instance, Winograd (93) emphasizes the importance of taking enharmonic spelling into account, but proposes an ad hoc representation of chords as Lisp dotted lists. Similarly, Steedman (84) proposes a solution for performing harmonic analysis of chords sequences but, considers all the entities (chords, intervals or notes) as Prolog-like constants and is interested only in higher level properties of sequences deduced from the mere ordering of their elements. More generally, MusES addresses the problem of providing a "good" representation of the algebra of pitch classes, including the notion of "enharmonic spelling", and a representation

of intervals, scales and chords to serve as a foundation for implementing various types of harmonic analysis mechanisms. This calculus must take into account various facts and properties of pitch classes, such as :

- There are conceptually 35 different PCs : 7 naturals, 7 flats, 7 sharps, 7 double sharps and 7 double flats, with only one occurrence of each PC (in our octave-independent context). Practically, this means that, for example, the minor second of B (C) is *physically* the same note as the minor seventh of D, and so on.

- PCs are linked to each other half-tone or tone wise, and form a circular list. But some notes are *pitch-equivalent*, (e.g. A# and Bb, or C##, D and Ebb).

- There is a non trivial algebra of alterations, which includes the following pseudo-equations :

$$\# \circ b = b \circ \# = \text{identity.}$$

$$\text{For any } x \text{ in } (\#, b, \text{natural}), x \circ \text{natural} = \text{natural.}$$

This algebra is non trivial because not everything is allowed, at least in the classical theory, e.g. triple sharps.

- PCs are linked by the notion of *interval*, which, in a way, *preserves* this algebra. For instance, the diminished fifth of C is not the same PC as the augmented fourth of C, but both PC sounds the same.

- Certain intervals are forbidden for certain PCs : for example, the diminished seventh of Cb does not exist (it would be Bbbb!).

- Certain scales do not exist, by virtue of the preceding remarks : G# major is impossible (because it would contain a F## in its signature). The same holds for Db harmonic minor, and so on.

## 2.2. Pitch classes as instances

Although it is possible to write a global algorithm in any procedural language that takes all these cases into account, there is clearly here a better solution, which consists in treating pitch-classes as instances of classes, in the sense of OOP, and alterations as methods for these classes, using polymorphism to represent their algebra, and all the properties mentioned above. This approach not only yields a simple implementation, but also provides us with a clear understanding of the operations on pitch classes.

We define 5 different types of pitch-class *classes* (to avoid long name, we refer to pitch-classes as "Note"): PCNatural, PCSsharp, PCFlat, PCDoubleFlat and PCDoubleSharp. Distinguishing between different classes for pitch classes gives us a precise definition to alterations : the #, b, and natural, are then represented as *polymorphic methods* on these classes. For example, the # operation maps instances of PCNatural to instances of PCSsharp: A# is then seen as the result of operation # to note A.

This operation is polymorphic because there are actually four distinct sharp operations, depending on the class of the argument. In order to represent notes according to these requirements, we define a hierarchy of classes as follows, where each class defines its set of instance variables and operations : PitchClass represents the root of all classes representing pitch-classes. It is an abstract class and has no instance variables. PCNatural represents natural pitch-classes. There are 7 instances of PCNatural, representing the 7 natural notes (A, B, C, D, E, F, G). They are linked to each other according to the order (A, B, C, D, E, F, G), and have two pointers on their corresponding *sharp* and *flat* PC. PCAltered is the root of the classes representing altered (and doubly altered) notes. It defines only one instance variable (*natural*) pointing back to the natural note it comes from (e.g. A#, A##, Ab, and Abb all have A as their *natural*). Finally, there are four subclasses of PCAltered for representing respectively sharp, flat, doubleSharp and doubleFlat notes. These classes implement the methods sharp, flat and double flat so as to respect the natural algebra of alterations. As an example, here is the list of all the implementations of method flat:

!PCNatural methodsFor: 'alterations'!	!PCFlat methodsFor: 'alterations'!
flat	flat
^flat	^flat
!PCSsharp methodsFor: 'alterations'!	!PCDoubleSharp methodsFor: 'alterations'!
flat	flat
^natural	^natural sharp

Note that the flat operation is intentionally not defined for class PCDoubleFlat. The flat message sent to a PCDoubleFlat will raise an error, which is consistent with our philosophy. The same patterns applies for method sharp in PCDoubleSharp, as well as for most operations in pitch classes (computation of intervals, of semitones distances, transpositions, etc.)

## 3. Basic Harmony

Once pitch classes are correctly represented, we add the representation of all the basic concepts of tonal harmony, including octave-dependent notes, intervals, scales (classical and exotic ones), and chords. These classes and methods were designed to support basic computations such as the one taught in first year courses. For reasons of space, we will not discuss their representation here (Cf. Pachet (1994) for details), and simply give a few examples of what the system can do.

### 3.1. Alterations on pitch classes and OctaveDependentNotes

A bunch of methods represent most common computations on pitch classes and octave dependent notes, to compute alterations, and test pitch equality, such as:

PitchClass C	-> C	"PCs are accessed by class messages"
PitchClass C sharp	-> C#	
PitchClass C sharp sharp flat	-> C#	
PitchClass C flat flat flat	-> Error: 'flat' not understood by class DoubleFlatNote	
PitchClass C sharp pitchEquals: Note D flat	-> true	
(PitchClass C sharp octave: 3) sharp	-> C##3	" an OctaveDependentNote"

### 3.2. Intervals

Intervals are represented as first class objects. Methods allow their creation from notes, and their manipulation in any possible way. Here is an excerpt of methods dealing with intervals:

PitchClass C flatFifth	->	Cb
PitchClass C augmentedFourth	->	F#
PitchClass C majorThird majorThird	->	G#
(PitchClass B sharp octave: 3) fifth	->	E4
PitchClass C flat diminishedSeventh	->	error: illegal interval
Interval diminishedFifth bottomIfTopIs: (PitchClass F sharp)	->	C
Interval diminishedFifth bottomIfTopIs: (PitchClass G flat)	->	Db
Interval majorThird reverse	->	minor sixth
Interval perfectFifth + Interval majorSecond	->	majorSixth
PitchClass C intervalWith: PitchClass F sharp	->	augmented fourth

### 3.3. Scales

MusES provides a representation of scales that allows easy computation of derived modes, and derived scale-tone chords. Adding new exotic scales is done by defining new subclasses of class Scale, with corresponding interval lists.

PitchClass A flat majorScale	->	Ab major
PitchClass A flat majorScale notes	->	(Ab Bb C Db Eb F G)
PitchClass C harmonicMinorScale notes	->	(C D Eb F G Ab B)

### 3.4. Chords

Chords are an important - a complex - concept in tonal harmony. MusES provides a complete vocabulary that allows to name and manipulate all possible chords (from 2 to 7 notes). Chords may be created either from their name (a string), or from a list of notes. Here are some examples of chord name computations using both mechanisms:

(Chord new fromString: 'D# maj7') notes	OrderedCollection (D# F## A# C##)
(Chord new fromString: 'C 13 aug9 no7') notes	OrderedCollection (C E G D# F A)
Chord newFromNoteNames: 'C E G'	[C]
Chord newFromNoteNames: 'C F G'	[C sus4]
Chord newFromNoteNames: 'C E F F#'	[C no5 no9 no7 11 aug11]
Chord newFromNoteNames: 'D# F## A C##'	[D# dim5 maj7] "from A. Holdsworth"

Similarly, there are methods to compute the list of all *plausible* chord names for a list of notes :

```
"the root is one of the notes : "
```

```
Chord allChordsFromlistOfNoteNames: 'C E G'
```

```
-> OrderedCollection ([C] [E min no5 no7 no9 no11 dim13] [G sus4 no5 6])
```

```
"the root is any note, possibly not in the list : "
```

```
Chord reallyAllChordsFromlistOfNotesNames: 'C E G'
```

```
-> OrderedCollection ([A noRoot min 7] [B noRoot sus4 no5 no7 dim9 dim13] [C] [D noRoot sus4 no5 7 9] [E min no5 no11 no9 no7 dim13] [F noRoot no3 maj7 9] [G sus4 no5 sixth] [A# noRoot no3 dim5] [C# noRoot min dim5] [D# noRoot no3 no5 dim9] [F# noRoot no3 dim5 7 dim9] [G# noRoot no3 no5 no11 no9 no7 dim13] [Ab noRoot aug5 maj7] [Bb noRoot no3 no5 no7 9 aug11 sixth] [Db noRoot no3 no5 maj7 aug9 aug11] [Eb noRoot no5 sixth] [Gb noRoot no3 no5 no9 no7 aug11])
```

An important notion is the notion of "scale-tone chord", extracted from scales by successive thirds. The following expression yields all 4 voice scale-tone chords generated from the scale (C Hungarian Minor) :

```
(HungarianMinor root: PitchClass C) generateChordPoly: 4 ->
```

```
OrderedCollection ([C min maj7] [D dim5 7] [Eb aug5 maj7] [F dim5 dim7] [G maj7] [Ab maj7] [B min dim7])
```

Scale-tone chords are used primarily to determine all possible analysis of a chord. According to the context (e.g. the neighboring chords in the sequence), the right analysis will be chosen (Cf. the analysis of jazz chord sequences). The following method yields all possible tonalities in which a chord may belong, according to the existing scale classes:

(Chord new fromString: 'C maj') possibleTonalities ->		
OrderedCollection (		
[V of F HungarianMinor)	{VI of E HungarianMinor}	{IV of G MelodicMinor}
[V of F MelodicMinor)	{I of C Major}	{IV of G Major}
[V of F Major)	{V of F HarmonicMinor}	{VI of E HarmonicMinor}

#### 4. Temporal objects

An other important aspect of MusES is the representation of time. Our representation of time is based on a simple inheritance scheme. A root class (`TemporalObject`) defines a `startTime` and a `duration`, expressed in fractions of a beat. Notions having a temporal extent are defined as subclasses of `TemporalObject`: `OctaveDependentNote`, `MusicalSilence` and `OctaveDependentChord` (and not `Chord`). Note that although a lot of musical notions have temporal extent, not all of them do! For instance, we thought it was important to represent pitch-classes independently of any representation of time, because all their properties are indeed time-independent. Hence, the classes representing pitch classes (seen above) are not subclasses of `TemporalObject`. On the contrary, `Octave-Dependent-Note` are represented as having an explicit temporal extension, because no interesting time-independent property was uncovered.

Temporal sequences of notes are called *Melodies*. *Melodies* hold a list of octave-dependent notes. We distinguish between monophonic polyphonic melodies, because the latter are strongly connected with the

representation of chords, and involve very specific representations that are irrelevant for simple monophonic melodies. A trivial connection to MIDI has been realized, using Bill Walker's *Midi Smalltalk* primitives for the Macintosh. Since this is not our priority, only basic (but useful enough) play functions have been implemented, and no sophisticated recording (and hence quantization) is realized in the current version.

#### 4.1. Graphical Editors for melodies

Graphical score editors are not only useful for our purposes. They are, in a way, particular knowledge bases, incorporating lots of knowledge about musical notation. For instance, the problem of knowing which way to draw beams (up or down), how and when to group eighth or sixteenth notes together, how to split notes whose durations exceed certain amounts of time (syncopations), where to position notes and so forth, are problems that do require lot of musical knowledge to be solved. We started to implement a series of graphical editors (Cf. Figure 1) for both monophonic and polyphonic melodies, with standard edition operations (key transpositions, copy/cut/paste, fileIn/out, etc.) For the moment these editors are written in *Smalltalk*. However, we plan to redesign them using more declarative knowledge representation mechanisms (constraints), and integrate them in a tutorial system about musical notation.

### 5. Extensions

MusES is used (and validated) by several knowledge-based system built on top of it. We give an overview of three of them here. More details can be found in the references.

#### 5.1. Project #1: Analysis of Chord Sequences

The first project is the construction of a knowledge-based analyzer for jazz chord sequences. The sequences are standard be-bop tunes as found in the *Real book/Fake book* corpus. The aim of this system is to find underlying tonalities for each chord in a sequence, when possible. Previous approaches to this problem were mostly based on a formal theory of the underlying domain. For example, Steedman (84) uses context-dependent grammar rules to model 12-bar blues, that capture all "legal" distortions from the original 12-bar blues sequence. However its model is not directly implementable, and yields solutions only for well formed chord sequences. On the contrary, our approach is based on a model of the reasoning as it is made by experts, and is divided in two phases: 1) *pattern recognition* in which the expert "sees" particular well-known shapes, whose analysis is trivial, such as Two-Five's, Two-Five-One's, Turnarounds, resolutions, etc. and 2) *gap filling* phase, in which isolated, non analyzed chords are grouped to adjacent analyzed shapes when possible.

The system is an extension of MusES with classes to represent chord sequences and objects used for the analysis (the well-known shapes, the analysis themselves, etc.). The reasoning is represented by rule bases expressed in *NéOpus*, a first-order forward-chaining inference engine integrated with *Smalltalk-80* (Cf. Pachet (1995)). The model of the reasoning is described in depth in Pachet (1994b) and Pachet (1991), and uses a declarative architecture for representing control knowledge (Pachet & Perrot (1994)).

#### 5.2. Project #2: Constraint satisfaction and automatic harmonization

This system is an attempt to capture musical rules as found in treatises of harmony and counterpoint. These rules are most often stated as constraints, such as "the interval between two successive notes in a melody should be consonant". One of the major drawbacks of the previous attempts (Ebcioğlu (1991), Chen (1991)) is the overuse of the constraint satisfaction mechanism, leading to inefficiencies and complex knowledge bases. The aim of the system is to explore the integration of constraint-satisfaction mechanisms (arc-consistency) and intelligent search (branch & bound), with our existing object structures. This work is still in progress and showed already very promising results in terms of efficiency, compared to previous attempts by Chen (1991) and Ballesta (1994).

#### 5.3. Project #3: Simulation of real-time jazz improvisations

This system is an attempt to build a musical memory that explains - at least partially - improvisation processes. A model of memory, based on case-based mechanisms (Cf. Ramalho & Ganascia (1994)) has been developed, and is used in conjunction with a representation of musical actions or PACTs (Cf. Pachet (1991),

Ramalho & Pachet (1994)). The idea is to model the complete sequence of processes involved in improvisation, from the beginning (parsing of the chord sequence, using the chord sequence analyzer mentioned above) up to the actual generation of notes. The generation of PACTs uses the case-based model of memory in conjunction with an algorithm that generates PACTs according to the musician experience, and general knowledge about musical actions.

Figure 1. The polyphonic score editor.

## 6. Discussion, future works

### 6.1. Why could not I reuse MODE classes ?

Reusing the MODE system seemed a good idea, given the range of problems MODE addresses, the fact that it is written in Smalltalk, a language that many consider as the most reusable of all. However, things did not turn out to be so easy. Almost no classes from MODE could be reused, and we would like to give here a few arguments that may explain why.

- Tonal harmony versus "enharmonic" harmony. Enharmonic spelling is considered in MusES as a central issue, as it supports all analysis reasoning. It cannot be made as an extension of any representation that would not take it into account from the very beginning.

- Analysis versus synthesis. As we said, MODE is oriented towards synthesis (of sounds, music, structures), whereas MusES is more intended to support the construction of reasoning systems. This has unexpected practical effects: reification is not done in the same spirit. For instance, we need in MusES to have intervals represented as both objects and operations, and therefore have to represent the corresponding relations with pitch-classes, and octave-dependent notes. This makes the whole music-magnitude classes of MODE unadapted, since these objects are at the bottom of the hierarchy, and support all the system. By comparison, we could imagine how much of the Smalltalk environment would have to be changed if collections were represented by chains of pointers rather than by arrays (as it is the case).

- Representations of time. Our representation of time is also different from the representation of MODE. MODE introduces the notion of *EventList*, as a list of association startTime/musical event. This has the advantage of genericity since the musical events can be any object that define a small set of necessary methods (including event list themselves). However, it has the disadvantage that musical event do not "know" their startTime, either directly (by an instance variable) or indirectly (since they do not have any reference to the eventList that hold them). Our representation of time, based on inheritance, solves this problem, has the major

advantage of being simple to implement, but suffers from the other drawbacks. For instance, information such as the "following" note in a melody is not easily accessible (in either of the representations). We are currently investigating a more convenient representation of time, based on the extensive use of "wrappers".

### 6.2. Conclusion, future works

We described MusES, a knowledge base that represent concepts of basic harmony and their most current operations. We gave an overview of three systems built on top of MusES, that use MusES structures in conjunction with various inference mechanisms. Future works include 1) The connection of the graphical editors with MusiTex, an extension of Latex for musical scores, to generate professional quality scores from our editors, 2) The representation of pitch-classes using the two-dimensional Harmony Space interface described in Holland (89); 3) Continue with the representation of musical rules (counterpoint of simple species), as well as Schenkerian analysis; and 4) Using MusES and its extensions as a tutorial system.

## 7. References

- Ballesta, J. (1994). Contraintes et objets : clefs de voûte d'un outil d'aide à la composition. *Journées d'informatique Musicale*, Bordeaux, march 1994.
- Cointe, P. Rodet, X. (1991) Formes: Composition and Scheduling of Process. In *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*, S. T. Pope, ed. MIT Press.
- Ebcioğlu, K. (1992). An Expert System for Harmonizing Chorales in the Style of J.-S. Bach. In M. Balaban, K. Ebcioğlu & O. Laske (Ed.), *Understanding Music with AI: Perspectives on Music Cognition*, The AAAI Press, California.
- Holland, S. (1994). Learning About Harmony Space: An Overview. M. Smith, A. Smail & G. Wiggins (Ed.), *Music Education: an Artificial Intelligence Perspective*, Springer-Verlag, London.
- Pachet, F. (1991) A meta-level architecture for analysing jazz chord sequences. *International Conference on Computer Music*, pp. 266-269, Montréal, Canada.
- Pachet, F. (1991b) Representing Knowledge Used by Jazz Musicians. *International Conference on Computer Music*, pp. 285-288, Montréal, Canada.
- Pachet, F. (1994). An object-oriented representation of pitch-classes, intervals, scales and chords. *Journées d'informatique Musicale*, Bordeaux, march 1994.
- Pachet, F. (1994b) A Refined Framework for Representing Knowledge Based on Simulation. *Colloque Langages et modèles à objets*, Grenoble, octobre 1994, to be published.
- Pachet, F. (1995) On the embeddability of production systems in object-oriented languages. *Journal of Object-Oriented Programming*, Dec. 1995. To be published.
- Pachet, F. & Perrot, J.-F. (1994). Rule Firing with Metarules. *Software Engineering and Knowledge Engineering*. SEKE '94, Jurmala, Lettonie. Knowledge System Institute Ed. pp. 322-329, 21-23 june 1994.
- Pope, S. (1991). Introduction to MODE: The Musical Object Development Environment. In *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*, S. T. Pope, ed. MIT Press.
- Ramalho, G., Pachet, F. (1994). From real book to real jazz performance. *International Conference on Music Perception and Cognition*, Liège, Belgium, july 1994.
- Ramalho, G., Ganascia, J.-G. (1994). Simulating Creativity in Jazz Performance. *Proc. of 12th AAAI conf.* Seattle, aug. 1994.
- Scaletti, C. (1987). Kyma: An Object-oriented Language for Music Composition. in *Proceedings of the International Computer Music Conference*. International Computer Music Association, San Francisco.
- Steedman, M.J. (1984). A Generative Grammar for Jazz Chord Sequences. *Music Perception*, Fall 1984, Vol. 2, N° 1, pp. 52-77.
- Walker, W., Hebel, K., Martirano, S., Scaletti, C. (1992). ImprovisationBuilder: improvisation as conversation. *Proc. of ICMC*, 1992.
- Winograd, T. (1993). Linguistics and the Computer Analysis of Tonal Harmony. In *Machines Models of Music*, Edited by S. M. Schwanauer and D.A. Levitt, MIT Press.

**ARTIST**  
**An AI-based tool for the design of intelligent assistants for sound synthesis**

*Eduardo Reck Miranda,*  
AI/Music Group,  
Faculty of Music and Dept. of Artificial Intelligence,  
University of Edinburgh,  
12, Nicolson Square,  
Edinburgh, EH8 9DF,  
Scotland, UK.  
E-mail: miranda@music.ed.ac.uk

**Abstract**

In this paper we introduce the fundamentals of ARTIST (an acronym for Artificial Intelligence-aided Synthesis Tool). ARTIST is a tool for the design of intelligent assistants for sound synthesis that allow composition of sounds thought of in terms of qualitative descriptions (e.g. words in English) and intuitive operations rather than low level computer programming. Our research work is looking for (a) plausible strategies to map the composer's intuitive notion of sounds to the parametric control of electronic sound synthesis and (b) how to provide artificial intelligence (AI) to a synthesiser. In this paper we introduce how we attempted to approach the problem by means of a compilation of a few well known expert systems design techniques used in AI research. ARTIST is a prototype system which embodies the results of our investigation so far.

**Keywords:** AI-based synthesiser, knowledge-based systems, machine learning

**Introduction**

In the final quarter of the 20th century the invention of sound recording followed by sound processing and then sound synthesis have changed our view of what constitutes music. These recent developments have vastly expanded our knowledge of the nature of sounds. Nowadays, computer technology offers composers the most detailed control of the internal parameters of sound synthesis and signal processing.

Wanting the effective use of the new technology, composers become more ambitious, but the complexity also increases. The scale and nature of the compositional task changes, technically and aesthetically. Theoretically the computer can be programmed to generate any sound one can imagine. But, on the other hand, this can get composers into trouble. Quoting Barrière (1989, pp. 116), "*it is too easy to fail to take various consequences into account, to get technology side-tracked by a tool whose fascinating complexity can become a disastrous mirage*".

Even if the composer knows the role played by each single parameter for synthesising a sound, the traditional way of working with computer synthesis, tediously entering exact data at the terminal, is not particularly stimulating. We are convinced that higher processes of inventive creativity and musical abstraction are often prejudiced in such a situation. In this case we think that the computer is being used as a kind of word processor combined with player piano, and not as a creative tool. We have come to believe that this can be improved by means of an appropriate coupling between human imagination and artificial intelligence (AI).

In this paper, we introduce the fundamentals of ARTIST (an acronym for Artificial Intelligence-aided Synthesis Tool). ARTIST is a tool for the design of intelligent assistants for sound synthesis that allow composition of sounds thought of in terms of intuitive qualitative descriptions (e.g. words in English) rather than low level computer programming (Miranda et al., 1993a; 1993b; Miranda, 1994a; 1994b; 1994c). By an intelligent assistant we mean a system which works co-operatively with the user by providing useful levels of automated reasoning in order to support laborious and tedious tasks (such as working out an appropriate stream of synthesis parameters for

each desired single sound), and to aid the user to explore possible alternatives when designing a sound. The desirable capabilities of such a system can be summarised as follows:

- The ability to operate the system by means of an intuitive vocabulary instead of sound synthesis numerical values,
- The ability to customise the system according to the user's particular needs, ranging from defining which synthesis technique(s) will be used to defining the vocabulary for communication,
- The encouragement of the use of the computer as a collaborator in the process of exploring ideas,
- The ability to aid the user in concept formation, such as the generalisation of common characteristics among sounds and their classification according to prominent attributes, and
- The ability of creating contexts which augments the chances of something unexpected and interesting happening, such as an unimagined sound out of an ill-defined requirement.

Apart from graphic workstations (such as the UPIC system (Xenakis, 1992; Marino et al., 1993; Lohner, 1986)) and medium level programming languages (see (Pennycook, 1985) for a survey), little research has been done towards a system for sound synthesis that responds to higher levels of sound description. An early attempt at the definition of a grammar for sound synthesis was made by Holtzman (1978) at Edinburgh University. Also, Slawson (1985) has proposed - not implemented on a machine though - a kind of vocabulary for sound composition based on his theory of sound colour which, we believe, he made up from Helmholtz's theory of vowel qualities of tones (Helmholtz, 1885). Lerdaahl (1987) too has done some sketches towards a hierarchical perceptually-orientated description of timbres. Apart from these, it is worth mentioning that there have been a few attempts towards signal processing systems that understand natural language. The most successful ones are interfaces developed to function as a front end for systems which perform tasks to do with audio recording studio techniques such as, mixing, equalisation, and multitracking (e.g. CIMS (Schmidt, 1987) and Elthar (Garton, 1989)). More recently, Ethington and Punch (1994) proposed a software called SeaWave. SeaWave is an additive synthesiser (Dodge and Jerse, 1985) in which sounds can be produced by means of a vocabulary of descriptive terms. Although of a limited scope, SeaWave proffers an excellent insight and it seems to work well. Vertegal and Bonis (1994) also have been working towards a cognitive-orientated interface for synthesisers.

We begin the paper by introducing the problem from a musician's point of view. Then we introduce the signal processing of the synthesiser which will be used as an example study. After this, we indicate some methods for describing sounds by means of their attributes and suggest a technique for mapping those attributes onto the parameters of a synthesiser. Then we study how this technique works and present some examples. Here, we also study the utility and the functioning of machine learning in this kind of system. Finally, we propose a system architecture which embodies all the concepts discussed so far and introduce its functioning through examples. We end the paper with some final remarks and ongoing work.

### An example study synthesiser

Assume that we wish a synthesiser which is able to produce human voice-like sounds. It is worth mentioning that to produce a perfect simulation of the human vocal tract is out of the scope of this paper. Thus, rather than making a description of the fundamental aspects of the phenomenon by means of a set of equations (e.g. (Woodhouse, 1992; Keefe, 1992)), we opted for observing it by means of a more traditional formant modelling technique which uses subtractive synthesis (Flanagan, 1984; Klatt, 1990; Sundberg, 1991; Miranda, 1992; 1993). We come to believe that this level of description (see also (Eckel, 1993) for a brief discussion about this business) suffices at this moment. The signal processing diagram of our example study synthesiser is shown in Figure 1.

Each block of the diagram (except the *Envelope*) is composed of several signal processing units (SPU). A composition of SPU's form sub-blocks within a block. Sub-blocks in turn may constitute sub-sub-blocks, and so forth. The *Voicing Source*, for example, has two sub-blocks: one, the *Vibrato sub-block*, contains an oscillator unit, and the other, the *Pulse Generator sub-block*, contains a pulse generator unit (Figure 2).

Each SPU needs parameter values for functioning. We say that, in order to produce a certain

Figure 1: The example study architecture.

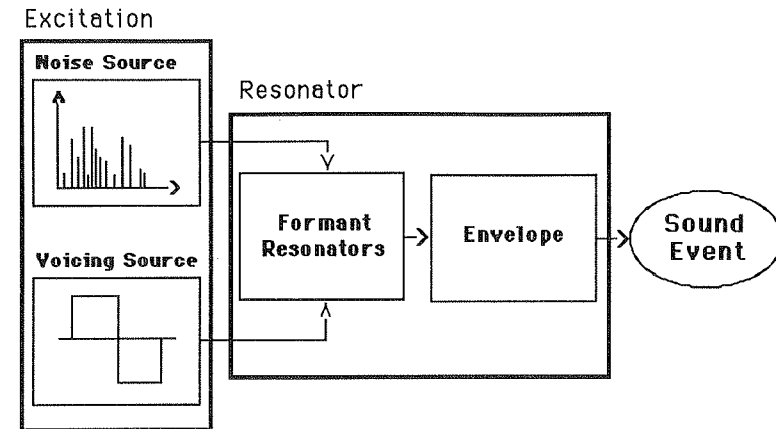
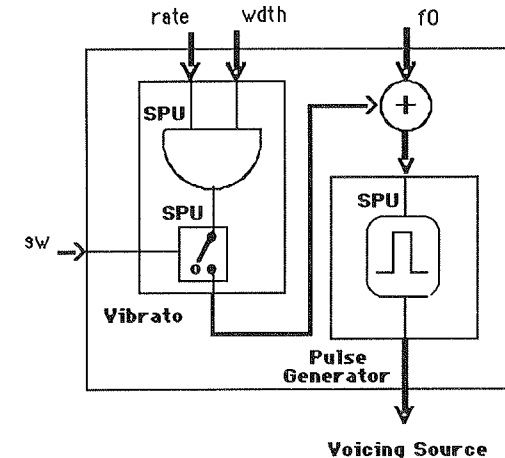


Figure 2: The voicing source block.



### Describing sound by means of their attributes

There have been several studies defining a framework to systematically describe sounds by means of their attributes ((Schaeffer, 1966; von Bismark, 1971; 1974a; 1974b; Cogan, 1984; Giomi & Ligabue, 1992; Carpenter, 1990; Terhardt, 1974) to cite but a few). They are derived mainly from work in the fields of both psychoacoustics and musical analysis. We classify these studies in two approaches: on the one hand, the *device-orientated* approach and, on the other hand, the *perceptually-orientated* approach. As it is not our aim to survey all these, we have selected one example of a



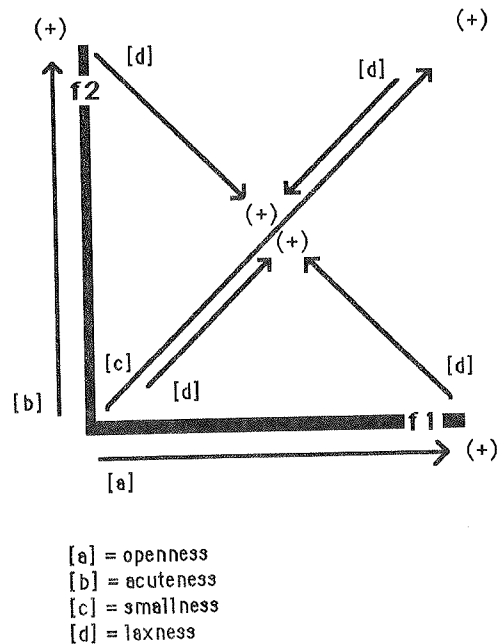
### The source-filter model: a device-orientated approach

The source-filter model formulates that the characteristic of a sound is determined by its spectrum envelope's pattern. This pattern is composed of multiple hills called formants. Each formant has a centre frequency peak and a bandwidth. According to this model, the lowest two formants are the most significant determinants of sound quality.

The pattern of the spectrum envelope of formant frequencies is thought of as the result of a complex filter through which a source sound passes.

We can define here a two-dimensional space whose axes are the first (f(1)) and the second (f(2)) centre formant frequencies respectively. Then, four perceptual attributes, namely *openness*, *acuteness*, *smallness*, and *laxness* (after Slawson, 1985; 1987), can be specified as categories of equal-values contours in this space. The attribute *openness* varies with the sum of f(1) + f(2), and *laxness* varies towards a neutral position in the middle of the space (Figure 3).

Figure 3: Two-dimensional sound space.



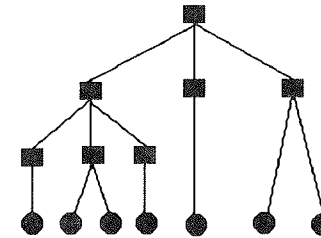
### The notion of Abstract Sound Schema(ASS)

The Abstract Sound Schema (ASS) is the representation scheme we designed for describing a sound in terms of its perceptual components and the relations between them. The ASS scheme is constituted of: nodes, slots, and links. Nodes and slots are the components, and the links correspond to the relations between them. The links are labelled.

The ASS is, in fact, a tree-like abstract data structure whose ultimate nodes (the leaves) are slots. Each slot has a name and accommodates a sound synthesis datum.

Slots are grouped bottom up into higher level nodes, which in turn are grouped into higher level nodes (Figure 4).

Figure 4: The ASS representation scheme.



### Implementing a sound event by means of the schema

We have seen before (Figure 1) that the synthesiser is composed of several connected blocks (Voicing Source, Noise Source, etc.), one of each responsible for a certain sound attribute. We can now define a compound *sound event* by means of the ASS scheme. Each component of the *sound event* is responsible for a certain aspect of the sound quality.

The leaves of the *sound event* are slots corresponding to the several sound synthesis parameters. Slots are grouped into nodes of a higher level layer, which in turn are grouped into nodes of a higher level, and so forth, up to the root of the tree (the *sound event*).

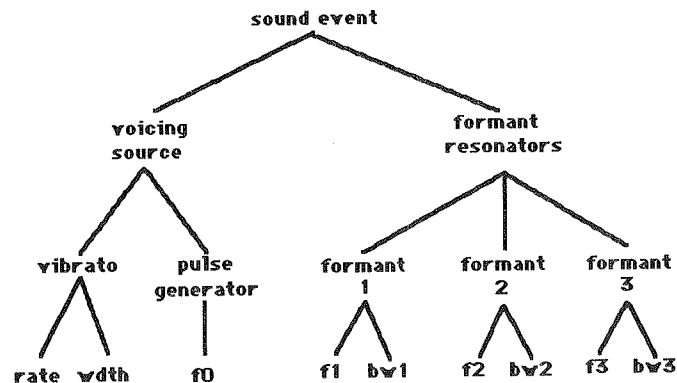
Figure 5 shows a partial definition of a *sound event* of the synthesiser shown in Figure 1. Although not shown in this figure, the links among the *sound-event's* components are labelled *has\_component*. They represent the offspring relation among nodes.

The partial *sound event* definition shown in Figure 5 can be implemented in Prolog as shown below. Each clause represents a *has\_component* relationship between two atoms. The first clause, for example, is read: 'a *sound event* has a component called *voicing source*'. A interpretation of the whole layer 1, for example, is: 'the *sound event* has two components named *voicing source* and *formant resonators*'.

```

%%% layer 1
%%%
has_component(sound_event, voicing_source).
has_component(sound_event, formant_resonators).
%%%
%%% layer 2
%%%
has_component(voicing_source, vibrato).
has_component(voicing_source, pulse_generator).
has_component(formant_resonators, formant(1)).
has_component(formant_resonators, formant(2)).
has_component(formant_resonators, formant(3)).
%%%
%%% layer 3
%%%
has_component(vibrato, rate).           % vibrato rate
has_component(vibrato, width).         % vibrato width
has_component(pulse_generator, f(0)).  % fundamental frequency
has_component(formant(1), f(1)).       % 1st formant frequency
has_component(formant(1), bw(1)).      % 1st formant bandwidth
has_component(formant(2), f(2)).       % 2nd formant frequency
has_component(formant(2), b(w2)).     % 2nd formant bandwidth
has_component(formant(3), f(3)).       % 3rd formant frequency
has_component(formant(3), bw(3)).     % 3rd formant bandwidth
    
```

Figure 5: Partial sound event definition.



All the slots of the ASS must be filled in order to completely specify a sound. We say that a completely specified sound is an *assemblage*. For each different sound there is a particular assemblage. Thinking of this synthesiser as a (rough) model of the vocal tract mechanism, an assemblage would correspond to a certain position of the vocal tract in order to produce a sound.

### Sound hierarchy and the inheritance mechanism

Recapitulating, we have defined a general abstract scheme for representing a sound. Then we defined and implemented the notion of the *sound event* by means of this scheme. We also introduced the idea of assemblage. It was explained that an assemblage occurs when all the slots of the scheme are properly filled. In this case, each assemblage corresponds to a particular sound.

In practice, sounds are represented in a knowledge base as a collection of slot values. In other words, the knowledge for the assemblage of a particular sound is clustered around a collection of slot values. An assemblage engine is then responsible for taking the appropriate slot values and 'assembling' the desired sound.

The following Prolog facts correspond to an example knowledge base which contains slot values for the (partial) *sound event* definition shown in Figure 5. Each clause represents a *slot*. It has two atoms: the first is a reference name and the second is a tuple. The reference name is an atom which identifies the affiliation of the slot, i.e. which cluster it belongs to. The first element of the tuple is the name of the slot and the second element is the value of the slot. This value can be either a number, a word, or a formula for calculating its value (these will be dealt later). This example knowledge base contains information about three sounds, namely *back vowel*, *front vowel*, and *vowel /a/*.

```

%%% back vowel
%%%
slot( vowel(back), [ rate, 5.2 ] ).
slot( vowel(back), [ width, 0.06 ] ).
slot( vowel(back), [ f(0), 155.56 ] ).
slot( vowel(back), [ f(1), 622.25 ] ).
slot( vowel(back), [ f(2), 1244.5 ] ).
slot( vowel(back), [ f(3), 2637 ] ).
slot( vowel(back), [ bw(1), 74.65 ] ).
slot( vowel(back), [ bw(2), 56 ] ).
slot( vowel(back), [ bw(3), 131.85 ] ).
%%%
%%% front vowel
%%%
slot( vowel(front), [ rate, 5.5 ] ).
slot( vowel(front), [ width, 0.06 ] ).

```

```

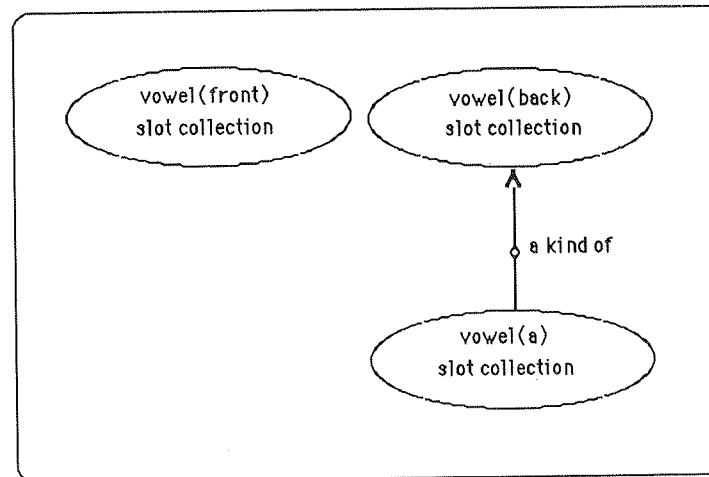
slot( vowel(front), [ f(1), 559.37 ] ).
slot( vowel(front), [ f(2), 1108.7 ] ).
slot( vowel(front), [ f(3), 2637 ] ).
slot( vowel(front), [ bw(1), 78.3 ] ).
slot( vowel(front), [ bw(2), 110.8 ] ).
slot( vowel(front), [ bw(3), 131.85 ] ).
%%%
%%% vowel /a/
%%%
slot( vowel(a), [ a_kind_of, vowel(back) ] ).
slot( vowel(a), [ f(0), 103.83 ] ).

```

Note that the representation of the sound *vowel(a)* is different from the other two: it is incomplete (i.e. there are no slot values for the *vibrato* nor for the *formant resonators*). On the other hand, there is new information in it. The new information, called *a\_kind\_of*, is not a simple *sound event* slot, as it might appear to be, but it is a link (see Figure 6). This is a link which associates one collection of slots with other collection of slots.

The link *a\_kind\_of* allows a hierarchical organisation of the knowledge. The ability to represent the relationship between slot collections hierarchically is useful for inheritance relation. Inheritance is a relation by which an individual assumes the properties of its class and by which properties of a class are passed on to its subclass. Thus, when a slot collection for a sound is attached to another slot collection at a higher level, the former inherits properties of the latter. The first fact of the third cluster of slots listed above states that a *vowel(a)* is a *kind\_of* *vowel(back)*. This is to say that slots not defined for *vowel(a)* will be filled with slot values taken from *vowel(back)* (Figure 6). In practice, the assembler engine has to 'know' that the missing slots in one level are inherited from a higher level.

Figure 6: The example knowledge base has information about three sounds. Each sound is represented as a collection of slot values. Note that *vowel(a)* inherits slots from *vowel(back)*.



knowledge base

### The notion of partial assemblage

We ought to make the assembler engine flexible so that it also may assemble single internal nodes of the schema. In other words, besides the assemblage of the whole scheme there might be (partial) assemblages of only certain nodes.

Let us observe again the example shown in Figure 5. It has a branch of filters which constitute three formant resonators. Taking as an example only the node **formant(1)**, we say that it needs only its affiliated slots (namely **f(1)** and **bw(1)**) for assemblage.

The advantage of being able to think in terms of assemblages of single nodes, as an alternative to the solely ASS root assemblage, is that now one can attach non-numerical attribute values (i.e. words in English) to partial assemblages too. For instance, one could refer to the node **formant(1)** as **low and wide** if it has **f(1) = 250 Hz** and **bw(1) = 200 Hz**. This is also represented in the knowledge base as a cluster of slots. Example:

```
slot([ formant(1), low_and_wide ], [ f(1), 250 ] ).
slot([ formant(1), low_and_wide ], [ bw(1), 200 ] ).
```

Now, for each node of the schema one can define a set of possible non-numerical attribute values. Back to figure 5, the slots **rate**, and **wdth** constitute a node called **vibrato** which in turn, with the node **pulse generator**, forms the higher level node **voicing source**. One could establish here that the possible attribute values for **vibrato** are **none**, **uniform**, and **too slow**. Each of these attributes will then correspond to either a numerical value or to a range of values within a certain interval. For example, one could say that **vibrato** is **none** if **rate = 0 Hz**, and **wdth = 0 %**. The node **voicing source** could be similarly defined: one could establish that **voicing source** is **steady low** if **vibrato = none** and **pulse generator = 55 Hz**, for example.

Hypothetically considering only this left part of the example schema shown in Figure 4, a sound, say **sound(a)**, could be described as *having steady low voicing source and none vibrato*. See example below:

```
slot([ vibrato, none ], [ rate, 0 ] ).
slot([ vibrato, none ], [ wdth, 0 ] ).
slot([ voicing_source, steady_low ], [ vibrato, none ] ).
slot([ voicing_source, steady_low ], [ pulse_generator, 55 ] ).
slot([ sound(a), [ voicing_source, steady_low ] ],
slot([ sound(a), [ vibrato, none ] ).
```

### The role of machine learning

In this section we will study the role played by two machine learning techniques in our proposed system, namely *inductive learning* and *supervised deductive learning*. Both are well known techniques which have been satisfactorily used in expert systems (see (Dieterich and Michalski, 1981; Quinlan, 1982; Winston, 1984; Bratko, 1990; Carbonell, 1990) for a survey).

The target of inductive learning here is to induce general concept descriptions of sounds from a set of examples. A further aim is to allow the computer to use automatically induced concept descriptions in order to identify unknown sounds or possibly suggest missing attributes of an incomplete sound description. Our main reason for inducing rules about sounds is that the computer can then aid the user to explore among possible alternatives during the design a certain sound. Here the user would be able to ask the system to 'play something that sounds similar to a bell' or even 'play a kind of dull sound', for example. In these cases the system will consult induced rules in order to work out which attributes are relevant for synthesising a bell-like sound or a sound with dull colour attribute (Smaill et al. 1993).

An example rule, when looking for a description for, say **sound(c)**, on the basis of some examples, could be as follows:

```
sound(c) = [ [ vibrato = fast ], [ openness = high ] ]
```

The interpretation of the above rule is as follows:

```
A sound is sound(c) if:
it has fast vibrato and
high openness.
```

No matter how many attributes **sound(c)** had in the training set, according to the above rule, the most relevant attributes for this sound are **vibrato = normal** and **openness = high**. 'Most relevant' here means what is most important for distinguishing **sound(c)** from other sounds of the input training set. In this case, if the system is asked to synthesise a sound with **fast vibrato** and **high openness**, then it will produce **sound(c)**.

The target of supervised deductive learning in our system is to allow the computer to update its knowledge about attribute values throughout user interaction. We remind the reader the fact that the input requirement for producing a sound can contain either or both: attribute values (e.g. **vibrato = none**) or slot values (e.g. **f(0) = 55 Hz**). The aim of supervised deductive learning here is at allowing the computer to infer whether or not input slot values (in a requirement) match with known attribute values. If there is no matching, then the system automatically adds this yet unknown information to the knowledge base and asks the user to give a name for this novel deduced attribute value. Suppose that the system knows three values for the attribute **vibrato**:

```
vibrato = uniform      if { rate = 5.2 Hz, wdth = 3 % }
vibrato = too slow    if { rate = 3.6 Hz, wdth = 3 % }
vibrato = none        if { rate = 0 Hz, wdth = 0 % }
```

If the user requires a sound with (**vibrato**) **rate = 12 Hz**, for example, then the system will synthesise it and deduce that there is no attribute value for **vibrato** in the knowledge base whose **rate** is equal **12 Hz**. In this case the system adds this new information to the knowledge base, works out the other slot values needed to create this new attribute value, and ask the user to name it. Let us say, for example, that the user wishes to call it **tremolo**. Eventually the system will add the following information in its knowledge base:

```
vibrato = tremolo      if { rate = 12 Hz, wdth = 3 % }
```

### Towards a system architecture

User configuration is one of the desirable capabilities of this system. Therefore rather than providing a closed architecture which reflects both a particular synthesiser and a particular vocabulary for sound description, we propose an architecture which provides open-ended modules (Figure 7).

This system architecture provides means for handling information about sound synthesis but it remains open-ended regarding what the information is about.

### Engines and services provided by the system

The role of the *assembler engine* and the functioning of the *machine learning engine* modules have already been introduced.

The *machine learning engine* module performs the two kinds of learning: *inductive learning* and *supervised deductive learning*, just discussed above. The training set for the inductive learning mechanism is given either by the user or it is automatically produced by the system by consulting its own knowledge base. The input for the supervised deductive learning mechanism is provided partly by the system and partly by the user.

The *user interface* module provides means for communicating with the system. Here the user can activate the *assembler engine* in order to produce a sound, consult the status of the system (such as the content of the *induced rules* module and the content of the *knowledge base* module), and input any external information the system might need (such as the names for new sounds and attributes, and training sets).

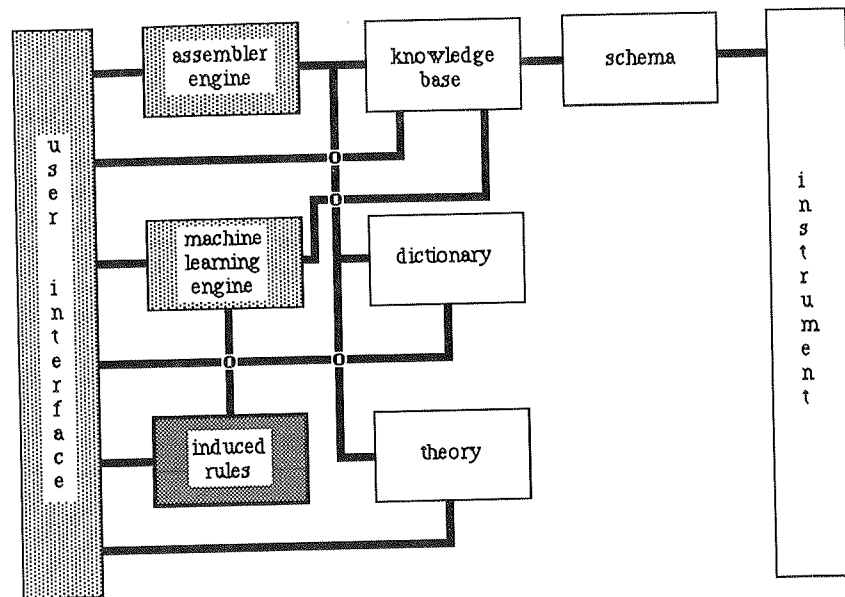
### Information internally generated and administered by the system

The *inductive rules* module holds the information internally generated by the system as the result of the inductive learning. As its name suggests this module contains rules which were induced by the *machine learning engine* module.

### User specified modules

These are the open-ended modules. They define the domain of the system, that is, the sonic world the system will deal with. Here the user implements the instrument (i.e. the synthesis algorithm), the schema on the top of it, the knowledge base whose information is used to 'play' it, a dictionary of slot values, and a theory for the instrument, using Prolog (Bratko, 1990).

Figure 7: The proposed system architecture.



The modules of the architecture are classified in three groups:

- User specified modules
- Information automatically generated by the system
- Engines and services provided by the system

Default libraries of such modules can be provided in case the user does not wish start from scratch. However, as these modules are to be user customised, it may not always be very useful to exchange highly customised libraries with other users.

Firstly, the user specifies the *instrument* module. This can be done by means of any suitable SWSS (Software for Sound Synthesis) package, such as CLM, (Schotstaedt, 1992), Csound (Vercoe, 1991), Mosaic (Morrison and Waxman, 1991), or ISPW Max (Puckette et al., 1992), to name but a few. Having specified the instrument then the user implements the *schema* on the top of it. Secondly, the *knowledge base* module is specified. In this module the user creates clusters of slot values. As it was mentioned before, each cluster corresponds to an instantiation of either a whole sound event or an internal node of the schema, i.e. a sound attribute. Thirdly, the user builds the *dictionary* module. In this module the user specifies the meaning of the vocabulary for speaking about slots, i.e. about each parameter of the instrument. Each word of the vocabulary for slots may mean either a numerical synthesis parameter or a pointer to a formula for calculating it. Finally, the user specifies a *theory* for the instrument. A theory is a set of formulas for calculating slot values. These formulas can calculate values either based on other slot values or by the random choice of a value within a certain interval.

As the system is to start with a certain body of knowledge which will be expanded through user interaction, these specifications do not need to be exhaustive.

### An example functioning

Let us study an example functioning of the architecture explained above. Assume that the following information can be used in order to assemble the scheme of Figure 5.

#### Knowledge base module:

```
slot( sound_event( sound(c) ), [ rate, fast ] ).
slot( sound_event( sound(c) ), [ width, default ] ).
slot( sound_event( sound(c) ), [ f(0), low ] ).
slot( sound_event( sound(c) ), [ openness, high ] ).
slot( sound_event( sound(c) ), [ acuteness, low ] ).
slot( sound_event( sound(c) ), [ f(3), 2637 ] ).
slot( sound_event( sound(c) ), [ bw(3), 131.85 ] ).

slot( attribute( [ openness, low ] ), [ f(1), low ] ).
slot( attribute( [ openness, low ] ), [ bw(1), 74.65 ] ).

slot( attribute( [ openness, high ] ), [ f(1), high ] ).
slot( attribute( [ openness, high ] ), [ bw(1), 78.3 ] ).

slot( attribute( [ acuteness, low ] ), [ f(2), low ] ).
slot( attribute( [ acuteness, low ] ), [ bw(2), 110.8 ] ).

...
etc.
```

#### Dictionary module:

```
dict( slot( f(1) ), [ value( low, 290 ),
                    value( medium, 400 ),
                    value( high, 650 ) ] ).

dict( slot( f(2) ), [ value( low, 1028 ),
                    value( medium, 1700 ),
                    value( high, 1870 ) ] ).

dict( slot( f(0) ), [ value( low, 220 ),
                    value( medium, rule( f(0), medium ) ),
                    value( high, rule( f(0), high ) ) ] ).

...
etc.
```

#### Theory module:

```
instrument_theory( rule( f(0), medium ), F0 ):-
    get_value( f(0), low, V ),
    F0 is V * 2.

...
etc.
```

Suppose that a training set has been input and the system has already induced some rules, such as:

```
sound(a) = { [ openness = low ] }
sound(b) = { [ f(0) = medium ] }
sound(c) = { [ rate = fast ], [ f(0) = low ] }

...
etc.
```

Now, let us suppose two hypothetical queries and examine what ARTIST would do in order to compute them.

Example query 1:

Produce a sound with fast vibrato rate and low pitch.

ARTIST functioning 1:

Firstly, the system consults the induced rules in order to find out if it knows any sound whose most prominent features are *rate = fast* and *f(0) = low*. In this case, there is a rule which tells that *sound(c)* matches this requirement. Thus, *sound(c)* will be produced. Before assembling the schema, the system consults the dictionary in order to compute the slots whose values are represented by a word (e.g. *f(0) = low* actually means 220 Hz).

Example query 2:

Produce a sound with medium pitch and high openness.

ARTIST functioning 2:

In this case the system has no matching induced rules. Thus, this sound will be created from scratch. The system consults the dictionary in order to compute the values of *f(0) = medium* and *f(1) = high*, and automatically completes the missing slot data with default values. Note that instead of a value for *f(0) = medium*, the dictionary points to a rule. In this case, the system consults the theory module in order to calculate it. The theory says that this value corresponds to the double value of *f(0) = low*. Therefore, *f(0) = medium* here means 440 Hz. The sound is then produced, the user is asked to name it, and a novel cluster of slot values is automatically created in the knowledge base for representing it.

### Conclusion and further work

In this paper we introduced the fundamentals of ARTIST: a tool for the design of intelligent assistants for sound synthesis.

ARTIST is provided with some degree of automated reasoning which supports the laborious and tedious task of writing down number sequences for generating a single sound on a computer. A 'synthesiser' implemented by means of ARTIST is provided with a certain knowledge about sound synthesis and it is able to infer the necessary parameters values for a sound from a quasi-natural language sound description.

Although the user has to specify the information of the knowledge base (i.e. the synthesis algorithm(s) and the vocabulary for sound description) beforehand, this does not necessarily need to be exhaustive. The system is to begin with a minimum amount of information about certain sounds and attributes, but it is able to automatically expand the scope of its knowledge by acquiring new information through user interaction.

At the moment we are developing a higher level interface for the user specified modules. We wish to enable the user to specify these modules by means of natural language-like statements, instead of Prolog. We also plan to devise a visual interface for the specification of some attributes, such as envelopes, for example.

ARTIST is being tested using synthesis by physical modelling technique (Roads, 1993). It seems that this technique matches many of the concepts developed in this paper, such as the representation of sound attributes and the mapping of these to synthesis parameters.

We are aware that ARTIST is still in its infancy. For the moment we regard it as a suggestive and plausible starting point only.

### References

- Barrière, J.-B. (1989), Computer music as cognitive approach: Simulation, timbre and formal processes, in *Contemporary Music Review*, Vol. 4, pp. 117-130, Harwood Academic Publishers.
- Bratko, J. (1990), Prolog programming for Artificial Intelligence, Addison-Wesley Publishers.
- Carbonell, J. (1990) (Editor.), *Machine Learning: paradigms and methods*, The MIT Press.
- Carpenter, R. H. S. (1990), *Neurophysiology*, Physiological Principles of Medicine Series, Edward Arnold.
- Cogan, R. (1984), *New Images of Musical Sound*, Harvard University Press.
- Dodge, C. and Jerse, T. (1985), *Computer Music*, Schirmer Books.
- Dobiesch, T. and Michalski, R. (1981), Inductive Learning of Structural Descriptions, in *Artificial Intelligence*,

- Eckel, G. (1993), La Maître de la Synthèse Sonore, in *La Synthèse Sonore*, Les cahiers de l'Ircam Nr. 2, pp. 97-106, Ircam - Centre Georges Pompidou.
- Ethington, J. and Punch, D. (1994), SeaWave: A system for Musical Timbre Description, in *Computer Music Journal*, Vol. 18, Nr. 1, pp. 30-39, The MIT Press.
- Flanagan, F. (1984), Voices of Men and Machines, in *Electronic Speech Synthesis*, Bristow, G. (Editor.), Granada.
- Garton, B. (1989), The Eltham Program, in *Perspectives of New Music*, Vol. 27, Nr. 1, pp. 6-41.
- Glomi, F. and Ligabue, M. (1992), *Analisi Assistita al Calcolatore della Musica Contemporanea*, Rapporto Interno C92-01, CNUCE/CNR, Conservatorio di Musica L. Cherubini (Italy).
- Helmholtz, H. L. F. (1885), *On the sensations of tone as a physiological basis for the theory of music*, Longmans, Green and Co.
- Holtzman, S. R. (1978), A description of an automated digital sound synthesis instrument, *DAI Research Report No. 59*, Dept. of AI, University of Edinburgh.
- Klatt, D. H. (1980), Software for a cascade/parallel formant synthesiser, in *Journal of Acoustic Society of America*, Vol. 67, Nr. 3, pp. 971-995.
- Keefe, D. H. (1992), Physical Modeling of Wind Instruments, in *Computer Music Journal*, Vol. 16, Nr. 4, pp. 57-73, The MIT Press.
- Lerdhal, F. (1987), Timbral Hierarchies, in *Contemporary Music Review*, Vol. 2, pp. 135-160, Harwood Academic Publishers.
- Lohner, H. (1986), The UPIC System: A User's Report, in *Computer Music Journal*, Vol. 10, Nr. 4, pp. 42-49, The MIT Press.
- Luger, G. F. and Stubblefield, W. A. (1989), *Artificial Intelligence and the design of Expert Systems*, Benjamin/Cummings.
- Marino, G., Serra, M.-H., and Raczinski, J.-M. (1993), The UPIC System: Origins and Innovations, in *Perspectives of New Music*, Vol. 31, Nr. 1, pp. 258-269.
- Miranda, E. R., Smail, A., and Nelson, P. (1993a), A Symbolic Approach for the design of Intelligent Musical Synthesizers, in *Proceedings of the X Reunión Nacional de Inteligencia Artificial in Mexico City*, Megabyte/Noriega Editores.
- Miranda, E. R., Smail, A., and Nelson, P. (1993b), A Knowledge-based approach for the design of Intelligent Musical Instruments, in *Proceedings of the X Simpósio Brasileiro de Inteligência Artificial in Porto Alegre*, pp. 181-196, SBC/UFRGS.
- Miranda, E. R. (1992), *Towards an Acousmatic Singer*, Research Report Nr. 1, Faculty of Music, University of Edinburgh.
- Miranda, E. R. (1993), Modelagem do Aparelho Fonador e suas Aplicações na Música, in *Acústica & Vibrações*, Journal of the Acoustic Society of Brazil (SOBRAC), Nr. 12, pp. 60-74.
- Miranda, E. R. (1994a), From Symbols to Sound: Artificial Intelligence Investigation of Sound Synthesis, in *Contemporary Music Review* (in press), Harwood Academic Publishers.
- Miranda, E. R. (1994b), The Role of Artificial Intelligence in Computer-aided Sound Composition, in *Journal of Electroacoustic Music* (in press), Sonic Arts Network.
- Miranda, E. R. (1994c), Towards an Intelligent Assistant for Sound Design, in *Musical Praxis*, Vol. 1, Nr. 1, pp. 53-57, Faculty of Music, Edinburgh University.
- Morrison, J. and Waxman, D. (1991), *Mosaic 3.0 Reference Manual*, Ircam.
- Pennycook, B. W. (1985), Computer-Music interfaces: A Survey, in *Computing Surveys*, Vol. 17, Nr. 2, pp. 267-289.
- Puckette, M., Lippe, C., and Waxman, D. (1992), *ISPW Max Reference Manual*, Preliminary Release 0.17, Ircam.
- Roads, C. (1993), Initiation à la Synthèse par Modèles Physiques, in *La Synthèse Sonore*, Les cahiers de l'Ircam Nr. 2, pp. 145-172, Ircam - Centre Georges Pompidou.
- Quinlan, J. R. (1982), Semi-autonomous Acquisition of Pattern-based Knowledge, in *Introductory Reading in Expert Systems*, Michie, D. (Editor), Gordon & Breach.
- Schaeffer, P. (1966), *Traité des objets musicaux*, Ed. du Seuil.
- Schmidt, B. L. (1987), Natural Language Interface and their application to Music Systems, in *Proceedings of the 5th Audio Engineering Society International Conference*, pp. 198-206.
- Schotstaedt, W. (1992), *Common Lisp Music Documentation*, available via Internet ftp from the clm directory on the host machine ccrma-ftp.Stanford.edu.
- Slawson, W. (1985), *Sound Color*, University of California Press.
- Slawson, W. (1987), Sound-color Dynamics, in *Perspectives of New Music*, Vol. 25, No. 1&2, pp. 156-179.
- Smail, A., Wiggins, G. A., Miranda, E. R. (1994), Music Representation - between the Musician and the Computer, in *Music Education: An Artificial Intelligence Approach*, Smith, M. et al. (Editors.), Workshops in Computing Series, Springer-Verlag, pp. 108-119.
- Spender, N. (1980), Psychology of music (I-III), in *The New Grove's Dictionary of Music and Musicians*, Sadie, S. (Editor), Vol. 15, pp. 388-427, Macmillan Publishers.
- Sundberg, J. (1991), Synthesising Singing, in *Representation of Musical Signals*, De Poli et al. (Editors), The MIT Press.
- Terhardt, B. (1974), On the Perception of Periodic Sound Fluctuation (Roughness), in *Acustica*, Vol. 30, pp. 201-213.
- Vercoe, B. (1991), *Csound Manual*, available via Internet ftp from the music directory on the host machine

- Vertegal, R. and Bonis, E. (1994), ISEE: An Intuitive Sound Editing Environment, in *Computer Music Journal*, Vol. 18, Nr. 2, The MIT Press.
- von Bismark, G. (1971), Timbre of Steady Sounds: Scaling of Sharpness, in *Proceedings of the 7th International Congress on Acoustics in Budapest*, Vol. 3, pp. 637-640.
- von Bismark, G. (1974a), Timbre of Steady Sounds: A Factorial Investigation of its Verbal Attributes, in *Acustica*, Vol. 30, pp. 146-158.
- von Bismark, G. (1974b), Sharpness as an Attribute of the Timbre of Steady Sounds, in *Acustica*, Vol. 30, pp. 159-172.
- Winston, P. (1984), *Artificial Intelligence*, (2nd ed.), Addison-Wesley.
- Woodhouse, J. (1992), Physical Modeling of Bowed Strings, in *Computer Music Journal*, Vol. 16, Nr. 4, pp. 43-56, The MIT Press.
- Xenakis, I. (1963), Musiques Formelles, in *La Revue Musicale*, double issue Nrs. 253-254, Editions Richard-Masse.
- Xenakis, I. (1971), *Formalized Music: Thought and Mathematics in Music Composition*, Indiana University Press.
- Xenakis, I. (1992), *Formalized Music: Thought and Mathematics in Music*, Pendragon Press, revised edition.

## Representing Musicians' Actions for Simulating Improvisation in Jazz

Geber Ramalho

LAFORIA-IBP-CNRS  
Université Paris VI  
4, Place Jussieu  
75252 Paris Cedex 05 - FRANCE  
Tel. (33-1) 44.27.37.27  
Fax. (33-1) 44.27.70.00  
e-mail: ramalho@laforia.ibp.fr

### Abstract

This paper considers the problem of simulating Jazz improvisation and accompaniment. Unlike most current approaches, we try to model the musicians' behavior by taking into account their experience and how they use it with respect to the evolving contexts of live performance. To represent this experience we introduce the notion of *Musical Memory*, which exploits the principles of Case-Based Reasoning (Schank & Riesbeck 1989). To produce live music using this *Musical Memory* we propose a problem solving method based on the notion of PACTs (*Potential ACTions*) (Ramalho & Ganascia 1994b). These PACTs are a generic framework for representing the musical actions that are activated according to the context and then combined in order to produce notes.

### 1 - Introduction

This paper considers the problem of simulating the behavior of a bass player in the context of Jazz live performance. We have chosen to work on Jazz improvisation and accompaniment because of their spontaneity, in contrast to the formal aesthetic of contemporary classical music composition. From an AI point of view, modeling Jazz performance raises interesting problems since performance requires both theoretical knowledge and great skill. In addition, Jazz musicians are encouraged to develop their musical abilities by listening and practicing rather than studying in *conservatoires* (Baker 1980).

In Section 2 we present briefly the problems of modeling musical creativity in Jazz performance. We show the relevance of taking into account the fact that musicians integrate rules and memories dynamically according to the context. In Section 3 we introduce the notion of PACTs, the basic element of our model. In Section 4, we give a general description of our model and show particularly how the composition module integrates the two above-mentioned notions to create music. In the last section we discuss our current work and directions for further developments.

### 2 - Modeling Musical Creativity

#### 2.1 - The Problem and the Current Approaches

The tasks of improvisation and accompaniment consist in playing notes (melodies and/or chords) according to guidelines laid down in a given chord grid (sequence of chords underlying the song). Musicians cannot justify all the local choices they make (typically at note-level) even if they have consciously applied some strategies in the performance. This is the greatest problem of modeling the knowledge used to fill the large gap referred to above (Ramalho & Pachet 1994). To face this problem, the first approach is to make random-oriented choices from a library of musical patterns weighted according to their frequency of use (Ames & Domino 1992). The second approach focuses on very detailed descriptions so as to obtain a complete explanation of musical choices in terms of rules or grammars (Steedman 1984). Regardless of its musical results, the random-based approach

cannot provide an accurate understanding of musical knowledge, since no explicit semantics is associated to randomness. On the other hand, the deterministic framework of the logic-based approach lacks of flexibility for modeling musical creativity. This crucial trade-off between "flexibility and randomness" and "control and semantics" affects the modeling of other creative activities too (Rowe & Partridge 1993).

## 2.2 - Claims on Knowledge and Reasoning in Jazz Performance

Our first claim is that Jazz musicians' activities are supported by two main knowledge structures: memories and rules. Jazz musicians use rules they have learned in schools and through Jazz methods (Baudoin 1990). However, these rules do not embody all knowledge. In fact, despite the availability of some rules for manipulating abstract concepts such as tension, style, swing, contour, density, contrast, etc., there is no logical rule chaining that can directly and uniquely instantiate these concepts in terms of notes. This phenomenon is a consequence of the Jazz learning process which involves listening to and imitating performances of great Jazz stars (Baker 1980).

To put it in a nutshell, musicians integrate rules and memories into their actions dynamically (Ramalho & Ganascia 1994a). Sometimes, the notes can be determined from their most abstract concepts by means of rules but, very often, these rules are not available. In these cases a fast search for appropriate musical fragments in the musician's auditory memory is carried out. This memory search is both flexible and controlled because of the mechanism of partial matching between the memory contents and requirements stated the available rules and concepts. In terms of modeling, this is an alternative approach that avoids the need for "artificial" rules or randomness.

Our second claim is that musical actions depend strongly on contexts that evolve over time. The great interaction between either musicians themselves or musicians and the public/environment may lead them to reinforce or discard their initial strategies while performing. The constraints imposed by real-time performance force musicians to express their knowledge as a fast response to on-going events rather than as an accurate search for "the best musical response". Jazz creativity occurs within the continuous confrontation between the musician's background knowledge and the context of live performance.

## 3 - PACTs: the Basic Notion of our Model

### 3.1 - Introduction

Pachet (Pachet 1990) has proposed the notion of PACTs (Potential ACTIONs) as a generic framework for representing the potential actions that musicians may take within the context of performance. Focusing the modeling on musical actions rather than on the syntactic dimension of notes, additional knowledge can be expressed. In fact, PACTs can represent not only notes but also incomplete and abstract actions, as well as action chaining. It is this homogeneous representation of both notes and their related abstract concepts PACTs that allows the integration of analytic (rule-based) and analogical (case-based) reasoning.

More precisely, PACTs are frame-like structures whose *main attributes* are: start-beat, end-beat, dimensions, abstract-level, type and instrument-dependency. PACTs are activated at a precise moment in time and are of limited duration which can correspond to a chord, a bar, the entire song, etc. PACTs may rely on different dimensions of notes: rhythm (r); amplitude (a); pitch (p) and their arrangements. When its dimensions are instantiated, the abstract level of a PACT is *low*, otherwise it is *high*. For instance, "play loud", "play this rhythm" and "play an ascending arpeggio" are low-level PACTs on amplitudes, rhythm and pitches respectively. "Play this lick transposed one step higher" is a low-level PACT on all three dimensions. "Play syncopated" and "use major scale" are high-level on respectively rhythm and pitches. PACTs can be of two types: *procedural* (e.g. "Play this lick transposed one step higher") or *declarative* (e.g. "play bluesy"). PACTs may also depend on the instrument. For example, "play five-note chord" is a piano PACT whereas "play stepwise" is a bass PACT.

For the sake of simplicity we have not presented many other descriptors that are needed according to the nature and abstract level of the PACTs. For instance, Pitch-PACTs have descriptors such as pitch-contour (ascending, descending, etc.), pitch-tessitura (high, low, middle, etc.), pitch-set (triad, major scale, dorian mode, etc.) and pitch-style (dissonant, chord-based, etc.).

### 3.2 - PACTs as basis of the problem solving method

From the above description two important properties of PACTs appear. The first one is the *playability* of a PACT. The less abstract a PACT is and the more dimensions it relies on, the more it is "playable" (e.g. "play ascending notes" is less playable than "play C E G", "play bluesy" is less playable than "play a diminished fifth on the second beat", etc.). A *fully playable* (or just *playable*) PACT is defined as a low-level PACT on all three dimensions. The second property is the *combinability* of PACTs, i.e. they can be combined to generate more

given context (e.g. C major) to yield "play C E G". In this sense, PACTs may or may not be compatible. "Play loudly" and "play quietly" cannot be combined whereas "swing", "play major scale" and "play loudly" can.

These properties constitute the basis of our problem solving method (Newell & Simon 1972; Nilsson 1971). Taking an initial state of a problem space as a time segment (e.g. bars) with no notes, a musical problem could consist in filling this time segment with notes which satisfy some criteria. This intuitive formulation of what a musical problem is (Vincinanza & Prietula 1989) has been criticized by many researchers because these criteria are not determined *a priori* (Johnson-Laird 1992). However, we present here a different point of view that allows us to formalize and deal with musical creativity as problem solving. We claim that the musical problem is in fact to know how to start from "vague criteria" and go towards a precise specification of these criteria. In other words, solving a musical problem consists in assembling (combining) a set of PACTs that have been activated by the performance context. The goal is fixed and clearly defined (i.e. the goal is to play!).

### 3.3 - PACTs as the contents of the Musical Memory

There is no guarantee that a set of PACTs contains the necessary information so as to produce a playable PACT. To solve this problem we have introduced the notion of Musical Memory which explores the principles of case-based reasoning (Schank & Riesbeck 1989). This Musical Memory is a long term memory that accumulates the musical material (cases) the musicians have listened to. These cases are represented using PACTs' framework and thus can be retrieved and modified during the problem solving to provide missing information.

The cases are obtained by applying transformations (e.g. time segmentation, projection on one or two dimensions, etc.) to transcriptions of actual Jazz recordings. This process (so far, guided by a human expert) yields cases such as melody fragments, rhythm patterns, amplitude contours, chords, etc. The cases are indexed from various points of view that can have different levels of abstraction such as underlying chords, position within the song, amplitude, rhythmic and melodic features (Ramalho & Ganascia 94a). These indexes are in fact the same attributes used to describe activated PACTs. For instance, pitches are described in terms of contour, tessitura, set and style as discussed in last section.

Low-level PACTs are PACTs whose attributes are all specialized, i.e. have defined values. Describing a Musical Memory case in terms of PACTs correspond to start from note-level attribute to fill in the more abstract ones. Whereas, in the process of assembling PACTs we start from abstract descriptions to combine them into note-level ones. When this latter is not possible, a match between the already specialized attributes and the PACTs in the Musical memory is performed.

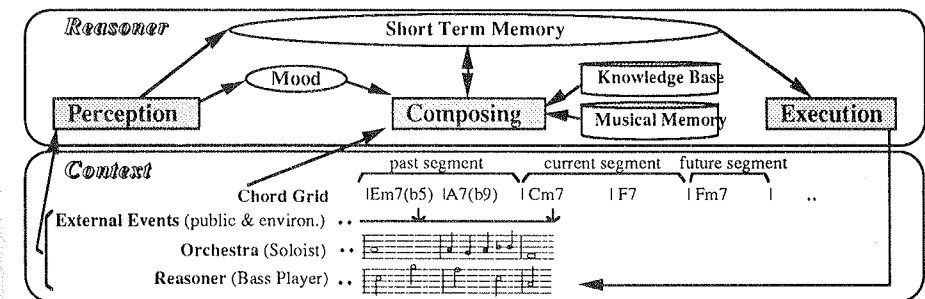


Figure 1 - Overall Description of the Model

## 4 - General Description of our Model

### 4.1 - The Reasoner and the Composing Module

What we do is model a musician as a *reasoner* whose behavior is simulated by three modules which work coordinately in parallel (see Figure 1). The modules of our model resemble the *Monitoring*, *Planning* and *Executing* ones of some robotics applications (Ambros-Ingerson & Steel 1988). The *context* is composed of a *chord grid* which is given at the outset and *events* that occur as the performance goes on, i.e. the notes played by the orchestra and reasoner and also the public reactions. The *perception module* "listens to" the context events and puts them in the *Short-Term Memory*. The *composing module* computes the notes (a playable PACT)

Short-Term Memory contents, the reasoner's mood and the chords of the future chord grid segment. The *reasoner's Mood* changes according to the context events. The *execution module* works on the current chord grid segment by executing the playable PACT previously provided by the composing module. This execution corresponds to the sending of note information at their start time to the perception module and to a MIDI synthesizer, which generates the corresponding sound.

The problem of playing along a given chord grid can be viewed as a continuous succession of three sub-problems: establishing the duration of the new chord grid segment; determining the PACTs associated to this segment; and assembling this group of PACTs in order to generate a unique playable PACT. The first two are more questions of problem setting, the third is a matter of problem solving and planning.

The composition model is supported by a Musical Memory and Knowledge Base. The former contains low-level PACTs that can be retrieved during the PACT assembly. The latter contains production rules and heuristics concerned with the segmentation of the chord grid, changes in the Mood and the selection/activation of PACTs. These rules are also used to detect and solve incompatibilities between PACTs, to combine PACTs and to modify low-level PACTs retrieved from the Musical Memory.

In next sections we give further details of the composition module. The discussion of perception and the execution modules is not in the scope of this paper (see Ramalho & Ganascia 1994b).

#### 4.2 - Segmenting the Chord Grid and Selecting PACTs

The chord grid is segmented in non regular time intervals corresponding to typical chord sequences (II-V cadences, modulations, turnarounds, etc.) abundantly catalogued in Jazz literature (Baudoin 1990). In fact, the reasoning of musicians does not progress note by note but by "chunks" of notes (Sloboda 1985). The criteria for segmenting the chord grid are simple and are the same as those used for segmenting the transcription of Jazz recordings in order to build the Musical Memory.

Given the chord grid segment, the group of associated PACTs derives from three sources. Firstly, PACTs are activated according to the chords of the grid segment (e.g. "if two chords have a long duration and a small interval distance between them then play an ascending arpeggio"). Other PACTs are activated from the last context events (e.g. "if soloist goes in descending direction then follow him"). The activation of a PACT corresponds to the assignment of values to its attributes, i.e. the generation of an instance of the class PACT in an Object-Oriented Language. Finally, the previously activated PACTs whose life time intersects the time interval defined by the segmentation (e.g. "during the improvisation play louder") are added to the group of PACTs obtained from the first two steps.

The reasoner can be seen as an automaton whose state (Mood) changes according to the context events (e.g. "if no applause after solo then Mood is *bluesy*" or "if planning is late with respect to the execution then Mood is *in a hurry*"). So far, the reasoner's Mood is characterized by a simple set of "emotions". In spite of its simplicity, the Mood plays a very important role in the activation and assembling of PACTs. It appears in the left-hand side of some rules for activating PACTs and also has an influence on the heuristics that establish the choice preferences for the PACT assembly operators. For instance, when the reasoner is "in a hurry" some incoming context events may not be considered and the planning phase can be bypassed by the activation of playable PACTs (such as "play this lick") which correspond to the various "default solutions" musicians play.

#### 4.3 - Assembling PACTs

The initial state of the assembly problem space is a group of selected PACTs corresponding to the future chord grid segment. The goal is a playable PACT. A new state can be reached by the application of three operators or operator schemata (since they must previously have been instantiated to be applied): delete, combine and add. The choice of operator follows an opportunistic problem solving strategy which seeks the shortest way to reach the goal. Assembling PACTs is a kind of planning whose *space state* is composed of *potential actions* that are combined both in parallel and sequentially since sometimes they may be seen as constraints and other times as procedures. Furthermore, the actions are not restricted to primary ones since potential actions have different abstract levels. Finally, there is no backtracking in the operator applications.

The *delete operator* is used to solve conflicts between PACTs by eliminating some of them from the group of PACTs that constitute the next state of the space problem. For instance, the first two of the PACTs "play ascending arpeggio", "play in descending direction", "play louder" and "play syncopated" are incompatible. As proposed in SOAR (Laird, Newell & Rosebloom, 1987), heuristics state the preferences for choosing a production rule from a set of fireable rules. In our example, we eliminate the second one because the first one is more playable.

The *combine operator* transforms compatible PACTs into a new one. Sometimes the information contained in the PACTs can be merged immediately to yield a low-level PACT on one or more dimensions (e.g. "play ascending notes" with "play triad notes" yields "play C E G" in a C major context). Other times, the information is only placed side by side in the new PACT waiting for future merger (e.g. "play louder" and "play syncopated" yields, say, "play louder and syncopated"). Combining this with "play ascending arpeggio" generates a playable PACT.

The *add operator* supplies the missing information that is necessary to assemble a playable PACT by retrieving and adapting adequate cases (low-level PACTs on one or more dimensions) from the Musical Memory. The retrieval is done by a partial pattern matching between case indexes, the chords of the chord grid segment and the current activated PACTs. Since the concepts used in the indexation of cases correspond to the descriptors of high-level PACTs, it is possible to retrieve low-level PACTs when only high-level PACTs are activated. For instance, if the PACTs "play bluesy" and "play a lot of notes" are activated in the context of "Bb7-F7" chords, we search for a case that has been indexed as having a bluesy style, a lot of notes and IV7-IV7 as underlying chords. When there is no PACT on a particular dimension, we search for a case that has "default" as a descriptor of this dimension. For instance, it is possible to retrieve a melody even when the activated PACTs concern amplitudes only.

The cases may correspond to some "chunks" of the note dimensions that may not *fit in* the "gaps" that exist in the current activated PACTs. Thus, retrieved cases may carry additional information which can be partially incompatible with the activated PACTs. Here either the conflicting information is ignored or it can "short-circuit" the current PACT assembly and lead to a different playable PACT. Let us suppose that the activated PACTs concern pitches and amplitudes and the retrieved case concerns pitches and rhythm. Only the activated PACTs on amplitude can be considered to be combined with the retrieved case generating a playable PACT. But, if the retrieved case concerns rhythm and amplitudes, perhaps the latter information could be ignored.

Choosing the add operator balances the cost in terms of memory search time with the possibility of short-circuiting the assembly process. Short-circuiting is an important feature of music creativity. For instance, in melody composition there is no chronological ordering between rhythm and pitches (Sloboda 1985). Sometimes both occur together! This feature is often neglected by computational formalisms (Vincinaza & Prietula 1989).

#### 5 - Discussion

We have shown how an extension to classical problem solving could simulate some features of musical creativity. This extension attempts to incorporate both the experience musicians accumulate by practicing and the interference of the context in the musicians' ongoing reasoning. Although we do not use randomness in our model, there is no predetermined path to generate music. The musical result is constructed gradually by the interaction between the PACTs activated by the context and the Musical Memory's resources.

The notion of PACTs was first implemented (Pachet 1990) for the problem of generating live bass line and piano voicing. At this time, results were encouraging but, exploring exclusively a rule-based approach, various configurations of PACTs were hardly treated, if at all. This was due to the difficulty of expressing all musical choices in terms of rules. Our work has concentrated on improving the formalization of PACTs within a problem solving perspective. We have also introduced the notion of Musical Memory and seen how it can be coupled with PACTs. Today, Pachet's system is being reconsidered and re-implemented using a Smalltalk platform to take into account both the Musical Memory and a wider repertoire of PACTs.

#### Acknowledgments

I would like to thank Vincent Corruble, Jean-Gabriel Ganascia, François Pachet and Jean-Daniel Zucker, from LAFORIA team, for the continuous encouragement and technical support. This work has been partly supported by a grant from the Brazilian Education Ministry - CAPES/MEC.

#### References

- Ambros-Ingerson, J. & Steel, S. (1988). Integrating Planning, Execution and Monitoring, In Proceedings of the Sixth National Conference on Artificial Intelligence, 83-88, AAAI Press.
- Ames, C. & Domino, M. (1992). Cybernetic Composer: an overview, In M. Balaban, Ebicoglu K. & Laske, O. eds., *Understanding Music with AI: Perspectives on Music Cognition*, The AAAI Press, California.
- Baker, M. (1980). Miles Davis Trumpet, Giants of Jazz Series, Studio 224 Ed., Lebanon.



- Baudoin, P. (1990). *Jazz: mode d'emploi*, Vol. I and II. Editions Outre Méasure, Paris.
- Johnson-Laird, P. (1992). *The Computer and the Mind*. Fontana, London.
- Laird, J., Newell, A. & Rosenbloom, P. (1987). SOAR: An Architecture of General Intelligence, *Artificial Intelligence* 33, 1-64.
- Newell, A. & Simon, H. (1972). *Human Problem-Solving*, Englewood Cliffs. Prentice Hall, NJ.
- Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill Book Co., New York.
- Pachet, F. 1990. Representing Knowledge Used by Jazz Musicians, In the Proceedings of the International Computer Music Conference, 285-288, Montreal.
- Ramalho, G & Ganascia, J.-G. (1994a). The Role of Musical Memory in Creativity and Learning: a Study of Jazz Performance, In M. Smith, Smaill A. & Wiggins G. eds., *Music Education: an Artificial Intelligence Perspective*, Springer-Verlag, London.
- Ramalho, G & Ganascia, J.-G. (1994b). Simulating Creativity in Jazz Performance, à paraître dans Proceedings of the Twelfth National Conference in Artificial Intelligence - AAAI '94, Seattle, AAAI Press.
- Ramalho, G. & Pachet, F. (1994). What is Needed to Bridge the Gap Between Real Book and Real Jazz Performance?, in the Proceedings of the Fourth International Conference on Music Perception and Cognition, Liège.
- Rowe, J. & Partridge, D. (1993). Creativity: a survey of AI approaches, *Artificial Intelligence Review* 7, 43-70, Kluwer Academic Pub.
- Schank, R. & Riesbeck, C. (1989). *Inside Case-based Reasoning*, Lawrence Erlbaum Assoc. Pub., New Jersey.
- Sloboda, J., (1985). *The Musical Mind: The Cognitive Psychology of Music*, Oxford University Press, New York.
- Steedman, M. (1984). A Generative Grammar for Jazz Chord Sequences, *Music Perception*, Vol. 1, No. 2, University of California Press.
- Vincinanza, S. & Prietula, M. (1989). A Computational Model of Musical Creativity, In Proceedings of the Second Workshop on Artificial Intelligence and Music, 21-25, IJCAI, Detroit.

## Desempenho, Interface com o Usuário e Projeto de Instrumentos

## A Phenomenological Study of Timbral Extension in Interactive Performance

ANNA SOFIE CHRISTIANSEN, UNIVERSITY OF COPENHAGEN, DEPARTMENT OF MUSIC

*Klerkegade 2, 1308 Copenhagen K, Denmark,*

### Abstract

This paper features an investigation of the interactive electronic extensions of the musical performance, based on a phenomenological approach to the human perception of sound in a musical context. The interactive extension of the musical performance offers, through performer control over the produced sound in real-time, possibilities for the composer to take advantage of subtle features of acoustic sound, due to the individuality of the live performance. Richard Leppert understands musical activity as a synthesis of sound experience in accordance with a visual experience: the sonoric landscape. Using this concept investigations about perceptive characteristics of synthetic and human sound will be made, with a purpose of describing the advantages of real-time sound processing. Further, the structure of the interactive process will be sketched, and in the final section a brief overview of a piece, *NoaNoa*, for flute and interactive electronics by Kaija Saariaho will be given.

### Introduction

In this paper I will investigate the phenomenologic difference between synthetic and human-produced sound, in order to explore the options for an interactive extension of the musical performance featuring real-time sound processing. This leads to a thorough description of the perceptive characteristics of the human produced sound in the musical performance, where composer, performer and listener have a certain common knowledge of the material, based on an empirically experienced musical idiom. By empirically experienced musical idiom is here understood music created in a tradition where there is a large degree of consensus in the experience of the sonoric landscape in composer, performer and listener. Utilisation of synthetic sounds interferes with this consensus by attacking the experienced expectations of accordance between cause (performance gesture) and effect (sound produced).

In the approach to composition of computer music, the role of the composer may therefore also encompass exploration of the field created between boundaries of the scientific representation of sound, to be manipulated by the composer, human perception, tradition and the cognition of sound, with an awareness of the fluid limits of human cognition within a musical context, but also within the boundaries of technology's still rather weak ability to accommodate human musical behavior. To serve this elucidation advantages of the interactive extension to the musical performance will be given. These will be presented in terms of the option for real-time sound processing to encompass a musical material embracing all perceptually significant details of the human produced sound. Providing these options, the interactive extension will offer a flexible expansion of the musical material, already anticipated in the traditional electronic extension.

### Perceptive Timbral Connotations to Natural and Synthetic Sound

#### Correlation of Performer and Synthetic Sound

The exploration of the computer as a compositional tool and musical instrument have involved an investigation of features important for the human perception of sound. In the following, some traits will be set out concerning the perception of music. This will serve to expose the central differences between an interactively extended performance, where electronic sounds are triggered by the performing musician's physical and acoustic gestures, as compared to a non-interactive but electronically enriched performance, such as with a sequenced accompaniment where whole sequences of sounds, of relatively fixed temporal and dynamic relationships, are merely started by a trigger, or the case of a tape accompaniment which must be synchronized by click-track or pre-defined cues for the performer to follow.

Initially three issues can be set up:

- The performer is accompanied by an independent sound track, synchronization is provided e.g. by cues or click-track.
- The performer is "tracked" and used to synchronize a synthetic playback.
- The performer's sound is processed directly on the synthesis of sound.

The interactive performance takes often advantage of the two last issues; using the tracking of the performer to synchronize direct processing of the acoustic sound. Thus will the performer be tracked either by, e.g., a pitch-tracker or a MIDI interface. The tracking will be compared to a representation of the score in the computer, where cues for the execution of the respective sound processings are marked.

#### Outlining of Differences between Synthetic and Acoustic Sound

This approach requires an outlining of characteristics of the synthetic and the acoustic (human-produced) sound. This difference may serve to illustrate two main differences between perception of electronically and human produced sound, because the interactive (and sonic processed) technique permits a synthesis of the advantages of the musical performance within its frame of reference to tradition and the sonorous extensions provided by electronic means. The human produced sound involves automatically information referring to a musical convention, whether this information is contained in the musical texture itself, or in the performer's gestures as will be explained in the following. The electronic sound, on the other hand, does not necessarily refer to a musical "performance" convention itself, but seems merely to form a perceptual significance by virtue of its context; and this is conditioned on expectations in relation to the traditional musical performance.

The synthetic musical texture itself thus differs significantly from the human controlled sound in many ways, e.g., by the difficulties in obtaining control for the performer of sonoric gestures as vibrato, transition phenomena, attack, timbre and more subtle irregularities in phrasing, creating the uniqueness of the human musical performance.

In music, a distinction is therefore often made between synthetic sound and acoustic-instrumental sound solely from the sound image, without taking into account connotations of sounds in relation to conventions in a musical tradition. The most salient difference is that the behavior of the acoustic-instrumental sound can be changed in real-time by the performer according to musical tradition. The behavior of the non-interactive synthetic sound is pre-determined, i.e., it does not mirror the individuality of the performer and further does not refer implicitly to a tradition, but nevertheless often is perceived in accordance with a musical tradition.

#### The Significance of Gestures

The following conventions are considered to have significance for musical perception: The presence of a human performer affects the sound production in a musical performance in two ways, both of which are connected to conventional musical performance means; these two effects I will refer to as sonorous and physical gestures. The sonorous gesture can be exemplified by connotations of the perceived sound itself, required by the listener to make sense in the context of a particular musical idiom. Specifically in connection to the perception of timbre, it can be exemplified as follows: when a flute plays a piano dynamic, not only a change in loudness is of importance; significant perceptive cues are provided by changes in vibrato, timbre and attack, all this in relation to the specific context, required to give the listener the right feeling of piano. I will not judge to which extent this is due to a combination of traditional habituation that has turned into a convention and an innate result of the human hearing sense.

The physical gesture, on the other hand, is more immediately explained as actions used more or less consciously by the performer to accentuate the musical phrasing; most obvious is the use of the breath to underline phrasing, or the various body movements such as a nod of the head or facial expressions serving to mark entrances, downbeats or sudden shifts in the musical ambience, like utilization of contrasting material engendering a sudden shift in the musical ambience, used more or less consciously to underline the musical texture. That is: the visual presence of a performer may have significance for the perception of sound. These close relations between perception of music and bodily movements are described as a cognitive phenomenon by Ray Jackendoff in *Consciousness and the Computational Mind*.<sup>1</sup>

<sup>1</sup>Jackendoff (1987) accentuates in relation to perception of music, the association of bodily motion in perception of rhythmical patterns. I consider that this association has also a significance in the perception of rhythmical irregularities or breathings used e.g. in phrasing.

#### A Phenomenological Approach

A more phenomenological approach is found in the work of Richard Leppert<sup>1</sup>. Leppert defines the concept of a sonoric *landscape* as the reconstitution in our minds of something in excess of the factual: "This excess is experienced as a representation - and as such is discursive."<sup>2</sup> By this Leppert wishes to emphasize the perception of music, not only from the perception of the sound, but as the impression of sound from the perception of sound and sight perceived by means of human experience and consciousness: "Music connects to the *visible* human body, not only as the receiver of sound but also as its agent or producer."<sup>3</sup> By using Leppert's descriptions we can understand musical activity as a synthesis of sound experience in accordance with a visual experience<sup>4</sup>. This visual experience will be in accordance with a musical performance idiom where there will be a habitual relation between visual impression and sound forming the sonoric landscape. I will here add, that the nature of the visual impression could rather manifest itself as a bodily connotation to the perceived sound in the listener, not depending on the visual presence of a performer, but merely a cognitive aspect in the nature of music which Jackendoff names general-purpose abilities<sup>5</sup> where the sensation of certain traits of musical sound, such as rhythm and rubato, induce associations to bodily motions.

Connotations of the aural perception may therefore be influenced by connections to the assigned visual perception, that is to say, a performer's movements and the perceptor's expectations of the sound in connection with a musical idiom, may have significance for the comprehension for the interpretation of the musical performance to the listener: the creation of a sonoric landscape. These relative perceptions I will define as casually interdependent, when the perceptor's expectations of the sound depend on the degree of conformance between the visual impression, the assigned emotional impression and the sound perceived.

#### The Importance of Causality for the Perception of Sound

The notion of causality<sup>6</sup> is important in connection with synthetic sounds, when the perceiver more or less consciously tries to identify the sound source. Causality should then be understood as a listener's immediate unconscious image of the sound producing *instrumental-mechanism*, not literally understood, but rather whether the *instrumental-mechanism* refers to traditional categories, e.g., voice, wind, string or percussion instruments, etc., or, rather as not referring to any instrument category, e.g., some kinds of noise, or, sound without the characteristics of the human performer such as subtle irregularities in vibrato, transitional changes, etc. Identification with known sound sources, might thus help the listener to place the sound picture in relation to a known sonoric landscape. In extension of the human performance with synthetic sound, the relation between sound picture and the visual impression might be segregated compared to the perception of a traditional musical performance, and this segregation could then be used by the composer to extend the sonoric material in the musical performance.

The distinction between sonorous and physical gestures can be used as a tool to elucidate idiomatic properties of the acoustic-instrumental sound in relation to the synthetic sound, with a purpose being to evaluate real-time sound synthesis as a qualitative phenomenon in relation to the musical work.

#### **Interaction between Computer and Performer**

##### The Sonoric Extension of Interactive Sound Processing

The interactive extension of the musical performance is a very important technique for taking advantage of electronic means within the context of traditional musical performance. The musical texture is extended in conjunction with traditional musical performative means, thus allowing the performer to directly control the synthesis of sound or the execution of synthesized material under traditional performance circumstances. In this paper I will not consider current implementation problems, but I will define and discuss the principle of interaction as a concept.

Interactive extensions to musical performance can enrich the musical texture in different ways according to the kind of processing the musical material is subjected to. These processes could be divided in accordance with the origin of the source material to be processed, and the nature of the processing itself as sonic and

<sup>1</sup> Leppert 1993, p. 17.

<sup>2</sup> Ibid. p.17-18.

<sup>3</sup> Ibid. p. 18.

<sup>4</sup> I do here consider any musical experience to give rise to notions about a visual experience in accordance with previous musical experiences including visual impressions.

<sup>5</sup> Jackendoff 1987, p. 216.

<sup>6</sup> This concept was introduced by Chabot in Leonardo, vol 3 1993.

gestural processing. In the sonic processing, the instrumental sound itself is processed and transformed by means such as reverberation, filtering, pitch-shifting or modulation. This affects mainly the timbral attributes of the sound. In the gestural processing, sonorous gestures such as pitch and dynamic (as opposed to timbre) are "abstracted" from the performance and subjected to higher-level processing, e.g., as sequences played back with a different distribution in time, or diffused in space: spatialisation. These played back sequences can also be sonically processed, or the instrument could simply trigger stored sequences in the computer, which can be considered as the ideal case of synchronization between performer and a pre-recorded sound track. The nature of the higher-level processes could be more or less in accordance with traditional musical gestures.

I have not here considered audification of the performer's physical gestures in this connection, since they do not provide any sonorous source material to be processed. The assignment of sonoric material to the performer's gestures, offers a rich spectrum of options for the composer to interpret the perception of the musical performance, thus allowing the composer to consciously create a sonoric landscape and not only a sound image.

#### The Structure of the Interactive System

An important aspect of interaction, compared to the traditional performance accompanied by a stored sound track, is the freedom provided by the synchronization mechanism between the interactive system and the performer. The synchronization is realized by a coordinator concept essential for interactive systems. An overview of an interactive system is shown in Figure 1:

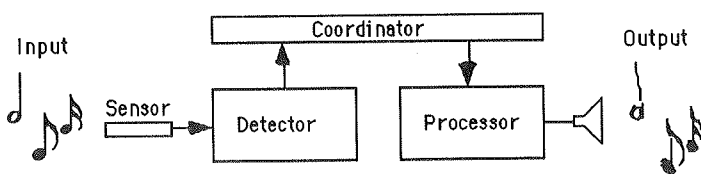


Figure 1

As can be seen on the figure, there are four principle parts of an interactive system: a *sensing* system that gleans information from the sound source, e.g., with a microphone; a *detecting* system, detecting pitch or trigger information, a *coordinator*<sup>1</sup> providing synchronization of the input and its respective events; and a last stage executing the sound *processing*.

#### Advantages offered by Real-time Sound Processing

The presence of this coordinating mechanism, effected by the score follower<sup>2</sup>, provides a freedom for the performer to take advantage of traditional performance means, such as rubato, individual phrasing, etc., as well as timbral extensions provided by the sound processing. Real-time synthesis, as opposed to tape music, for example, provides possibilities for using a technological approach while permitting composer and performer to take advantage of traditional means as sonoric and physical gestures. Another important point is the possibility for the electronic extension of the musical performance to move, or even reduce, the border between non-idiomatic and idiomatic musical gestures, an elimination to be provided through renewed possibilities for changing connotations to, or causality of, sound, as explained in the previous section.

An approach to understanding these options is offered partly by the timbral exploration. Timbre contains in itself fairly limitless extensions of new sound textures, amalgamations of different timbral idioms, causality of sound as an artistic effect: taking advantage of the options for manipulation of the acoustic properties of the instrument, or capturing the performer's gestures to process a sound in order to emphasize the

<sup>1</sup> The significance of the coordinator varies in this connection from the more specific notion of a score follower requiring a representation of the score in the coordinator e.g. as it is defined by Rowe 1993. It signifies here a mechanism maintaining a synchronization of particular detected events or conditions in relation to more or less predetermined responses executed by the computer.

<sup>2</sup> See Lippe and Puckette 1992.

significance of physical gestures. The last issue requires an exploration of the significance of bodily motion in a musical performance. These advantages can be best treated by considering the experience of sound phenomenologically, as shown in the previous section. The conceptual flexibility of interaction thus provides a possibility to let the work allude to something external as, for instance, the musical performance as a perceived phenomenon: the sonorous landscape, within the musical work's own means of expression: the timbral texture.<sup>1</sup>

#### NoaNoa by Kaija Saariaho

##### A Brief Introduction to the Interaction in NoaNoa

In the following section I will examine the processing provided by interaction in the piece NoaNoa by the Finnish composer Kaija Saariaho. The piece was originally conceived for solo flute and electronics in a Macintosh version. There, the interaction was provided by the performer's release of certain events by a pedal. In a new version the real-time processing of the acoustic sound is adapted for the IRCAM Signal Processing Workstation (ISPW). NoaNoa provides an illustrative example to evaluate the contributions to the acoustic sound image provided by the interactive extension of the sound. The schematic structure of the workstation is similar to the construction shown in figure 1.

##### The Interactive System in NoaNoa

The sensing is performed by two microphones; one right in front of the blowhole to sense the words pronounced by the performer, and another one "beside" the flute to capture the flute's sounds. The signals from the microphones is scrutinized by the pitch-tracker, so as to be used by the score-follower to synchronize the performer with the processing employed in relation to given cues in the internal representation of the score in the computer<sup>2</sup>, the other is connected to a pitch-tracker. The selection of which microphone to take the input from, is pre-determined in the internal score. The sound processing, score following and pitch detection are designed in MAX. The sound processing, in the recent version, such as filtering, play back of stored sequences and real-time sampling is executed by the ISPW, the reverberation by a Lexicon sound processor, and the superposed envelopes from crotales are made through a Reson8.

##### Contributions to the Musical Texture by Means of the Interaction

In NoaNoa the use of electronics helps to fulfill some of the composer's musical intentions, which would not be possible without an electronic extension. The primary intention is concerned with conveying a continuous character to the monophonic musical line. This is obtained by adding reverberation to the flute sound: "the quieter the sound, the longer the reverberation."<sup>3</sup> The breathing is also smoothed by reverberation, but retains never the less a character of breath determined phrasing while still maintaining a continuous musical texture. The timbre is extended using special techniques, for instance, is the pitched nature of the flute contrasted by the whispered (noisy) "second voice" made by the performer, sometimes played back from stored segments in the ISPW, and sometimes sampled from the performer and played back. Further the flute timbre is contrasted by using special effects: whisper tones, flutter-tongue, or fingered tones solely attacked by the consonant sounds of the spoken voice of the performer, a technique resulting in a percussive effect. The monophonic line of the flute part is fractured using multiphonics and whispering into the instrument while playing, but additional timbral layers are provided by the sound processing. The composer thus approaches the nature of the acoustic sound image to encompass an electronic sound image containing sounds that are not within the flute's normal spectrum.

It remains to discuss what is conveyed to the piece by the use of interactive means. The first attribute is a more homogenous sonic extension, perfectly synchronized with the performer part, permitting the performer an extended timbral and rhythmic control over the course of events, thus providing freedom in the shaping of musical phrases. This results in a synchronization between performer and a synthetic sound image provided by a virtual performer, but crystallized over the soloists individual sound. The electronics further extend the polyphonic character, anticipated in the instrumental voice itself, by implementation of several instrumental techniques at the same time, resulting in a very complex sonoric texture, enriching the flute sound.

<sup>1</sup> I do not in this include *musique concrète*, when it is merely constituted as montage referring to the surrounding world by virtue of collocations of fragments originating in the surrounding world.

<sup>2</sup> The pitch-tracker does not respond to the unpitched sound of the spoken voice, because of that, a pedal is used to trigger during voiced events.

<sup>3</sup> Preface in the Chester Music edition 1992.

### Conclusion

Thus having outlined some important traits of the human experience of electronic and acoustic sound in a musical context, I have ignored an evaluation of any technical means to provide the discussed extensions. Many problems seem to be connected to this field. A major problem is, of course the costs of equipment and soft ware, and following; the problems of performing pieces made for other equipment and requiring a knowledge about the actual equipment and the systems used in the piece.

Another important feature concerning the representation of sound in the computer, is the difference between the phenomenological and the physical description of the same sound caused by the different approaches to description of sound: the phenomenological description, based on empirical musical experience often taking into account the origin of the sound; and the numerical description, based on scientific terms not immediately connected to the empirical experienced sound world, by opening possibilities for transcending the empirical experienced production of acoustic sound.

A main problem is that the creation of tools to implement computers in the creation of music is a hybrid area claiming knowledge of the scientific description of sound, computers, composition and human perception of sound in order to develop adequate tools for the artist. This evolution is far from fulfilled, but will probably get easier as interfaces between man and computer will make the approach towards control of sound, simpler and more immediate. Neural networks open up possibilities, since they can be trained to approximate algorithms representing the details of human skills, and give the artist access to control permitting him to use more traditional artificial means; skills too complex and subtle to be represented as algorithms in traditional computer systems.

It seems several of the above mentioned approaches still belong to the future, requiring more reliable systems, and disseminate of equipment and knowledge to put the interactive performance on equal footing with standard performance institutions as symphonic orchestras and chamber ensembles, it is necessary to provide a sufficient performances number of for the composer to dare to use time and energy exploring the unknown.

### References

- Chabot, Xavier: *To listen and to see: Making and Using Electronic Instruments*, Leonardo Music Journal, vol. 3 1993, pp.11-16.  
 Jackendoff, Ray: *Consciousness and the Computational Mind*, MIT Press 1987.  
 Leppert, Richard: *The Sight of Sound, Music, Representation, and the History of the Body*, University of California Press 1993.  
 Puckette, Miller and Lippe, Cort: "Score following in practice" in *Proceedings of the International Computer Music Conference*, San Jose 1992. International Computer Music Association, San Francisco.  
 Rowe, Robert: *Interactive Music Systems: machine listening and composing*, MIT Press, 1993.  
 Saariaho, Kaija: *NoaNoa* for flute and electronics, Chester Music 1992.

With special thanks to Xavier Chabot and Kaija Saariaho, Institut de Recherche et Coordination Musicale, Guy Garnett, Center for New Music and Audio Technologies, UC Berkeley and Jens Brincker, University of Copenhagen.

## UM NOVO MÚSICO CHAMADO 'USUÁRIO'

FERNANDO IAZZETTA

Pontifícia Universidade Católica de São Paulo  
 Comunicação e Semiótica / Laboratório de Linguagens Sonoras  
 R. Monte Alegre, 984 - Perdizes - São Paulo-SP - CEP 05014

### Resumo:

Nossa tradição ocidental estabeleceu uma separação aguda entre os papéis desempenhados pelos agentes da atividade musical: o compositor, o intérprete e o ouvinte operam em campos muito bem delimitados e específicos. As possibilidades de interação trazidas pelo uso do computador em música, de certa forma diluem estas separações e fazem surgir uma nova categoria, capaz incorporar habilidades específicas de cada um destes agentes tradicionais. Esta nova categoria estaria ligada ao que chamamos correntemente de "usuário".

### Introdução

"Estou plenamente certo de que chegará o dia em que o compositor, após realizar graficamente sua partitura, a terá automaticamente colocada em uma máquina que transmitirá fielmente o conteúdo musical ao ouvinte" [Varèse:1983, 92].

Quando o compositor Edgar Varèse fez esta previsão, o computador, do modo como o concebemos hoje, não passava de uma projeção num futuro incerto. Varèse encontrava-se, já no início deste século, profundamente interessado na criação de novos modos realização musical e de novas posturas de escuta e profetizava a emergência de novas maneiras de produção sonora que, até então, só podiam ocorrer na imaginação do compositor.

Poucas décadas depois, as previsões de Varèse foram se tornando realidade e não podemos deixar de nos sentir curiosos por saber quais seriam as soluções musicais concebidas pelo compositor de *Ionisation* e *Poème Électronique* se tivesse à sua disposição todo o arsenal tecnológico que pode ser empregado na música feita nos dias de hoje.

A história que se passa entre o surgimento dos primeiros instrumentos elétricos do início do século, como o *ondes de martenot* e o *theremin*, e a criação das atuais interfaces sonoras inteligentes controladas por computadores, é uma história repleta de idéias que apostaram numa transformação do nosso universo musical de um modo denso e profundo. Se observamos hoje, em diversas esferas da produção musical, um movimento de reflexão sobre estas transformações, isso não se deve ao fato de que o músico passou a ter à sua disposição uma gama de timbres maior do que se poderia obter através de instrumentos tradicionais, nem tão pouco à capacidade dos computadores para executar passagens musicais impossíveis para qualquer instrumentista virtuose, mas sim porque a própria música passou a ocorrer dentro de um contexto completamente novo.

### Aquele que faz e aquele que ouve música

A música ocidental, dentro de sua tradição, estabeleceu uma separação explícita entre os agentes musicais, a qual recentemente vem sendo colocada em questão. Diferente do que ocorre em outras culturas onde a música é uma manifestação coletiva, o Ocidente foi estabelecendo, aos poucos, limites estritos entre aqueles que criam, aqueles que executam e aqueles que ouvem música.

O início deste processo pode ser identificado no final da Idade Média quando o cantochão passa a ser a base para a criação de novas formas dentro da música profana, ao mesmo tempo que reluta em ser influenciado por estas. É neste período que a música vai se desligar de sua função estritamente ritual para assumir um novo papel na cultura, muito mais voltado ao lúdico e ao estético. Impõe-se ao músico uma necessidade de criar novas fórmulas e padrões dentro da linguagem e, com isso, surge o desejo de compor. A música se afasta da tradição do canto gregoriano, tornando-se cada vez mais rica e complexa, exigindo, também, intérpretes dedicados e ágeis em seus instrumentos.

Inaugura-se aí o grande processo delineador das figuras do compositor, do instrumentista ou cantor, e do ouvinte como os três elos centrais da produção musical. Este quadro já é explícito no período Renascentista e,

nas épocas seguintes, a música seguirá, sem desvios, este projeto de especialização. Compositor, interprete e ouvinte têm que desenvolver ao máximo suas capacidades específicas para que nada escape no complexo discurso dos sons. Tal processo que ocorre durante a passagem da Idade Média para a época Moderna reflete uma crise dentro da linguagem que carrega muitas analogias com a situação atual, pois, da mesma maneira vivemos hoje mudanças profundas em todas as esferas culturais, incluindo aí a música e as artes em geral [ver Iazzetta:1993].

Uma reflexão a respeito da linguagem musical nos dias de hoje passa, obrigatoriamente, pela questão da introdução do uso de computadores nos diversos estágios da produção musical. Da mesma maneira que os processos de escrita musical surgidos entre os séculos IX e XIII foram essenciais para a formação da música clássica europeia, hoje o computador vem alterando sensivelmente o desenvolvimento dessa linguagem.

Nos últimos anos, a informática tem penetrado rapidamente em todos os tipos de atividades humanas e, aos poucos, sua utilização vem se incorporando totalmente ao nosso cotidiano, ao ponto que sua presença nos passa naturalmente despercebida, como ocorre em relação à eletricidade ou ao uso do automóvel.

O fato de se encarar uma tecnologia poderosa como é a do computador como um sistema que é gerado, não apenas para suprir algumas de nossas necessidades, mas que também exerce influência no surgimento destas necessidades, traz uma idéia complementar àquela de McLuhan de que o homem cria ferramentas para servirem com extensões de suas habilidades naturais (*homo extendere*). Neste estágio atual, onde já podemos trabalhar com máquinas digitais através de interfaces mais ou menos satisfatórias, o computador deixa de ser apenas uma extensão humana para se tornar um novo instrumento que poder agir e reagir às nossas próprias ações. Não se trata de defender aqui a idéia utópica de uma máquina autônoma, capaz de se desenvolver e aprender por si própria. A questão é olhar o computador como uma possibilidade auxiliar, que vai além da execução infinitamente rápida de cálculos complicados e ações precisas. A questão é a de conseguir criar um tipo de relação onde o computador seja utilizado realmente como uma máquina interativa e capaz de gerar possibilidades realmente novas.

## Música Digital

Muito embora a utilização de computadores em música tenha se iniciado a quase 40 anos e o próprio computador seja um pouco mais antigo que isso, o pleno estabelecimento do que hoje conhecemos por *computer music* ainda é um projeto em andamento. Há 30 anos atrás, a música digital consistia em uma área experimental dentro de um campo mais vasto conhecido como música eletrônica. Naquela época, eram poucos os compositores que podiam se envolver num trabalho multidisciplinar que englobava desde o desenvolvimento de *hardware* até a concepção de algoritmos para síntese ou composição musical. Durante todo este período o computador foi utilizado em situações que simulavam os processos tradicionais da produção musical. Os programas de síntese sonora gerados a partir do MUSIC IV desenvolvido por Max V. Mathews, a composição algorítmica inaugurada com a *Illiac Suite* de Hiller e Isaacson e o posterior lançamento comercial de instrumentos digitais foram projetos cujas concepções partiram de práticas correntes na composição e execução da música tradicional. Ou seja, eram novas ferramentas que se destinavam a produzir sons semelhantes aos gerados pelos instrumentos tradicionais e peças musicais que seguiam os mesmos princípios básicos da música europeia.

Parece que somente agora o computador começa a apontar para direções realmente originais na realização musical. O surgimento do computador pessoal, o barateamento dos equipamentos e a rápida evolução na capacidade de processamento e armazenamento de dados, estão fazendo com que o computador deixe de ser apenas uma ferramenta auxiliar no processo de composição e produção para trazer um novo vigor à música contemporânea. Se num primeiro momento essa trama digital atraiu muitos compositores mais pelo interesse por novos processos de composição e pelos resultados sonoros que podiam ser obtidos, hoje é inevitável que as ferramentas digitais são responsáveis pela introdução de processos realmente novos na música.

Certamente, inicia-se uma nova fase dentro da história da música que vem se desviando da tendência à individualização e racionalização que começou a se estabelecer a partir do Renascimento. O que se nota agora, através da ampliação de possibilidades trazidas pelo computador, é uma reordenação nos processos de composição, execução e audição. No lugar do projeto racionalista inaugurado por Descartes, surge uma nova heurística bastante poderosa baseada no *conhecimento por simulação*. O computador permite que se construam modelos que podem ser constantemente testados, redirecionados, e realimentados a partir de resultados obtidos previamente. A obra individual e acabada cede lugar à obra em constante evolução, que retroativamente vai gerando suas próprias soluções num jogo interativo de tentativas e erros. A obra perde sua aura no sentido de benjaminiano [Benjamin:1985] e torna-se obra virtual.

A possibilidade de se transformar música em informação digital tem modificado marcadamente nossa relação com o universo de signos sonoros. Se as gravações em fitas eletromagnéticas ou em discos de vinil

ampliaram espantosamente a portabilidade do repertório musical, a gravação digital praticamente destrói todas as barreiras de distribuição e manipulação do material sonoro. O planeta está rapidamente se tornando uma imensa rede digitalmente interligada e o fluxo de informação que transita nessa rede é algo que seria impensável há alguns poucos anos.

Com isso, a idéia de que a sociedade atual é a sociedade da cultura da massa, onde cada produção signica busca atingir o maior número de pessoas, começa a se modificar neste final de século. A revolução digital aponta para uma inversão neste processo: estamos entrando na era da cultura personalizada, onde cada indivíduo será obrigado a navegar dentro de uma quantidade enorme de informações e selecionar aquilo que realmente lhe interessa. Ou seja, ao mesmo tempo que há uma explosão na produção signica, ocorre também uma segmentação desta produção. As revistas interativas feitas a partir de recursos multimídia que vêm substituindo o papel por suportes digitais ou as redes de BBSs (*Bulletin Board Systems*) que hoje interligam pessoas no mundo inteiro e oferecem os mais variados serviços via computador são alguns exemplos da força deste novo paradigma que vai se formando.

## Interatividade

Talvez o conceito mais importante dentro do universo que se abre com a utilização do computador na música seja o de interatividade. A possibilidade de se intervir nos processos musicais, seja na composição, seja na execução, destrói as fronteiras entre o compositor, o interprete e o ouvinte: os papéis de cada um deles aparecem sobrepostos a partir do momento em que lhes é dada a possibilidade de interagir nas diversas fases da produção musical, da composição à audição. A própria fronteira entre o compor e o ouvir tende a se diluir à medida em que qualquer um tem à sua disposição recursos de *software* e *hardware* que podem ser utilizados na realização de uma peça musical. Nossa idéia de compositor é

"uma concepção específica e peculiar europeia: a inteligência solitária do compositor cria a música e a música é uma imagem do pensamento do compositor. A fuga do condicionado, do específico e do provisório é algo complementar a essa concepção de música[...] A tecnologia musical é hoje largamente dedicada à promover este projeto de fuga, fazendo-nos acreditar, cada vez mais, que a música está em algum objeto sonoro abstrato, cuja imagem nós polimos no computador com ferramentas cada vez mais refinadas" [Perkis:1987, 365].

Hoje em dia contamos com sistemas que permitem a qualquer um "fazer" música sem a necessidade de se passar por anos de aprendizagem musical. O que poderia ser, a primeira vista, entendido como uma banalização dessa que, desde o romantismo, serve como modelo de criação para as outras artes, pode significar a ampliação de seus horizontes, hoje restritos a iniciados aos quais chamamos *músicos*. O leigo é mais despojado da pesada tradição musical do Ocidente que o músico e, por isso mesmo, pode ser um elemento valioso na produção de novas formas musicais. Mas, para que isso ocorra é necessário que 1) o músico abdique da "posse" exclusiva do conhecimento musical e passe a acreditar em maneiras alternativas de se pensar a música e que 2) se desenvolvam novas interfaces, não mais criadas a partir das estruturas de composição e notação da música tradicional, mas segundo as novas necessidades que se impõem à música contemporânea.

A questão da geração de novas interfaces é de extrema importância para o desenvolvimento da música pois estamos acostumados a pensá-la segundo um padrão de sonoridade que pouco se modificou nos últimos séculos e a representá-la através de um sistema notacional que existe desde o Renascimento. A idéia da música de concerto executada por uma orquestra que, em sua base, é a mesma que foi utilizada por Mozart ou Beethoven, perdura ainda em boa parte das manifestações da música atual. A transição de um sistema onde colcheias e semínimas são executadas por um violino ou oboé, para um sistema de valores digitais que correspondem a sons e que podem ser controlados por um *mouse* através de alterações gráficas na tela do computador, representa a transição de um paradigma musical para outro e deve ser tratada com seriedade.

Os botões e *sliders* dos primeiros sintetizadores operavam de forma bastante intuitiva sobre diferentes parâmetros sonoros. Seu movimento guardava uma semelhança mecânica de extensão e direcionamento com as modificações sonoras que eram produzidas na saída de um alto-falante. Ou seja, mantinham o mesmo tipo de relação gestual que ocorre entre os movimentos dos dedos de um pianista e o som que produz em seu instrumento. Hoje, porém, o computador permite que se criem interfaces onde não existe nenhuma relação pré-estabelecida entre a nossa intervenção e o resultado sonoro: um gesto brusco pode gerar um som suave e a mudança de um pequeno detalhe pode acarretar uma transformação global dentro de uma peça.

Esta idéia que aponta para um novo "solfejo" musical não é nova. Pierre Schaeffer, já no final dos anos 40, dava passos importantes nessa direção com seu solfejo de objetos musicais [Schaeffer:1966] e, nos anos 60, Iannis Xenakis desenvolvia o UPIC, sistema onde o usuário desenhava em uma interface gráfica formas que seriam posteriormente convertidas em som. Mas, ainda hoje não dispomos de interfaces satisfatoriamente capazes de tornar a produção sonora algo realmente virtual, de maneira que não esteja mais presa à relação estrita entre

gesto e resultado sonoro. Certamente com muitas dificuldades técnicas e tecnológicas a vencer, o campo que hoje se chama realidade virtual poderá trazer, num futuro bastante próximo, possibilidades musicais extremamente ricas.

Em contraponto com a música coletiva e ritual realizada na Idade Média e com a música de massa do mundo moderno, estas novas interfaces apontam para a personalização e segmentação da produção musical cada vez maiores. A multimídia, as TVs a cabo, a utilização de fibras óticas na transferência de dados, a conexão em rede de computadores pessoais a centros de pesquisa no mundo todo, tornam o usuário incapaz de consumir a quantidade enorme de informação que encontra à sua disposição. Essa proliferação significa exige uma produção cultural específica, confiável e qualificada de acordo com as necessidades de cada indivíduo.

Se por um lado o produto cultural se torna cada vez mais segmentado e personalizado, por outro seus processos de geração, a medida em que se distanciam de modelos artesanais e mecânicos, tornam-se processos multidisciplinares. Nos anos 60, o desenvolvimento da eletro-acústica tornou possível a realização de peças musicais que prescindiam da atuação de intérpretes, prenunciando o seu desaparecimento num futuro próximo. Porém, o que tem se notado é que a *computer music* não apenas traz um novo vigor ao papel do intérprete como proporciona uma aproximação muito maior entre este e o compositor: a maneira aberta como se produz música com computadores exige uma interatividade do músico com a obra muito maior que a existente entre o instrumentista e a composição tradicional mediada pela partitura. Além disso, a noção de intérprete se expande para além da figura do instrumentista habilidoso ao exigir-lhe uma atuação mais completa que pode estar envolvida com outras mídias que não a sonora.

### Um novo músico

Finalmente, é preciso pensar a respeito de novas possibilidades dentro da própria informática musical. O computador ainda hoje é, na maioria dos casos, um protótipo da conhecida máquina de Turing e seu funcionamento se baseia na arquitetura projetada por von Neumann. Seu modelo consiste em uma *unidade de memória* e uma *unidade central de processamento* (CPU) que realiza seqüências operacionais sobre a informação binária armazenada em alguma parte da memória. Estamos, de certa forma, tão habituados a esse modelo que não percebemos o quanto ele determina os tipos de operações que realizamos no computador.

Na verdade, quando utilizamos o computador, seja em música, seja em outro campo qualquer, estamos sujeitos a um tipo particular de linguagem que é imposta pela estrutura intrínseca destas máquinas. E como diz Roland Barthes em sua *Aula* [Barthes: 1978], a linguagem se define muito mais pelo que ela nos obriga a dizer do que pelo que ela nos permite dizer. De qualquer forma, por mais rápidas que sejam as CPUs, por maiores que sejam as memórias e por mais desenvolvidos que sejam os *softwares*, estamos sempre presos à linguagem da máquina de von Neumann.

Uma abordagem complementar a este modelo de computação tradicional que vem se desenvolvendo dentro dos mais diversos campos nos últimos anos, inclusive o da música, é a do conexionismo. O modelo conexionista de computação apresenta um ponto de vista bastante diferente que tem servido como uma alternativa em casos onde a computação clássica não se mostra satisfatória.

O conexionismo se inspira num modelo fisiológico da estrutura mental para conceber sua "máquina", chamada de *rede neural*. Enquanto sistemas clássicos constroem suas estruturas a partir do encadeamento serial de comandos, os sistemas conexionistas operam através de uma estrutura topológica, a rede, que contém virtualmente as informações ("conhecimento") do sistema. Aparentemente, as peculiaridades de cada um destes sistemas não fazem com que um se mostre mais ou menos eficaz que o outro, mas apenas determinam a especificidade de cada um [Andler:1990, Clark:1990]. Diferente dos sistemas clássicos, as redes neurais eliminam a dicotomia entre memória e CPU, estando aptas a trabalhar com informações incompletas. Além disso, o sistema pode incorporar processos de aprendizagem sem a necessidade de ser alimentado por regras formais (o que se mostra impossível no domínio da computação clássica). É óbvia a importância do modelo conexionista na atividade musical: pode-se criar redes capazes de realizar e avaliar eventos musicais segundo critérios que jamais conseguiríamos formalizar.

Apesar do desempenho visivelmente superior das redes neurais na solução de problemas onde existem informações incompletas ou que exigem flexibilidade no tratamento de dados, os procedimentos da computação clássica ainda são a melhor solução para tarefas que podem ser objetivamente formalizadas, como a ordenação de listas de dados ou a efetuação de cálculos. Visto que ambos os sistemas podem ser implantados num mesmo tipo de máquina, talvez a realização de programas híbridos, que incorporem técnicas clássicas e conexionistas, possa representar um aumento na capacidade de execução de tarefas e simulação de processos musicais em computador.

O que essas novas "linguagens" nos trazem é a capacidade de simular nossos modelos mentais, prever resultados e retroagir nos processos criados. A realização de uma obra musical não está mais presa às notas de uma partitura fixada sobre o papel, mas pode ser trabalhada sobre um outro tipo de modelo, dinâmico e interativo, que pode ser constantemente atualizado pelo usuário.

"Um modelo digital não é lido ou interpretado como um texto clássico, ele geralmente é explorado de forma interativa. Contrariamente à maioria das descrições funcionais sobre papel ou aos modelos reduzidos analógicos, o modelo informático é essencialmente plástico, dinâmico, dotado de uma certa autonomia de ação e reação" [Lévy:1993, 121].

Esse modelo digital está baseado em novas formas de representação que incorporam interfaces interativas, cujas possibilidades vão muito além daquelas oferecidas dentro do contexto da música realizada por meios tradicionais. Os papéis desempenhados pelos três elos da produção musical - compositor, intérprete e ouvinte - perdem sua especificidade e convergem, agora, a um único elemento: o *usuário*.

Através da interatividade com a máquina, o usuário, especialista ou não, pode ter acesso rápido e eficiente a dados sonoros de diferentes naturezas e manipulá-los de modos diversos, podendo aceitar ou rejeitar o produto de cada uma de suas intervenções para, recursivamente, chegar a um resultado que lhe agrade. O desenvolvimento de uma informática musical baseada em um conceito aberto de usuário faz vislumbrar uma espécie de democratização no acesso a todo o processo musical (por muito tempo restrito a uma classe de especialistas, os músicos), ao mesmo tempo que impõe uma reflexão cada vez maior a respeito da interação entre o universo sonoro e outros códigos (imagens, hipertextos, gestualidade, etc.) dentro da produção musical.

### Referências Bibliográficas

- Andler, D. (1990). Connexionisme et Cognition: à la recherche des bonnes questions. In *Revue de Synthèse* IV, 1-2 (95-127).
- Barthes, R. (1978). *Leçon*. Paris: Éditions du Seuil.
- Benjamin, W. (1985). A Obra de Arte na Era de sua Reprodutibilidade Técnica. In *Obras Escolhidas*, Vol. 1. Trad. de Sérgio Paulo Rouanet. São Paulo: Editora Brasiliense.
- Clark, A. (1990). *Microcognition: Philosophy, Cognitive Science, and Parallel Distributed Processing*. Cambridge: MIT Press.
- Iazzetta, F. (1993). *Música: Processo e Dinâmica*. São Paulo: Ed. AnnaBlume.
- Lévy, P. (1993). *As Tecnologias da Inteligência*. Trad. de Carlos Irineu da Costa. Rio de Janeiro: Editora 34.
- Perks, T. (1987). The Future of Music. Compiled by Larry Polansky. In *Leonardo*, vol. 20, n.º 4, pp. 363-365.
- Schaeffer, P. (1966). *Traité des Objets Musicaux*. Paris: Editions Seuil.
- Varèse, E. (1983). *Écrits*. Trad. para o francês de Christiane Léaud. Paris: Christian Bourgeois Éditeur

## A New Technology for Musical Sound Synthesis and Control

RICHARD HODGES

*Gibson USA  
2560 9th St.  
Berkeley CA 94710*

### Abstract

In using electronic music synthesis for performance, an important issue is providing the musician with a familiar and powerful way of controlling the instrument. Keyboards have been the most common control device, but the keyboard is limited in expressive possibilities compared to other instruments such as the guitar. We are developing an electronic instrument which uses the electric guitar as a performer interface which controls a wide palette of sounds and special effects.

We use a special guitar that is fitted with a fret scanning system and with separate magnetic pickups on each string. These inputs are processed digitally to analyze the guitarist's musical gestures. Several parameters are calculated for each string in real time and passed to a synthesis engine which plays sampled sounds and applies controllable modifications to the samples.

In another mode of operation, the actual signals from the guitar strings are modified by combinations of basic algorithms such as non-linear distortion, pitch shifting, and tuneable filtering.

This instrument incorporates a new digital protocol for communicating music performance data, called ZIPI. This protocol has features lacking in MIDI, including real-time updating of information for individual notes.

### Electronic music instruments: promise unfulfilled

Electronic musical instruments have become very important in live musical performance and studio production. They have several advantages over conventional acoustic and electro-acoustic instruments: wide palette of sounds; flexibility and ease of control; and reproducibility of sound, rhythm, and intonation. But they have several disadvantages also. Principally, these stem from the keyboard model on which most electronic instruments are based. The keyboard has severe limitations as a musical interface and control device. Control of musical events is limited basically to timing of attack and release, and velocity of attack (of course some keyboards also support after-touch). Electronic instruments differ greatly from traditional instruments in which the musician is directly in touch with the sound producing vibrations. In wind instruments, he can make subtle and rapid changes in timbre and pitch by changing the pressure of his lips, tongue, and diaphragm. In plucked string instruments such as the guitar, he can change the timbre by where and how he plucks the strings and by touching the strings before and during the note, and he can change the pitch by "bending" the strings. In bowed string instruments, the possibilities of control are even greater.

Additional limitations on electronic instruments are incurred by the nearly universal electronic protocol known as "MIDI" which is used for controlling, recording, and communicating performances of electronic musical instruments. MIDI has two chief problems which relate to its origin as a way of communicating keyboard data: speed of communication, which can result in noticeable delays in musical events; and lack of a flexible and powerful way to specify modifications to ongoing notes.



As a result of all these factors, musicians have been frustrated by the inability to combine the advantages of electronic and conventional instruments. The benefits of electronic sound generation and manipulation have not been extended to the widest possible range of different kinds of music.

#### A new kind of electronic instrument

We are attempting to address the limitations of electronic music with a new family of instruments that provide much more intimate real-time control to the musician, and a wider palette of sounds and sound-modification algorithms. The first instrument in this family is a second-generation instrument based on the electric guitar. The basic aim is to enable the use of the electric guitar as a controller in such a way that the skilled guitarist can employ the large repertoire of instrumental techniques he has laboriously developed to control the synthesis and processing of musical sounds. Other instruments, including violin, woodwinds, voice, and drums, can also be fitted with electronic pickups and used as inputs to this new device, giving a similar range of expressive power to skilled performers on these instruments.

This work builds on experience gained with the guitar- and violin-based synthesizers developed and marketed by ZETA Music, which is now a division of Gibson USA. The principal predecessor instrument, the ZETA Mirror-6 MIDI guitar, used a combination of fret scanning, independent electronic pickups on each string, and real-time pitch analysis to produce MIDI signals that can be used to control a conventional electronic synthesizer.

The new instrument goes far beyond the earlier ZETA instruments in several directions. The continuing evolution of digital audio electronics allows much more complex processing of the guitar signals in real time within the practical constraints of a commercially viable package and makes possible many new capabilities in electronic music instruments. Features of the new instrument include:

- Built-in 16-bit sample playback synthesis engine, with high-quality sounds in internal ROM
- PCMCIA card interface for libraries of additional sounds, patch storage, and firmware upgrades
- Stereo output, with controllable pan for each voice
- Fast and accurate tracking of guitar strings vibration parameters
  - Pitch (fret pitch and instantaneous pitch)
  - Amplitude
  - Spectral envelop
  - Spectral Timbre analysis
- Complex synthesis engine
  - Multiple-component sound wave files (odd harmonics; even harmonics; non-pitched component)
  - Post-synthesis filter with real-time controllable parameters
  - Real-time matrix mapping of analysis parameters and external inputs to control synthesis
- ZIPI (and MIDI) protocol for fast flexible communication of control parameters
  - Can control other synthesizers
  - Can respond to external controllers
- Hexaphonic waveform mode processes string signals individually
  - Six channels of 16-bit low-noise digital-to-analog conversion
  - Independent processing of each string signal
  - Variable pitch-shifting and harmonizing
  - Other nonlinear algorithms
  - Pre- and post-filtering, controllable in real time

#### Implementation

The new instrument employs several high-speed digital processors, including CISC, RISC, and DSP technology. These processors perform all functions for analyzing and synthesizing waveforms, user interface, and communications, allowing functions of the instrument to be upgraded and customized in the field by loading new firmware.

In order to obtain accurate string pitch information, we perform a time-domain analysis of the waveform of each individual string. Based on our experience with the ZETA Mirror-6 MIDI guitar controller, the time periods are measured between successive inflection points of the waveforms and a variety of heuristics are

applied to these raw measurements. These heuristics are aided by knowledge of the position at which the string is fretted; allowance has to be made however for special techniques such as harmonics and deep whammy-bar pitch bends. A very accurate value for string vibration frequency can be determined in a little more than one complete period of the waveform. Accurate control signals for synthesis are produced with a time delay that is musically minimal.

We also use a digital processor to perform a spectral analysis. This enables us to measure parameters reflecting the timbre of the string signal. Spectral tilt is determined as a weighted ratio between high- and low-frequency components. We also measure the proportion of three different components of the string signal: odd harmonics; even harmonics; and non-harmonic vibrations (i.e. components of the sound not resulting from the vibration of the string). These kinds of information are computed in real time and combined according to the user-specified program or "patch." The results are used as control information for the synthesis engine.

A digital processor is used for synthesis of output waveforms. Sample files representing different instrument voices are accessed from built-in ROM or from a PCMCIA card; as in conventional sampling synthesizers, these instrument samples are interpolated and resampled to shift them to the pitch determined by the user patch. Often, the synthesized pitch would be the actual instantaneous pitch of the guitar string; however one major advantage of our instrument is that the pitch can be transposed by a fixed tonal interval or otherwise modified for musical effects. Each instrument voice is actually represented by three different files which contain the odd harmonic information, the even harmonics, and the non-harmonic information for the voice. These files are obtained by complex off-line processing of sound recorded from actual instruments under studio conditions.

Digital filtering is applied to the voices synthesized for the six strings. The resulting signals are mixed and panned between left and right output channels. These functions are also controlled by analysis parameters as specified in the user patch. Finally the signals are sent to two 16-bit digital-to-analog converters for output to external sound amplification equipment.

In the hexaphonic waveform mode, the processors perform a variety of algorithms which act directly on the waveforms generated by the strings and modify them; it is these modified waveforms that are sent to the output, rather than synthesized waveforms. This is an important extension of the technique of applying deliberate electronic distortion to the sound of the electric guitar. This technique became enormously important in commercial music due to the success of the genre of Heavy Metal. In our instrument, the modifying algorithms can be applied to the waveform of each string individually, or to combinations of the string signals. The advantage is that whereas traditional distortion devices inevitably result in cross-modulation products of the signals of the simultaneously played strings, in our instruments, this does not necessarily occur. Of course cross-modulation is sometimes musically desirable, but with our instrument it can be controlled. Also, importantly, our method allows the natural dynamics of the guitar to be reflected in the output dynamics if desired, as opposed to conventional nonlinear processing which results in dramatic compression of the dynamics.

The power of digital processing allows much more complex algorithms than the simple nonlinear mapping of waveform that was characteristic of previous distortion devices. We have implemented algorithms for pitch shifting and harmonizing of the guitar notes, preserving all timbral and dynamic qualities of the original signal. Other algorithms are under development which generate transformations of string timbre.

In the hexaphonic mode, individual filters for each string are also implemented which act both before and after nonlinear processing. These filters can be controlled in real time based on measured properties of the string signals.

#### A new paradigm for electronic music instrument communication

Contemporary practice in electronic music is to use modular units connected by MIDI cables. A typical setup for live performance or studio recording consists of several units including controllers (keyboards, drum pads, breath controllers, guitar and violin controllers), sound generators (samplers are the most common type today, although FM and other technologies are widely used also), and integrated units combining a keyboard and sound generation capability. In addition, personal computers are often fitted with MIDI interfaces which allows them to run software to record, play back, and modify MIDI data during performances. The development of the MIDI standard began about a decade ago in response to the need for such networks of music performance devices.

However, MIDI is saddled with numerous limitations. First, there is speed. In order to utilize the serial communication IC's used in early personal computers, the communication speed was limited to 31.25 kbaud.

Since the basic performance model of MIDI was the keyboard controller, which can only generate a limited number of note-on and note-off events at one time, this was marginally adequate. For example, the attack information for a ten-note chord can be transmitted in about 7 milliseconds, which is on the borderline of perceptibility. However, serious speed problems appear when several controllers are sharing the same MIDI network, a common practice, when musicians try to use techniques that depend on very accurate time relationships between events, for example flam techniques on percussion controllers, and especially when information from pedals, joysticks, and other continuous controllers needs to be updated at a rapid rate. Communicating guitar controller information becomes very difficult; just to update pitch bend information 100 times a second for six strings would exceed MIDI bandwidth.

Another category of MIDI limitation stems from its implicit model of musical events. Basically, MIDI assumes that the principle information about a musical event is established at the attack, and comprises simply semitone pitch and velocity (a keyboard concept translating into amplitude). This model derives from MIDI's heritage as a representation for keyboard and percussion events, where it is appropriate. But it is not a good model for instruments where the performer remains in control of pitch, amplitude, and timbre during the course of the entire note. Some features were added to MIDI to try to accommodate continuous information: controllers based on the joystick and foot pedal model; keyboard aftertouch; and pitch bend. These features are not well integrated in the MIDI model however, and different implementations are not always compatible.

People have been complaining about these problems ever since MIDI first began to be used, but prior to our efforts, no one had done anything about them. Faced with the requirement for real-time musical communication suitable at least to the guitar controller, we have developed a new protocol called ZIPI. This protocol addresses the major problems with MIDI that we have mentioned and serves the future needs of a wide variety of different controllers and performance situations.

The ZIPI protocol is designed on the philosophical base of contemporary practice in high-speed digital communications, and conforms to the OSI layered network model. At the physical layer, we define an electrical specification and a 250 kbaud (minimum) serial stream. At the data link layer, we use a token-ring architecture for deterministic performance. In complex setups where there are several controllers and synthesizers, these instruments would normally be connected in a star topology through a hub which would implement the ring and maintain reliable communications even in the presence of transmission errors and events where instruments go on-line or off-line during a performance.

The main content of the ZIPI protocol is at the application layer of the OSI model, where we define several protocols, most importantly a Music Parameter Description Language, MPDL, for communicating real-time music event control (McMillen, Wessel, & Wright 1994). A key feature is the concept of a hierarchical address space. Control messages can be addressed to instruments, to families (of instruments), or to individual notes. Addressability of notes is an important advance over MIDI, since it allows a single note to evolve over time, changing its properties such as pitch, time, and amplitude, in response to ZIPI messages addressed to it.

The messages "trigger" and "release," which start (or rearticulate) a note and stop it respectively do not in themselves contain information about the pitch or amplitude of the note. These properties of a note are established by other ZIPI messages that are sent before or during the note. A large number of different types of messages are given predefined meanings (although detailed semantics are in many cases left to the implementation, for example the mapping of the "amplitude" message value into physical units). Some message types provide redundant ways to control the same physical parameter, for example "pitch," specified in semitones and fractions of semitones, and "frequency," specified in Hertz. Timbre space is addressed by predefined messages such as "brightness," "Even/Odd harmonic balance," "roughness," "attack character," and by messages specifying abstract timbre space coordinates, the meaning of which is left entirely to the implementation of individual instruments. Other messages are provided for spatial location of instruments (a generalization of pan), for modulation (a generalization of vibrato), and for synthesizer housekeeping functions such as defining the priority of a note for allocation to available instrument voices.

In addition to messages whose semantics are defined in terms of parameters that affect the way a synthesizer is to generate note sounds, another class of messages is provided whose semantics are specified in terms of measured performer gestures. Examples of these messages include "key velocity," "pick/bow velocity," "lip pressure," "drum-head position," "position in space" (i.e. position of a magic wand-type controller), etc. In MIDI, there was no distinction between these two classes of message, and in practice the semantics of messages was sometimes interpreted as if they were what we would call synthesizer control messages, sometimes as performer gesture messages; this confusion was the cause of some cases of inconsistent behavior with MIDI.

ZIPI provides communication bandwidth and control flexibility able to be the basis for whole orchestras combining many heterogeneous controllers and synthesizers, and supports a vast range of expressive possibilities for many new types of electronic music performance environments.

### Conclusions

We have developed a new generation of electronic music instruments allowing an unprecedented degree of control and expressiveness. A guitar controller with integrated sound synthesis and processing capabilities will be the first member of this family. It will be released as a commercial product early in 1995.

To support our new control concepts and sound generation methods, we are also introducing a new protocol for digital communication between instruments in a performance network. These two elements are enabling technologies, allowing electronic instruments to enter a new age of diverse musical creativity.

### References

- International MIDI Association (IMA) (1988) *MIDI 1.0 Detailed Specification, Document Version 4.0*, Los Angeles, CA, IMA.
- Loy, G. (1985) Musicians Make a Standard: The MIDI Phenomenon. *Computer Music Journal*, 9(4), 8-26.
- McMillen, K. (1994). ZIPI—Origins and Motivations. *Computer Music Journal*, 18(4). [in press]
- McMillen, K. Wessel, D. & Wright, M. (1994). ZIPI's Music Parameter Description Language. *Computer Music Journal*, 18(4). [in press]
- McMillen, K. Simon, D. & Wright, M. (1994). ZIPI Network Summary. *Computer Music Journal*, 18(4) [in press]
- Moore, F. R. (1988). The Dysfunctions of MIDI *Computer Music Journal* 12(1), 19-28. In Deutsch, D., ed. *The Psychology of Music 2nd Edition*, Academic Press.
- Scholz, C. (1991). A proposed extension to the MIDI specification concerning tuning. *Computer Music Journal*, 15(1), 49-54.
- Wright, M. (1994). ZIPI Examples. *Computer Music Journal*, 18(4). [in press]
- Wright, M. (1994). MIDI/ZIPI Comparison. *Computer Music Journal*, 18(4). [in press]
- Wright, M. (1994). ZIPI Frequently Asked Questions. *Computer Music Journal*, 18(4). [in press]

### Acknowledgements

I would like to acknowledge the creators and implementers of the concepts reported here: Keith McMillen, Chief Scientist, Gibson Western Innovation Zone Labs (G-WIZ), and the staff of G-WIZ, especially Marie Baudot; and David Wessel and the staff of the Center for New Music and Audio Technology (CNMAT), music department, University of California at Berkeley.

**Virtual Musical Instruments:  
Accessing the Sound Synthesis Universe as a Performer.**

AXEL MULDER  
*School of Kinesiology  
Simon Fraser University  
Burnaby, B.C., V5A 1S6 Canada*

**Abstract**

With current state-of-the-art human movement tracking technology it is possible to represent in real-time most of the degrees of freedom of a (part of the) human body. This allows for the design of a virtual musical instrument (VMI), analogous to a physical musical instrument, as a gestural interface, that will however provide for much greater freedom in the mapping of movement to sound. A musical performer may access therefore the currently unexplored real-time capabilities of sound synthesis systems. In order to decrease the learning and adaptation needed and avoid injuries, the design must address the musculo-skeletal, neuro-motor and symbolic levels that are involved in the programming and control of human movement. The use of virtual musical instruments will likely result in new ways of making music and new musical styles.

**Introduction**

In figure 1 an attempt has been made to show the development of musical instruments. Acoustic instruments transduce movements of a performer into sound. The performer has limited timbral control and a limited gesture set that can hardly be adapted to the performer's needs. Electroacoustic instruments do not allow for more gestures or adaptivity, but increase the range of sounds compared to the acoustic instrument without the electronic gadgetry. Normally this increased range of sounds is an extension of the sounds that are produced with the original acoustic instrument.

Examining electronic musical instruments it can be seen that gestural interfaces for sound synthesis systems have largely been copied from traditional physical musical instrument designs (e.g. MIDI keyboard or piano, MIDI guitar, Yamaha wind controller, percussion controller, Zeta violin, MIDI accordion etc., see Pressing (1992)). Many performing musicians are dissatisfied with the expressive capabilities of these instruments when compared with traditional acoustic instruments. On the one hand this dissatisfaction can be attributed to limited resolution, accuracy and responsiveness of the gestural interface, on the other hand the sound synthesis system that is driven by the gestures, usually via MIDI, is not adequate to satisfy the auditory needs of the performer. Another important point that is seldomly addressed, is that such gestural interfaces do not allow for real-time control during performances of a number of parameters for sound synthesis that were not available on traditional physical musical instruments.

With current state-of-the-art human movement tracking technology it is possible to represent in real-time most of the degrees of freedom of a (part of the) human body. This allows for the design of a virtual musical instrument (VMI), analogous to a physical musical instrument, as a gestural interface, that will however provide for much greater freedom in the mapping of movement to sound. A musical performer may access therefore the currently unexplored real-time capabilities of sound synthesis systems. Although it would be possible to build a simulation of, for example, a guitar as a VMI it may be more interesting and desirable to design a VMI that matches as closely as possible the capabilities of the human performer, such that less learning and adaptation is needed and injuries avoided.

In order to achieve these ergonomic goals the musculo-skeletal, neuro-motor and semiotic issues that are involved in the design of a VMI must be addressed. It is advisable to limit the scope of this research by defining a class of VMIs that are controlled only through movements of the upper limbs and which is perceived mainly through kinaesthetic and auditory feedback as an "audio-kinaesthetic object", thereby leaving out tactile and force feedback. However, some indirect tactile (e.g. fingers touching each other) and visual (e.g. seeing your own fingers) feedback remains available.

It is likely that the resulting virtual instrument cannot be physically constructed, so that, in the case of music, new ways of making music and musical styles may develop. Some of these numerous possibilities for musical performers to change the presentation of musical ideas and the re-integration of music with dance will be discussed.

#### Electronic musical instruments: Overview of experiments with new controllers

Due to the development of (computer) technology, many musicians who were technically oriented, have experimented with these technologies to change various aspects of musical performance. On the one hand much effort has gone into designing new ways of sound generation. On the other hand some effort has gone into designing new controllers, physical devices that implement a motion sensing technology and translate that motion, generated by the performer, into a (MIDI) signal that in its turn controls a sound synthesis device. Tables 1a to 1d list some of the people and their designs, classified mostly by the human movement tracking method. In the case of use of a glove for conducting purposes this classification scheme does not work perfectly as one can see. In essence this shows the difficulty in providing clear boundaries in the levels of control of human movement and the levels operating in auditory perception.

At any rate, these new musical instrument designs were hardly or not at all based on a model of human performance, nor on a model of human auditory perception, let alone their relation. Therefore, most of these designs are not more attractive to use for a performer, more than traditional acoustic instruments. In effect, most of these designs were mainly concerned with the implementation of the technology instead of exploring the use of psychomotor parameters in the human system.

Table 1a. An overview of recent experiments with new musical instrument designs.

Author	Musical instrument
Rubine & McAvinney (1990)	Videoharp MIDI controller (optical sensing of fingertips)
Matthews & Schloss (1989)	Radiodrum MIDI controller (short range EM sensing)
Palmtree Instruments Inc., La Jolla CA, USA	Airdrum MIDI controller (acceleration sensitive)
Cadoz, Luciani & Florens (1984)	CORDIS modular feedback keyboard, physical modeling approach to musical instruments
Ray Edgar, STEIM, Amsterdam, the Netherlands	Sweatstick MIDI controller (stick with buttons a.o.)
STEIM, Amsterdam, the Netherlands	Web MIDI controller (strain gauge based)
Michel Waisvisz, STEIM (see Krefeld, 1990)	Hands MIDI controller (buttons, ultrasound ranging and mercury switches)
Steve O'Hearn, Rhode Island School of design, USA, see Moog (1989)	Design of a new stick-like MIDI controller, with pressure sensitive buttons)

Table 1b. An overview of recent experiments with devices for conducting.

Author	Conducting device
Keane & Gross (1989)	MIDI baton (AM EM sensing)
Bertini & Carosi (1992)	Light baton (LED sensed by CCD camera)
Don Buchla	Lightning MIDI controller (Infrared LED sensing)
Morita, Hashimoto & Ohteru (1991)	Dataglove for conducting a MIDI orchestra
Machover & Chung (1989)	Exos DHM glove for conducting MIDI devices, hyperinstrument concept.

Table 1c. An overview of recent experiments with new musical instrument designs involving gloves.

Author	Glove application
Pausch & Williams (1992)	Handmotions control speech synthesizer
Fels & Hinton (1993)	GloveTalk: CyberGlove controls a speech synthesizer
CNMAT, Berkeley; Scott Gresham-Lancaster, Berkeley; Mark Trayle, San Francisco; James McCartney, University of Texas; Thomas Dougherty, Stanford University; William J. Sequeira, AT&T; Tom Meyer, Brown University; Rob Baaima, Amsterdam; and probably many others	Powerglove as a MIDI controller
Joel Ryan, Laetitia Sonami	Hall-effect glove as a MIDI controller
Tom Zimmerman, VPL; Banff Center for the Arts, Calgary, Canada; Brian Karr & Lucinda Hughey, Seattle WA and probably others	Dataglove as a controller
Gustav's Party	Virtual reality rock band using a.o. Datagloves as MIDI controllers

Table 1c. An overview of experiments with new musical instrument designs involving whole body movements.

Author	Motion to sound design
Leon Theremin (see Vail, 1993)	Capacitively coupled motion detector controls electronic oscillator
Chabot (1990)	Ultrasound ranging to detect whole body movements and to control MIDI devices
Bauer & Foss (1992)	GAMS: Ultrasound ranging to detect whole body movements and to control MIDI devices
Leslie-Ann Coles, USA	Bodysuit performance during CHI 92
Coniglio (1992)	MIDI Dancer bodysuit / Interactor mapping software
Yamaha Corp., Japan	MIBURI: arm gestures, finger buttons to MIDI translation
David Rokeby, Toronto, Canada	Very Nervous System: video image processor translates movement into MIDI
Fred Kolman, Amsterdam, the Netherlands	Video image processor translates movement into MIDI
Camurri (1987)	Costel opto-electronic human movement tracking system controls a knowledge based computer music system

Table 1d. An overview of experiments with new musical instrument designs involving bioelectric signals.

Author	Bioelectric design
Knapp & Lusted (1990)	Biomuse EMG signals control a DSP
Chris van Raalte / Ed Severinghaus, San Francisco CA, USA	BodySynth MIDI controller
Rosenboom (1990), Richard Teitelbaum, Germany, Pjotr van Moock, the Netherlands and probably others	EEG/EMG interface to synthesizer setup

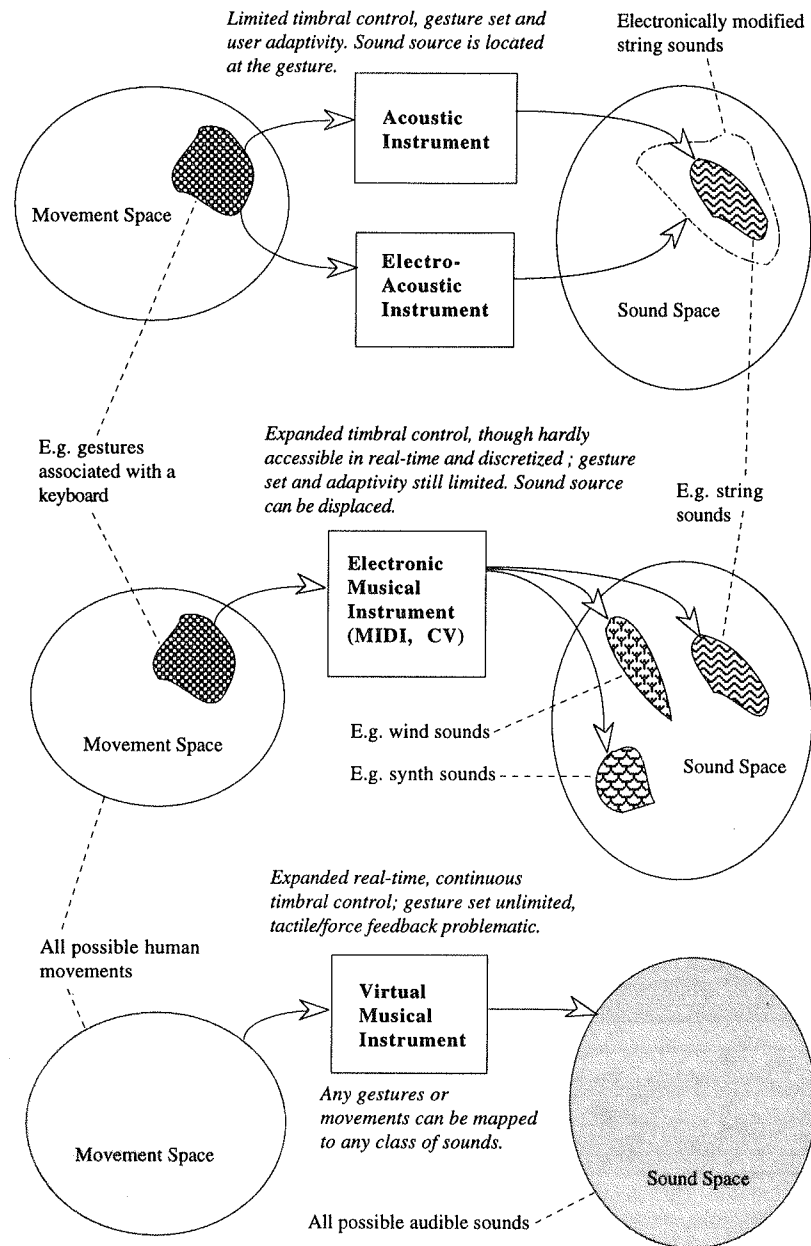


Fig. 1. The development of musical instruments.

In addition to the above designs an innumerable number of setups has been created that involve (electro-) acoustic instruments equipped with electronics, on the instrument or as signal processors, or computers. Such setups are generally called performances with live electronics. As the input or controller is usually not the innovative component, but the way in which the sounds are processed or generated in such setups, no further attention will be paid in this paper to such designs. Also, setups where the computer, via keyboard or mouse, is the main controlling device will not be further considered here.

### Psychomotor issues

Pressing (1990) addresses musical instrument design from an ergonomic point of view. Although his work is very useful and important, it is not based upon work in the field of motor control, but more on common musical instrument design knowledge and performance experience. The field of motor control specifically aims to provide models of human movement through empirical evidence.

Human movement control can be conceived of as taking place at musculo-skeletal, neuro-motor and symbolic levels, where each of these levels interact. So far gestural interfaces have mainly addressed the musculo-skeletal level (e.g. number of degrees of freedom, movement range, bandwidth and resolution), whilst some work has been done at the symbolic level (e.g. American Sign Language recognition). Neuro-motor aspects that can be included in gestural interfaces, although they are mostly unresolved as yet, are amongst others the control of speed and accuracy and their trade-off, the parameters that specify the movement at the neuromotor level (i.e. not joint angles may be relevant parameters in formulating a gesture but higher level parameters), the various phases during a movement, each of them with different control parameters and timing and the various internal representations of the movement and their transformations. Central to the neuromotor level is the concept of a motor program.

Additionally, the identification and structure of the audio-motor channel, similar to the visuo-motor channel of Jeannerod (1990), may provide a significant framework. For instance, what is the relation of the visuo-motor channel and the audio-motor channel. At a low level, via perturbation experiments in singing, e.g. perturbed feedback of pitch (shifted pitch), timbre (formant remapping), or timing (delayed signals), a definition of the lower level aspects of the audio-motor channel may be obtained. The recognition of gestures has up to now been implemented with engineering techniques that describe the movement in physical terms at a musculo-skeletal level. Recognition in terms of neuromotor and symbolic level models of human movement is as yet unimplemented, although some work has been done using a connectionist paradigm (see below).

Palmer & van de Sande (1993) and Palmer (1989) are concerned with music theories and linguistic theories in the way they relate syntactic (structural) representations to phonetic (sounded) representations. The object of their studies is the ordering and manipulation of "sound objects" (phonemes, phones) as symbols, and not the creation of the sound objects (with their pitch, duration, volume, and timbre functions) themselves. Shaffer (1989), also concerned with high level aspect of motor programs, discusses the implementation of music performance by a robot. Sloboda, (1985) discusses three basic aspects of music performance: sight reading (how are performance plans acquired), performance after repeated exposure or rehearsal (what is the role of feedback in performance plans) and expert or skilled performance. Clynes & Nettheim (1982) work addresses emotion and meaning of human movement. Baily (1985) studied movements patterns of African music players. His work investigates a.o. whether the spatial properties of an instrument may influence the shape of the music played on it.

Whilst most of this research is done by researchers in behavioural science, approaches by choreographers may be helpful too. Their efforts include the definition of dance notation systems, e.g. Labanotation which specifically addresses the concept of effort in movement. This can be conceived of as a approach to approximate psychomotor parameters. The concept of effort, as used in labanotation, has been further explored in a musical instrument design context by Ryan (1992) and Ryan (1991).

It is interesting to note that the computer music community has paid a great deal of attention to production of sound directly from abstract thought by implementing a model of cognition, using e.g. artificial intelligence technology. These models did usually only address the higher levels involved in the performance of music, in contrast to human music performers, who physically effectuate the performance - with effort. Also, such models only apply in cases where the musical style is well defined and formalized, and does not apply in situations that involve a great deal of improvisation. Pressing (1984) outlines some of the cognitive aspects in improvisation.

## VMI's: the future ?

From the above and figure 1, a VMI is characterised by at least two features. Any gesture can be used to control the sound synthesis process and the mapping is entirely programmable and limited by the sound synthesis model only. In addition, the mapping possibly incorporates motor control or ergonomic principles and may be adaptive to the user. In other words, the shape and sound generation and their relation of a VMI are not defined by physical laws necessarily, but can be arbitrarily defined, most likely by ergonomic principles. Due to the fact that music can be performed at various levels a VMI can be conceived of in various ways. At a high level, a virtual orchestra, explored by Morita et al (1991), comes to mind. At a low level Gibet (1990) explored virtual drumming by modeling a human arm and a vibrating surface (using the synergetics approach of Haken). Stephen Pope at CCRMA, Stanford CA, USA, explores virtual sound objects and Bolas & Stone (1992) explored a virtual theremin and a virtual drum, using virtual reality technology. Obviously the relation between dance and music can become very tight using the above ideas (Mulder, 1991). Ungvary et al (1992) presented their system NUNTIUS, which provides direct data transfer and interpretation between dance and music. Their system implements high level relations between dance and music using amongst others labanotation.

The author's work included experiments with an instrumented bodysuit that registered human joint angles. Two performances were implemented in 1993. In one performance the values of the joint angles of the performer (Egmont Zwaan) were used to drive a Lexicon LXP 5 effects processor that processed the voice of the performer. During the performance, the performer lost track of what he was actually controlling, i.e. he was more involved with his movements than the aural result and/or there were too many parameters to control (more learning is needed). Also, the mapping was too simple, too direct, i.e. some parameters were too sensitive to movement. There was no compensation for interaction between various acoustic effects, that resulted in changing sensitivities of the parameters. Last but not least the LXP 5 had some problems processing the amount or combinations of MIDI data.

In the other performance only a few joint angle values of the author's movements were used to drive the effects processor. The performance included Mari Kimura playing a Zeta MIDI violin and a dancer (Anita Cheng). The effects processor transformed the violin signals. The dancer interacted physically with the author and symbolically with the violin player, so that all performers were communicating. The most interesting result in the context of this paper was that the author was dancing and not playing an instrument - while in fact he was. This illustrates the possibilities for merging dance and music as one art form, as it used to be (and still is) for many African tribes.

The translation of the joint angles into the parameters of the LXP 5 was simple - there was no algorithm involved. Due to this simple mapping the VMI was not very intuitive. However, the aim of user adaptivity was achieved: it was possible to map any movement to any parameter of the LXP 5. Also the real-time capabilities of the sound processing device were fully used. Furthermore it became very clear that human movement tracking is a hard problem (Mulder, 1994).

Other ideas that may be worthwhile exploring are control of a singing or speech synthesizer by hand or arm gestures or even whole body movements. Lee & Wessel (1992) have built systems that use artificial neural nets to implement a control structure that adapts to nonlinear human behaviour. Similarly, Fels & Hinton (1993) have used neural networks to achieve translation of hand shape and position to articulated speech. The speech synthesizer may also be replaced by a granular synthesizer processing a text sample, a vocoder or an effects processor (as above) with voice input. It would be possible then for instance to present a poem with very strange and strong expression, both gesturally and acoustically. Currently the author is investigating use of an instrumented glove to control a granular synthesis system developed by Barry Truax and Harmonic Functions in Vancouver, BC, Canada.

Another obvious performance idea would be to use an instrumented suit to control percussive synthesizers. The fact that disco, house, African and many other dance forms involve mostly repetitive movements may allow for interpretation or recognition in terms of the so-called dynamic pattern approach in human movement control. As for the performer, he or she might become involved in an intense audio-kinaesthetic experience.

A future musical ensemble may consist of a drummer or percussive controller, various timbral (sound) controllers, various melodic controllers (musical structure controllers), a spatialization controller and an effects controller. All the movements of these performers would be choreographed to achieve a maximum performance effect.

## References

- Baily, J. (1985). Music structure and human movement. In: Howell, P., Cross, I., (editors), *Musical structure and cognition*, 237-258. London, UK: Academic Press.
- Bauer, W. & Foss, B. (1992). GAMS: an integrated media controller system. *Computer Music Journal*, 16 (1), 19-24.
- Bertini, G. & Carosi, P. (1992). The light baton: a system for conducting computer music performance. *Proceedings International Computer Music Conference*, San Jose, California, USA, 73-76. San Francisco CA, USA: International Computer Music Association.
- Bolas, M. & Stone, P. (1992). Virtual mutant theremin. *Proceedings International Computer Music Conference*, San Jose, California, USA, 360-361. San Francisco CA, USA: International Computer Music Association.
- Cadoz, C., Luciani, A. & Florens, J-L. (1984). Responsive input devices and sound synthesis by simulation of instrumental mechanisms: The CORDIS system. *Computer Music Journal*, 8 (3), 60-73.
- Camurri, A. et al (1987). Interactions between music and movement: A system for music generation from 3D animations. Proceedings of the 4th international conference on event perception and action, Trieste.
- Chabot, X. (1990). Gesture interfaces and a software toolkit for performance with electronics. *Computer Music Journal*, vol 14 no 2 p 15-27.
- Clynes, M. & Nethem, N. (1982). The living quality of music: neurobiologic basis of communicating feeling. In: Clynes, M., (editor), *Music, mind and brain: the neuropsychology of music*, 47-82. New York, USA: Plenum Press.
- Coniglio, M. (1992). Introduction to the Interactor language. *Proceedings International Computer Music Conference*, San Jose, California, USA, 170-177. San Francisco CA, USA: International Computer Music Association.
- Fels, S.S. & Hinton, G.E. (1993). Glove-talk: A neural network interface between a dataglove and a speech synthesizer. *IEEE Transactions on neural networks*, 4 (1), 2-8.
- Gibet, S. & Marteau, P.-F. (1990). Gestural control of sound synthesis. *Proceedings International Computer Music Conference*, Glasgow, UK, 387-391. San Francisco CA, USA: International Computer Music Association.
- Jeannerod, M. (1990) The neural and behavioural organization of goal directed movements. New York, USA: Oxford University Press.
- Keane, D. & Gross, P. (1989). The MIDI baton. *Proceedings International Computer Music Conference*, Columbus, Ohio, USA. San Francisco CA, USA: International Computer Music Association.
- Knapp, R.B. & Lusted, H. (1990). A bioelectric controller for computer music applications. *Computer Music Journal*, 14 (1), 42-47.
- Krefeld, V. (1990). The Hand in the Web: An interview with Michel Waisvisz. *Computer Music Journal*, 14 (2), 28-33.
- Lee, M. & Wessel, D. (1992). Connectionist models for real-time control of synthesis and compositional algorithms. *Proceedings International Computer Music Conference*, San Jose, California, USA, 277-280. San Francisco CA, USA: International Computer Music Association.
- Machover, T. & Chung, J. (1989). Hyperinstruments: Musically intelligent and interactive performance and creativity systems. *Proceedings International Computer Music Conference*, Columbus, Ohio, USA. San Francisco CA, USA: International Computer Music Association.
- Mathews, M. & Schloss A. (1989) The radiodrum as a synthesis controller. *Proceedings International Computer Music Conference*, Columbus, Ohio, USA. San Francisco CA, USA: International Computer Music Association.
- Moog, B. (1989). An industrial design student's MIDI controller. *Keyboard*, January, 108-109.
- Morita, H., Hashimoto, S. & Ohteru, S. (1991). A computer music system that follows a human conductor. *IEEE Computer*, July, 44-53.
- Mulder, A.G.E. (1991). Viewing dance as instrumental to music. *Interface* 4 (2), 15-17. Columbus, Ohio, USA: ACCAD, Ohio state university.
- Mulder, A.G.E. (1994). MyGlove: A glove input device based on the PowerGlove flex sensors. *PCVR*, 16.
- Palmer, C., van de Sande, C. (1993). Units of knowledge in music performance. *Journal of experimental psychology: learning*,

- memory and cognition, 19 (2), 457-470.
- Palmer, C. (1989). Mapping musical thought to musical performance. *Journal of experimental psychology: human perception and performance*, 15 (12), 331-346.
- Pausch, R. & Williams, R.D. (1992). Giving CANDY to children: user tailored gesture input driving an articulator based speech synthesizer. *Communications of the ACM*, 35 (5), 60-66.
- Pressing, J. (1990). Cybernetic issues in interactive performance systems. *Computer Music Journal*, 14 (1), 12-25.
- Pressing, J. (1992). *Synthesizer performance and real-time techniques*. Madison, Wisconsin, USA: A-R editions.
- Pressing, J. (1984). Cognitive processes in improvisation. In: Crozier, W.R., Chapman, A.J., *Cognitive processes in the perception of art*, 345-363. Amsterdam, The Netherlands: Elsevier Science Publishers.
- Rosenboom, D. (1990). The performing brain. *Computer Music Journal*, vol 14 no 1 p 49-66.
- Rubine, D. & McAvinney, P. (1990). Programmable finger tracking instrument controllers. *Computer Music Journal*, 14 (1), 26-41.
- Rubine, D. & McAvinney, P. (1990). Programmable fingertracking instrument controllers. *Computer Music Journal*, vol 14 no 1 p 26-41.
- Ryan, J. (1992). Effort and expression. *Proceedings International Computer Music Conference*, San Jose, California, USA, 414-416. San Francisco CA, USA: International Computer Music Association.
- Ryan, J. (1991). Some remarks on musical instrument design at STEIM. *Contemporary music review*, 6 part 1, 3-17.
- Shaffer, L.H. (1989). Cognition and affect in musical performance. *Contemporary music review*, 4, 381-389.
- Sloboda, J.A. (1985). *The musical mind: the cognitive psychology of music*. Oxford, UK: Clarendon press.
- Ungvary, T., Waters, S. & Rajka, P. (1992). NUNTIUS: A computer system for the interactive composition and analysis of music and dance. *Leonardo*, 25 (1), 55-68.
- Vail, M. (1993). It's Dr. Moog's traveling show of electronic controllers. *Keyboard*, March, 44-49.

## Índice por Autor

- Aguiar, G. L., 3  
 Arcela, A., 33
- Ballista, A. L. C., 139  
 Barbar, K., 99  
 Beckenkamp, F. G., 189  
 Beurivé, A., 99  
 Brandão, M. C. P., 39
- Cabral, E. E., 79  
 Castro, R. R. F., 155  
 Cerana, C., 21  
 Choi, A., 27  
 Christiansen, A. S., 225
- Desainte-Catherine, M., 99
- Engel, P. M., 189
- Farra, R. D., 185  
 Ferneda, E., 177  
 Fritsch, E. F., 107
- Gioia, O. G., 147
- Hitt, D., 133  
 Hodges, R., 237
- Jaffe, D. A., 53, 63
- Lazzetta, F., 231  
 Lo, Y., 133  
 Loureiro, C. A. J., 39
- Malt, M., 125  
 Manzolli, J., 45  
 Marar, J. F., 9  
 Martins, A. J. B., 139
- Meireles, A. O., 73  
 Miranda, E. R., 203  
 Moraes, M. R., 83  
 Morales, E., 169  
 Morales-Manzanares, R., 169  
 Moreira, E. S., 9  
 Mulder, A., 243
- Pachet, F., 195  
 Pimenta, M. S., 139  
 Pope, S. T., 161  
 Porcarro, N., 63  
 Prignano, I., 15
- Ramalho, G., 217  
 Rueda, C., 91
- Scipio, A., 15  
 Silva, C. A. P., 177  
 Silva, D. A. B., 121  
 Silva, H. M., 177  
 Smith, J. O., 63
- Teixeira, L. M., 177
- Vicari, R. M., 107
- Zannon, T. C., 39