UNIVERSIDADE CATÓLICA PORTUGUESA

COMPOSIÇÃO DE SISTEMAS MUSICAIS INTERACTIVOS

(COMPOSING INTERACTIVE MUSIC SYSTEMS)

Tese apresentada à Universidade Católica Portuguesa

para obtenção do grau de Doutor em Ciência e Tecnologia das Artes,

especialidade em Informática Musical

por

Miguel Marques Cardoso

ESCOLA DAS ARTES

Março 2015

UNIVERSIDADE CATÓLICA PORTUGUESA

COMPOSIÇÃO DE SISTEMAS MUSICAIS INTERACTIVOS

(COMPOSING INTERACTIVE MUSIC SYSTEMS)

Tese apresentada à Universidade Católica Portuguesa

para obtenção do grau de Doutor em Ciência e Tecnologia das Artes, especialidade em Informática Musical

Por Miguel Marques Cardoso

Sob orientação de António de Sousa Dias e Luís Gustavo Martins

This work was developed with the financial aid of

the Fundação para a Ciência e Tecnologia (FCT),

under the Programa Operacional Potencial Humano

(SFRH / BD / 62236 / 2009)

Acknowledgements

My family,

Daniela for these years of inestimable perseverance, patience, and incentive;
My mother;

Advisors António de Sousa Dias and Luis Gustavo Martins for guidance and confidence;

Luisa Ribas, without whose invaluable help, advice and collaboration this work could not have been completed;

José Aguirre, Andrés Ortiz, Raimon Mirada, Daniel Aguilar, Carlos Diego and all the Bestiario team for their support and allowing my absence for such a long period of time;

Ricardo Guerreiro, Santiago Ortiz, Tiago Henriques, Mónica Nogueira, Pedro Rodrigues, José Luis Ferreira, Victor Almeida, for the friendship and sharing insight;

Paulo Ferreira Lopes for the provided opportunities;

All the *SuperCollider* community for their determination in sharing knowledge.

(blank page)

Abstract

This work proposes a contribute to the understanding of computer-based interactive music practices, in order to establish methodologies that can provide a more transparent, flexible and dynamic process in related compositional activity.

This research is developed by examining and articulating complementary perspectives on the intersection of computer music with interaction. We begin by researching on models, materials, techniques and concepts that can contribute to this practice, attempting to understand the affordances and constraints that the computational medium provides. We then study how interaction has contributed to the contemporary context of interactive music practices by reshaping the relationship between the composer, the performer, the audience and the environment. In order to further scrutinize the potential of computer-based interactive music, we investigate new programming paradigms and how they can yield a more adequate context for artistic practices such as music. Complementing these explorations, we implement a computational framework, suggesting an alternate approach to interactive music practices.

Keywords: Computer Music, Interaction, Computation, Music System, Composition, Improvisation

(blank page)

Resumo

As novas abordagens aos meios digitais levaram à criação de técnicas e processos que procuram solucionar as mais variadas tarefas na composição e performação. Neste contexto, o uso de computadores para a prática musical não contribuiu apenas para a representação complexa de processos musicais com um elevado grau de detalhe e repetibilidade nos domínios da síntese e processamento de sinal em tempo real, para a composição algorítmica ou análise musical. O exponencial crescimento da capacidade de armazenamento e da velocidade de processamento dos computadores, tornou-os instrumentos viáveis para a composição e performação, permitindo a extemporização de decisões musicais e a expansão da performação para além dos limites fisiológicos humanos. Neste sentido, temos assistido a um crescente desenvolvimento de aplicações dedicadas à performação musical, tais como o *Ableton Live*[1] ou o *Reason*[2]. Estas aplicações permitem a gravação, edição e reprodução de som e a sua integração com uma grande diversidade de interfaces para o controlo e parametrização de eventos musicais.

Encontramos, apesar de tudo, vários obstáculos ao tentar expandir o nosso potencial criativo para o desenvolvimento de situações musicais interactivas. Em geral, ferramentas como as acima mencionadas, condicionam o âmbito de decisão do compositor ao proporcionar soluções generalistas e pré-determinadas, reduzindo as suas possibilidades a uma mera selecção de processos e respectivos ajustes de parâmetros. Consequentemente, estas ferramentas não só conduzem a um inconsciente tecnológico, ao ocultar os processos que caracterizam a prática musical, mas também a uma escuta tecnológica, em que facilmente se percepciona o software utilizado na criação das obras.

A utilização de software programável, como o *Max/MSP*[3], *Pure Data*[4] ou *SuperCollider*[5] já permite um maior dinamismo, embora instigue modelos de interacção

---

[1] https://www.ableton.com

[2] http://www.propellerheads.se

[3] http://cycling74.com

[4] http://puredata.info

[5] http://supercollider.sourceforge.net

específicos que se tornam relevantes compreender. Os fluxos de informação das linguagens *datafow* assemelham-se à configuração de circuitos analógicos na música electrónica. Os 'instrumentos' das linguagens *Music-N* seguem o paradigma acústico, reforçando consequentemente a distinção entre composição e performação; neste sentido, a composição assume-se como a definição de possibilidades de interacção através da criação de estruturas, instrumentos e interfaces, em que todas as decisões composicionais são tomadas *a priori*, negligenciando o potencial expressivo das práticas de improvisação.

Na tentativa de colmatar estas limitações, consideramos as potencialidades de expressão musical em tempo-real e as possibilidades comunicacionais que daí advêm, reconhecendo que os meios computacionais estão ainda abertos a muitas opportunidades de investigação. Neste sentido, o presente estudo procura contribuir para o alargamento do conhecimento artístico e científico no âmbito da Informática Musical, mais concretamente no que concerne a concepção e utilização de sistemas musicais interactivos, com o fim de estabelecer metodologias que possam fornecer um processo composicional mais transparente, flexível e dinâmico.

Pretendemos então desenvolver uma metodologia que permita uma prática composicional mais adequada a este contexto, tornando os processos mais subjectivos, dinâmicos e transparentes, no sentido de facilitar as relações que se estabelecem com o computador, explorando ideias musicais de uma forma criativa, assim como melhorar o discurso que pode ser estabelecido com outros músicos, o público e o meio ambiente.

Ao direccionar esta investigação para a intersecção entre música por intermédio do computador e a interacção levantam-se questões importantes que serão abordadas no desenvolvimento desta tese. A definição dos limites da composição no contexto da música interactiva por computador é ambígua, dado que esta prática se encontra intimamente relacionada com composição algorítmica ou com design sonoro por um lado, e por outro com performação e improvisação. A identificação e compreensão de modelos, técnicas e conceitos utilizados neste contexto tecnológico torna-se fundamental na definição da abrangência do nosso estudo. Deste modo, questionamos as ferramentas que utilizamos, o modo como são usadas e quais os objectivos que levaram à sua criação. Esta ideia também se estende à natureza dos materiais utilizados, exigindo uma compreensão mais aprofundada do computador como um *medium*, e das possíveis articulações que se estabelecem entre o compositor, os resultados sonoros e respectivos processos musicais. A definição de obra musical no contexto da música interactiva entende-se como uma acção ao invés de um objecto, dada a sua natureza efémera. Esta característica aprofunda a sua relação com as práticas de música improvisada.

Com o objectivo de desenvolver estratégias mais adequadas para criação de situações interactivas por intermédio do computador são seguidas duas direcções de investigação complementares. Em primeiro lugar, levanta-se a necessidade de fazer um levantamento histórico sobre as práticas musicais interactivas no sentido de extrair os aspectos mais significativos para a música contemporânea. Pretendemos deste modo identificar possíveis soluções a desenvolver na componente prática da nossa investigação. Em segundo lugar, passamos para a aplicação prática destas soluções, através de um trabalho musical que permite a implementação e avaliação dessas hipóteses.

Reconhecendo a natureza transdisciplinar do nosso estudo, adoptamos uma metodologia orientada para a acção (*action-based research*) tal como preconizado por Coghlan e Brannick (2010). Deste modo, direccionamos a revisão literária para a compreensão geral da música interactiva em busca de estratégias e modelos que possam indiciar possíveis acções na nossa prática. Em seguida, prosseguimos com a nossa investigação desenvolvendo uma análise teórica e metodológica em paralelo com uma experimentação prática, em ciclos iterativos de acção, planeamento, implementação e avaliação.

Fazer um levantamento das possíveis articulações que podem ser estabelecidas entre o compositor e a sua obra permite identificar abordagens alternativas que têm sido levadas a cabo para mediar ideias composicionais, com a expectativa de proporcionar uma compreensão mais profunda da natureza do meio computacional. Desta forma, no primeiro capítulo, investigamos diferentes direcções apontando para a exploração de novas sonoridades e para a delegação de intenções musicais a meios mecânicos, performadores e audiência. Consagramos também neste capítulo a nossa atenção à mudança de paradigma: das obras de formato fixo para obras que tomam a noção de processo como foco central. Esta noção de processo, como o conjunto de técnicas e estratégias mentais que levam à criação da obra musical, é de particular relevância para o nosso estudo. Nesta perspectiva, abordamos diversas obras: desde as primeiras obras no seio da música experimental de John Cage, aos compositores minimais como Steve Reich, passando pelo trabalho desenvolvido por Karlheinz Stockhausen. Também relevantes, são as possibilidades de interacção que se desenvolveram a partir da música electrónica, em que a espontaneidade da performação é particularmente valorizada em prejuízo da forma, contexto em que a *Sonic Arts Union* assumiu um papel preponderante.

No segundo capítulo, a investigação centra-se na natureza computacional e nas estratégias que foram desenvolvidas no sentido de tomar partido do computador para a prática musical. Partindo da *máquina de Turing* e do conceito de algoritmo, procuramos também proporcionar um entendimento do funcionamento deste tipo de máquinas e das estratégias que se adoptaram para formalizar processos musicais, dando origem à composição assistida por computador. Deste modo, questionamos as ferramentas que utilizamos, o modo como são usadas e quais os objectivos que levaram à sua criação. Esta ideia também se estende à natureza dos materiais utilizados, exigindo uma compreensão mais aprofundada do computador como um *medium*, e das possíveis articulações que se estabelecem entre o compositor, os resultados sonoros e respectivos processos musicais.

No sentido de aprofundar as possibilidades de interacção que os computadores proporcionam, no terceiro capítulo analisamos os sistemas musicais interactivos que se desenvolveram a partir da década de 1970, com o intuito de sublinhar possíveis direcções para a nossa prática musical. Procuramos não só identificar as técnicas e estratégias utilizadas para a criação sonora em tempo-real mas também as possibilidades comunicacionais que daí advêm.

No quarto capítulo abordamos os desenvolvimentos mais recentes das artes computacionais no sentido de estabelecer uma nova perspectiva relativamente à prática musical interactiva. Considerando o meio computacional do ponto de vista creativo, em oposição à ideia do computador como uma mera ferramenta, assumimos a programação interactiva como um potencial promissor para a prática musical interactiva.

Esta investigação é sustentada por uma prática musical interactiva que se desenvolve através da criação de *Thr44*, um *framework* computacional através do qual procuramos proporcionar um contexto mais dinâmico, flexível e transparente para a criação de sistemas musicais interactivos. Em *Thr44*, relacionamos abordagens à programação interactiva, nomeadamente *o live-coding*, com o desenvolvimento prévio de um conjunto de objectos e comportamentos que podem ser articulados durante a performação. Estratégias e paradigmas, como por exemplo, a programação baseada em protótipos podem ser usados e modificados

dinamicamente (*on-the-fly*) facilitando a prototipagem de ideias, assim como o incremento de decisões composicionais no âmbito da performação.

Procuramos com esta investigação contribuir para a discussão sobre a música interactiva baseada no computador com o objectivo de proporcionar uma compreensão mais profunda destas práticas, dos seus desenvolvimento e das relações que se estabelecem com as ciências e as tecnologias, no contexto da música contemporânea. Nesta perspectiva, propomos uma alternativa às práticas dominantes, que não só afirmam a nossa impotência para interferir com a evolução tecnológica, mas também dificultam as potencialidades criativas que daí podem advir. Consequentemente, procuramos não apenas contribuir com uma reflexão teórica sobre estas práticas, mas também disponibilizamos o sistema computacional desenvolvido, de modo a promover a discussão e experimentação em torno das práticas musicais interactivas.

Palavras-chave: Informática musical, Interacção, Computação, Sistema Musical, Composição, Improvisação

(blank page)

Table of Contents

Table of Figures

(blank page)

INTRODUCTION

Computers are ubiquitous in all features of our culture and society, mediating every aspect in our lives, with a vital role on communications, economy, politics, social and artistic activities. From a compositional perspective, computers continuously expand the possibilities for the creation and manipulation of sound. These machines and their affordances provide many advantages, allowing complex representations of musical processes with a high level of detail and repeatability, in the domains of real-time audio synthesis and signal processing, algorithmic composition, score representation and musical analysis. Their ever-increasing capabilities in processing speed and storage have made them viable instruments for live electronic music by allowing the extemporisation of compositional decisions and the extension of performance attributes beyond human physiological limits.

Due to this potential, live electronic music has become a common practice and music industries have taken advantage of it by continuously developing and releasing a wide variety of products. Advanced software such as *Ableton Live*[6], *Logic*[7] or *Reason*[8] allow real-time recording, editing and playback of musical material, and facilitate the integration of a wide variety of control devices to trigger events and control digital synthesis, filtering and processing parameters, making the creation of sound easier than ever.

Within the context of interactive music, however, one finds many obstacles while attempting to extend one's creative capacity and accessing new experiences (Ryan, n.d.n.d.). By performing with these tools, one seems constrained to a limited scope of action, reduced to a mere selection and adjustment of parameters within 'pre-composed' tools. The general

---

[6] https://www.ableton.com

[7] http://www.apple.com/logic-pro

[8] http://www.propellerheads.se

purpose solutions that this kind of software provides not only lead to a technological unconsciousness[9] but also one is capable of identifying the software that is being used.

This situation is not improved by merely using more programmable software, such as *Max/MSP*[10], *Pure Data*[11] or *SuperCollider;*[12] as we will see, in spite of the fact that these tools allow more flexibility, they still prescribe models of interaction, namely the *dataflow* patching logic or the 'acoustic model'. The latter, for instance, reinforces the separation of composition from performance. In this view, composition is regarded as making provision for interaction, as the creation of a sound machine, preparing structures, instruments and interfaces, where as all the decisions to be carried out in performance are made beforehand, neglecting the communicative potential of real-time musical expression.

Purpose

As the focus of this work turns towards the dynamic unfolding of sound and further explorations of communicational possibilities, we recognise that the computational media is still rather recent and open to many directions, as of yet to be investigated.

Therefore, this research is developed around the possibilities of computer-based interactive music practice in order to access how it can be a more subjective, dynamic and transparent process. This work seeks to improve the relationships that are established with the computer not only as a means to mediate and creatively explore musical ideas, but also to improve the discourse that can be established with other musicians, the audience and the environment.

Researching at the intersection of computer music and interaction withholds a variety of questions.

[9] Wishart refers to a 'culture of neophilia' (Wishart, 2009).

[10] http://cycling74.com

[11] http://puredata.info

[12] http://supercollider.sourceforge.net

One direction of enquiry is the understanding of which models, techniques and concepts are fundamental in this technological context. This obliges the questioning of what tools we use, how they are used, and the purpose for which they were created. This also concerns the nature of the materials used, requiring a deeper understanding of the computer as a medium, and the possible articulations between the composer and the sounds and processes one relates to.

Another direction is the definition of a musical work, given the ephemeral nature of interactive music, which strongly links to improvised music practices. The notion of work inherits a certain operational openness, a certain liveliness that implies regarding this practice as activity rather than an artefact. Furthermore, in this context, it becomes difficult to define the boundaries of composition itself as, at one end, it encompasses algorithmic composition and sound design, and at the other, it relates to performance and improvisation.

Strategy

In order to investigate the possibilities of computer-based interactive systems, in search of more adequate approaches to our practice, we find the necessity of following two complementary research paths. Not only do we need to survey the developments of interactive music creation throughout history and extract the significant aspects that can contribute to contemporary music, but also, we must formulate, experiment and develop on possible solutions that can be suitable for our practice. Therefore, we find the need to develop a computational work where we can implement and evaluate these hypotheses, making this research practice oriented, adopting an action-based methodology (Coghlan & Brannick, 2010), guided by the above mentioned directions, due to the trans-disciplinary nature of our study.

Accordingly, we begin this research by seeking a general understanding of interactive music by surveying the topic in an exploratory manner in search of approaches and models that can direct us towards possible actions for our practice. We then proceed our research by concurrently developing theoretical and methodological analysis and empirical experimentation, in iterative cycles of action, planning, implementation and evaluation.

We begin by surveying the possible articulations that can be established between the composer and his work. Questioning the fact that a diversity of commercial software enclosures its inner processes, we attempt to identify alternate approaches that have been carried out to mediate compositional ideas, with the expectancy to provide a deeper understanding of the nature of the computational medium.

One aspect that interests us is the impact of technological and scientific developments in musical practice. This not only allowed the exploration of new sounds, namely the advent of recording and reproducing sound, but also the delegation of compositional decisions to mechanical devices, through the creation of *automata*.

In view of this, the notion of *process* is of particular relevance for our study. Such notion in music is often associated to the experimental practices by John Cage, the minimal composers such as Steve Reich or the compositional work by Karlheinz Stockhausen.

We are also concerned with the possibilities for interaction that evolved from the live electronic music practice carried out by the Sonic Arts Union, for instance, in which performance spontaneity is valued over form.

To complement this enquiry path, we look into the computer from a technical perspective and attempt to understand how it operates. From the *Turing Machine* and the concept of *algorithm*, we analyse the strategies that composers carried out in order to formalise musical ideas in this medium.

Turning back to the prospects of interaction we investigate the dedicated computational tools that have been created since the 1970s in order to outline possible directions for our musical practice. We are particularly concerned with improving the dynamic unfolding of sounds so it is not a mere execution of prior compositional creation. We not only intend to underpin technical aspects, but we are also concerned with the potential of discourse they afford.

Improving interactive music practice is only possible through action, through planning and implementing ideas that can return some satisfying result within our musical activity. Therefore, complementing the above mentioned explorations we engage in a compositional activity by developing tools that can enhance the relationships that are established with the computer not only as a means to mediate and creatively explore musical ideas, but also to improve the discourse that can be established with other musicians, the audience and the environment.

Significance

This research aims to contribute to the discussion on interactive music, namely the techniques, tools and materials that are used, in order to provide a deeper understanding of these practices.

We promote a growing awareness of how electronic music practices have historically evolved and re-appropriated knowledge from a diversity of fields in ways that significantly affected the contemporary music context.

In this perspective, we propose alternatives to the dominating discourses, which not only assert our powerlessness to interfere with technological developments, but also hinder the creative potentialities that may evolve from them.

Consequently, we not only provide a theoretical ground for such practices, but also advance an implementation of a computational system that is open to further discussion and experimentation.

Overview

The first chapter is dedicated to an overview of distinct approaches that composers have carried out in order to mediate musical ideas and create musical works. We attempt to outline the most significant events and paradigms that contributed to the emergence of interactive music. We investigate the efforts into the exploration of new sounds and the delegation of musical intentions to mechanical devices, performers and the audience. We also dedicate our concerns to the shift from fixed-form works to ones that emphasize the notion of process as their core subject.

In chapter two, we focus on the nature of computers and the strategies that were carried out in order to take advantage of these machines for musical practice. From Turing's blueprint of a computer, and the concept of *algorithm*, we not only provide an understanding of the inner workings of these symbolic operators, addressing the emergence of computer-assisted composition, but also emphasize the notion of the computer as a fully realised machine, capable of generating sound from an abstract description.

An overview on computer-based interactive music practice is presented in chapter three, revealing the most significant events and paradigms that characterize it. Concerned with the communicative potential computers provide, we briefly examine distinct approaches to this practice, attempting to provide a wider perspective on the concept of interaction within this context.

In the fourth chapter, we outline recent developments in computer-based arts and attempt to develop a distinct view on interactive music practice. Reconsidering the computational medium from an aesthetic perspective, in opposition to the notion of computer as a tool, we look into the prospects of *live coding* and identify interactive programming as a promising potential for interactive music practice.

The final chapter is dedicated to the description of the computational system we have developed as a result of this research. We present the context in which it was created, and the criteria that guided its implementation. Afterwards, we present a detailed description of some of the most significant components that constitute it, finalizing with an evaluation of the work.

We finally conclude this work with final considerations on the topic of composing interactive music systems, and drawing possibilities for future research.

1 TOWARDS INTERACTIVE MUSIC

Concepts and ideas related to the emergence of technology-mediated interactive music are underpinned by establishing the relationship between music, science and technology. We are particularly interested in the interdependence between sound, procedures and technology and their contribute to the performance oriented practices that are found today.

In this perspective, we approach the early developments of procedural music identifying some of the techniques that are still present in computer music practice, such as combinatorics and chance procedures and the first attempts to automate and mechanize compositional ideas. Then we will analyse the early approaches into the articulation of sounds that were pioneered by Luigi Russolo and Edgard Varèse, extending the palette of sounds throught mechanical devices, and opening the way to the establishment of electro-acoustic music, such as the *Musique Concréte* by Pierre Schaeffer and the *Elektronische Musik* in Cologne. Afterwards, we will explore how the experiments on indeterminism led to the delegation of compositional decisions to devices and performers, prefiguring interactive music approaches.

1.1 Early Procedures

As far back as in Classical Antiquity music has been associated with the sciences and regarded as the purest form of expression.[13] The arithmetic principle of Pythagoras, which related pitch to the length of strings, evolved towards an aesthetic philosophy that closely linked art, science and nature. Although no writings remain from Ancient Greek culture, its theories arrived to our days through the Quadrivium, credited to Cassiodorus (6th Century), which was taught until the Renaissance period, and comprised Music, along with Geometry, Arithmetic and Astronomy, the four subjects of mathematical science, as they all dealt with "abstract quantity". In this context music is susceptible to be treated formally and procedurally, like mathematics.

One of the earliest procedural examples of music composition can be traced back to the 9th Century in Europe. The *Organum* is described as a method for improvising a second voice of a Gregorian chant through parallel intervals (Essl, 2007:109). Also, in the 11th Century, Guido d'Arezzo presented in *Micrologus* a method for creating musical phrases from religious hymns (Miranda, 2002:VI). By assigning a set of notes of a scale to each vowel of the text, one could generate a series of possible phrases to develop a music composition (Palisca, 2001).

Other examples that emerged in the late Middle Ages are the *canon*, sets of formal rules for the generation of voices by derivation of an initial voice. One of such examples is the isorhythm technique, credited to Philippe de Vitry, that allowed the coupling of phrases to serially repeating rhythmic patterns. Towards the Enlightment Age, the usage of these approaches became more usual. Musical Acrostics, which relate to Arezzo's method, were widely in the Western tradition, namely J.S.Bach's *Art of the Fugue*, Ludwig van Beethoven's *Opus 59*, Robert Schumann's *Abegg Variations*, among others. This method consists of

[13] As Flusser refers, "at highly inspired moments [the Greeks] spoke of *musike kay mathematike techne* as the means of attaining wisdom" (Flusser, 2011:29).

encoding words, usually the composer's name, into musical phrases by relating text characters to notes. These examples already suggest how the mere execution of rule-based techniques can yeld relevant and diversified results for music composition.

The employment of chance operations can be found in compositional techniques from the Enlightment Age, such as *Musikalisches Würfelspiel* credited to W.A.Mozart, and *Einfall einin doppelten Contrapunct in der Octave von sechs Tacten zu machen ohne die Regeln davon zu wissen*[14] (1758) by C.P.E. Bach. The compositions' notation to be performed would be created by iteratively mapping the outcome of throwing dice, or spinners, to several possible musical material from tables of musical figures (Loy, 1989:298). The notion of developing formal systems or devices for cognitive reasoning was already evident in the Enlightment Age, although their artistic relevance was questioned. Chance games were very appreciated, in spite of being regarded as mere entertainment, since composition was viewed as a practice that can only be achievable by human intellect. On a 19th Century edition of Arezzo's *Micrologus*, for instance, Hermesdorff justifies such methods as "help provided for the incompetent beginner" (Loy, 1989:298), revealing a derided view on the usage of procedural methods. Nevertheless, such constructions not only evidence the possibility of devices that mediate compositional thought, but also connect to the work of John Cage and Iannis Xenakis who extensively explored probabilistic methods.

As early as the 17th Century, the attempt to register compositional procedures on media begins to emerge, and many processing machines and devices were created, long before the invention of mechanical computers. Athanasius Kircher's *Musurgia Universalis*, cited as the precursor of computer-generated music (Cramer, 2005:106) describes the design of one such attempt. The *Arca Musarithmica* (Fig. 1) is a "collection of pre-composed musical, poetic, and rhetorical patterns from where it was possible to create variable, harmonic compositions" (Carvalhais, 2010:107). This conception is based on the work of Ramon Lull, a 13th Century Catalan monk who claimed to have had a divine revelation of a formal system for composing and deriving philosophical-theological statements (Cramer, 2005).

[14] A method for making six bars of double counterpoint at the octave without knowing the rules.

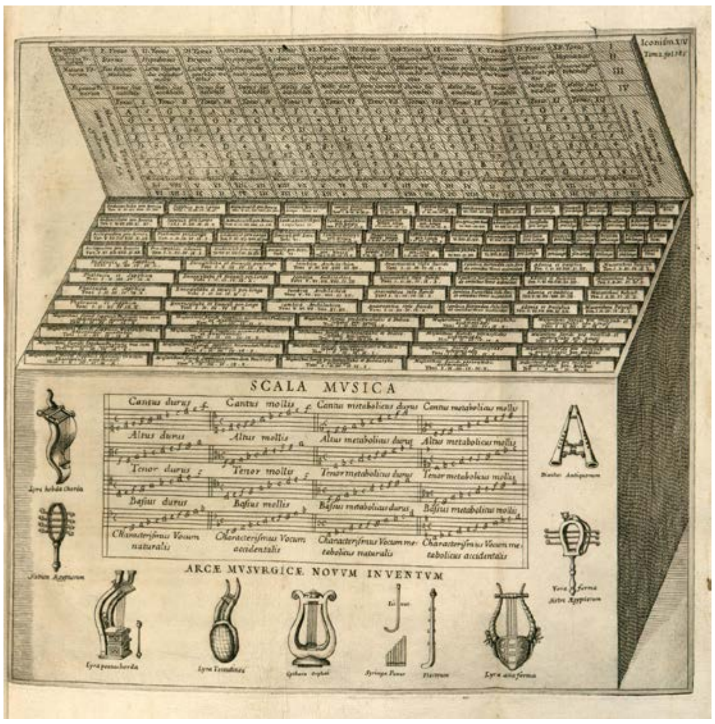


Fig. 1 Arca Musarithmica (Zielinski, 2006:146)

Probably influenced by kabbalistic thinking, as expressed in *Sefer Yatzirah* (Book of Creation), the *Circles of Lull* were oracles that consisted of several rotating paper discs with inscribed symbols that allowed combinatorial operations between distinct sets of materials (Essl, 2007:110). Cramer refers to Quirinus Kuhlmann's published correspondence with Kircher, where the former found in the combinatorics of Lull a key to unlock the secret of the *tree of life*. We notice in Kuhlmann the suggestion of some sort of artificial life, a delicate subject for their religious context. For this reason, and alerting to the risk of heresy, Kircher suggested a strictly technical perspective on the possibilities of combinatorics.

A major attempt to create a mechanized non-human computer is credited to Charles Babbage, who conceived the *Analytical Engine*, in 1837. Babbage also theorized on a similar machine for music composition, as described by Ada Lovelace:

> [The Engine's] operating mechanism might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations…. Supposing, for instance, that the fundamental relations of pitched sound in the signs of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.
> (Lovelace qtd. in Mathews & Pierce, 1989:318)

Programmable music-machines began to appear in the following years, such as carillons, music-boxes and mechanical organs. The *player-piano*, for instance, could play by itself patterns that were perfurated in paper rolls. In fact, these devices were the precursors of recording technologies and composers such as Percy Grainger[15] and Colon Nancarrow explored their potential for the realisation of works that extended far beyond human capabilities.

1.2 Formal approaches to Composition

The turn of the XXth Century encompassed a series of deep scientific and cultural changes that were expressed on the structuring of art forms. The formalization and mechanization of processes gained strength with the new scientific discoveries made.

Prior laws of physics, as formulated in a traditional way, describe an idealized stable world of closed, single, fixed hierarchy of "preordained orders" (Eco, 1989:13). The developments brought by Albert Einstein, Henry Poincaré and Werner Heisenberg introduced new formulations that adopt a view of the world as a complex, dynamic and multidimensional system, deeply influencing compositional practice. Einstein's *Theory of Relativity* emphasizes that the laws of physics can only be established with respect to an observer; the *Uncertainty Principle*, by Heisenberg, describes the impossibility of simultaneously determining position and velocity of certain particles; in turn, Poincaré discovers processes that are extremely sensitive to initial conditions, where changing them produces an exponential change. As Friedman and Donley point out, since the Renaissance there had never been so many artists aware of the developments of other fields of knowledge, in particular, the fields of science. This transdisciplinarity opened up a vast scope of unexplored possibilities within artistic practice (Friedman & Donley, 1985:2). The influence of these scientific discoveries, as Chadabe argues, brought an opening up of ideas in early XXth Century compositional practice, namely, the music from Claude Debussy, Igor Stravinsky or Charles Ives that already presented complex combinations and superimpositions of rhythms and melodies in a multiplicity of new combinations (Chadabe, 1997:22). Describing how Ives attempted to represent an interest in uncontrolled, naturally

[15] Hugill (2007:20) provides details on Percy Grainger' s work.

occurring events in *Symphony No.4* (1916), Perkins describes how science contributed to a redefinition of the composer's role in this period:

> "The role of the composer is in a sense more passive than that of a romantic composer: once set in motion, the music has its own life and the composer is a listener like any other. Calling this music experimental is quite precisely correct: like a scientist setting up an experiment, the experimental composer sets up the conditions that define the piece, and is interested in hearing what actually happens with it." (Perkins, 2002)

These new scientific logics can also be regarded in the rigorous application of algorithmic procedures for music composition found in the twelve-tone formalism, developed by Arnold Schoenberg in the 1920s. As an attempt to redefine compositional practice, his approach was regarded as revolutionary as Einstein's Relativity theory was for physics and mathematics. His notion of "beauty" is closely related to the ability to comprehend musical form. "Music is not merely another kind of amusement but a […] thinker's representation of ideas [that] must correspond to the laws of human logic" (1975:220) Schoenberg rejected the use of tonality or any intervals that reminded tonality to avoid perceived attractions of a melodic trajectory to the tonic and thus proposed a formal method for composition where all pitches are equally relevant. This set of procedures consists in the use of all the twelve notes of the chromatic scale in a consistent pattern defined as a row, set, or series, where no note would appear more than once. Formal techniques to operate with such sets would be used, such as the inversion, retrograde, retrograde-inversion and transposition.

The rigorous application of logic in musical creation was also advocated by Joseph Schillinger, who was a mathematician. He asserted that "Analysis of aesthetic form requires mathematical techniques, and the synthesis of forms […] requires the technique of engineering". He refuted all previous art theory characterizing it as a mere "imitation of appearances" that failed in the analysis and synthesis of art", and proposes a method of "creation from principles", suggesting the complete formalization[16] not only of artistic technique, but also of creativity.

[16] In the following chapter we will discuss David Hilbert's interest in the complete formalization of mathematical axioms, and how this concept evolved towards the invention of the computer's blueprint: the *Turing machine*.

Another approach towards the redefinition of compositional theory is developed by Messiaen, who, as Xenakis argues, "took a great step in systematizing the abstraction of all variables of instrumental music" (Xenakis, 1992:5). Instead of removing the tonal function of chords, Messiaen superimposes a diversity of functions creating a multi-modal music.

1.3 Material approaches to Composition

As previously regarded, the beginning of the XX^th^ Century documents a general dissatisfaction with the direction of traditional music, from which new approaches to musical form emerge.

In such context, a new approach that evolved was to regard sound material as a semantic unit for music composition. Timbre, traditionally treated as an inherited property from a given orchestration set, or at the most, "a matter of colourisation of musical structure" (Thoresen, 2007), finds new possibilities with the technological developments of that period, such as the advent of sound recording and the development of electronic instruments.

The recording of sounds was made possible with Thomas Edison's experimental inscriptions of vibrations on wax, tinfoil or lead cylinders (Holmes, 2008:33) in 1870, and subsequent proliferation of mechanical recorders and playback machines[17]. The potential to reproduce sound through technological means brought a perspective of autonomy to compositional practice that had always been subjected to an indirect mediated practice of notating a work to only later be materialized by performers. Composers such as Ferrucio Busoni, Luigi Russolo or Edgard Varèse had realized that machines could be a means to fulfil compositional practice in an almost unmediated way. Their pioneering work created the conditions for the emergence of *Musique Concréte* and *Elektronische Musik* and subsequent developments that we inherit in today's compositional practice.

[17] In 1977, Edison invented the first commercially available microphone. In the same year he invented the phonograph, although Charles Cros came up with the same idea independently. In 1887, Emile Berliner patented the gramophone, using flat discs rather than cylinders (Hugill, 2007:14).

Ferrucio Busoni is cited as one of the first composers to consider technology as a means to develop musical ideas (Holmes, 2008:12). On *Sketch of a New Aesthetic of Music*, Ferruccio Busoni wrote "Music was born free; and to win freedom is its destiny" (Busoni, 1911), foreseeing the potential of machines for new directions in musical practice, influencing Varèse and the artists that became known as the Futurists.

While Busoni found in machines an opportunity to expand musical ideas, Luigi Russolo found the possibility of new sounds. In 1913, Russolo proclaimed the birth of a new art form through his manifesto *The Art of Noise*. Russolo proposed that "new music could be based on turning the noises of the world into music" (Cascone, 2000) and that instruments should be replaced by machines, providing an enriched sonic diversity.

> The variety of noises is infinite. If today, when we have perhaps a thousand different machines, we can distinguish a thousand different noises, tomorrow, as new machines multiply, we will be able to distinguish ten, twenty, or thirty thousand different noises, not merely in a simply imitative way, but to combine them according to our imagination. (Russolo qtd. in Cascone, 2000)

For Russolo, noise is musical and can be organized musically. He categorized the distinct sounds generated by the noise constructions (intunarumori) he developed with Ugo Piatti. Focused on the inner properties of noises, "he provided descriptive names, such as 'exploders', 'roarers', 'croakers', 'thunderers', 'bursters', 'cracklers', 'buzzers', and 'scrapers' (Cascone, 2000) foreseeing Raymond Schafer's sound classification (Schafer, 1979).

Edgard Varèse also demarked himself from the note tradition reiterating that composers continue clung to traditions that are nothing but limitations to compositional possibilities. Varèse describes his music as "organized sound", reinforcing the notion that he is "a worker in rhythms, frequencies, and intensities" (Varèse & Wen-chung, 1966:18). It is important to note that though he refers to parameters such as frequencies and intensities, for Varèse, the "raw material" of his work is 'sound'. His compositional work develops from the organization of timbre and rhythm. Like the futurists, he was also interested in enriching the musical alphabet by extending the scope of sonic possibilities, though he criticises their approach to be an imitation of "superficial and boring" sounds of urban life, advocating that "a new world of unsuspected sounds" could be developed. One must consider, nonetheless, Risset's argument that the idea of novelty of sounds is a question of perception, and that quickly they become ordinary[18].

Varèse is particularly relevant as he was one of the first composers to conceptualize the implications of mechanized means for composition and performance. Describing music as an *art-science*, he considers the instruments immediacy in relation to the composer's thought, and anticipates the advantages of technological developments to help the composer realize what had never before been possible. With such machinery, the ideas of the composer would "no longer be desecrated by adaptation or performance as all the past classics were" (Holmes, 2008:5); one would be able to obtain any number of cycles, any subdivision of them, omission or fraction of them – all these in a given unit of measure or time that is humanly impossible to attain (Varèse & Wen-chung, 1966:13). Varèse had to wait many years before he found the means to develop his compositional ideas. His only tape music, *Poéme electronique* (1958) was created a few years before his death.

1.3.1 Musique Concrète

Developing on precedent considerations of using machines for musical expression, Pierre Schaeffer's *Musique Concréte* evolved from his experiments with radiophonic equipment, and represents an operational change in composing music by direct manipulation of sound through the recording medium.

The use of recording media as a productive tool, such as the gramophone, had already been considered, namely by Apollinaire (Battier, 2007), by Lázlo Moholy-Nagy who along with Oskar Fishchnger and Paul Arma physically manipulated records to generate new sonic material (Manning, 2003), by experimental filmmaker Walter Ruttmann, who created *Weekend*, a sound collage on an optical soundtrack in 1930, by Hindemith and Torch (1935) and by John Cage for his composition *Imaginary Landscape nº1* (1939).

However, Schaeffer's interest was in the actual possibility of composing with sound. As Pierre Henry notes, the focus was on "‘plastifying’ music, of rendering it plastic like sculpture", formulating "a new mental framework of composing" (Pierre Henry qtd. in

[18] Personal presentation at Universidade Católica do Porto 2008/09/10.

James, 1981:79)[19]. While researching on the possibilities of *Musique Concrète*, Schaeffer introduced the notion of the 'sound object', which he developed in subsequent years, resulting in the publication of *Traité des Objets Musicaux* (Schaeffer, 1966). Regarding this work, Michel Chion exposes that the motivations that led to the notion of the 'sound object' were due to the discoveries made by early experiences carried out with the gramophone and the recognition of the status of recorded sounds, as dissociated from their cause, hence, acousmatic (Chion, 1983).

Schaeffer recognizes the multiple possibilities of describing sound. At one end the scientific manner, where sound is characterized by its acoustic and mathematical properties, or the aesthetic manner, where it is characterized for its psychosocial characteristics [20], and the composer's subjective formulations of value. For Schaeffer, phenomenological studies could not explain the richness of how humans perceive sound, requiring a music theory to sustain this practice. As Collins reminds us, a complete understanding of human perception is a worthy aim for the cognitive science of music (Collins, 2006:8). As such, Schaeffer proposed "reduced listening" as a framework to operate with sounds. His reduced listening consists of understanding sound as a signifier on its own, reduced to its intrinsic attributes, its materiality, its substance. The experience of sound transcends the object itself and enables a unique formalization over sound material. Thus, the 'sound object' is "the meeting point of an acoustic action and a listening intention" (Schaeffer, 1966).

The theoretical work developed by Schaeffer echoes through time in a diversity of readings and interpretations, such as the work on Spectromorphology by Dennis Smalley (Smalley, 1994), or the appropriations of the 'sound object' in computational media by William Buxton (Buxton, Reeves, Baecker, & Mezei, 1978) and Horaccio Vaggione (Budón, 2000; Vaggione, 1991)[21]. His work reminds us that technology does not define what is perceptually important in music.

[19] See also *Guide to Sound Objects* (Chion, 1983:77).

[20] An empiric approach that values sounds as they are.

[21] See also *L'Object Sonore: Situation, Évaluation et Potentialités* (Dias, 2005).

1.3.2 Elektronische Musik

A distinct approach was carried out by Werner Meyer-Eppler, Robert Beyer, and Herbert Eimert in WDR Studios, known as *Elektronische Musik*. Their purpose was to extend the serialist tradition by synthesizing music with electronic devices that could easily be mapped to parameters such as pitch and amplitude, and extended to the control of timbre, dynamics, and densities.

In the first years of WDR, the sonic possibilities of the equipment were very limited, consisting of a single sine wave generator, a white noise generator and a Monochord. In the course of a few years, due to the collaboration with composers such as Koenig, Stockhausen, Pousseur, among others the WDR Studios offered a wide variety of devices to generate rich sounds.

Risset makes a clear distinction between *Musique Concrète* and *Elektronische Musik* by stating that the former, taking Russolo's direction, incorporated sounds of all kinds, offering a richness of sound constrained by the rudimentary means to control it. The latter, in contradistinction, offered means at the cost of 'simple and dull sounds' (Risset 1976).

Nonetheless, the incompatibility between *Musique Concréte* and *Elektronische Musik* remained, pointing to limitations within machines to incorporate certain compositional principles, which we will resume later (cf. Ch.2.2.2).

Studios like WDR, GRM expanded throughout the world, promoting the establishment of electronic and electroacoustic music practice and providing a context to support compositional practice. The relation between composers and scientific researchers not only enabled the development of new tools and theories, such as *Musique Concréte* and *Elektronische Musik*, but also created an interdisciplinary context that allowed the continuous research and experimentation of musical practice. Many researchers working in these studios, such as Abraham Moles, Jacques Poullin, Werner Meyer-Eppler or Fritz Enkel, facilitated composers with the most recent scientific studies. As such, quantum theory developments, system theory and cybernetics[22] became known and appropriated, enabling musical developments in years to come, namely the process-based approaches by John Cage and Iannis Xenakis that we will hereafter discuss.

1.4 Process-based composition

From the mid 1950s, the influence of science in compositional practice becomes more evident, particularly concerning the means to structure music works. Instead of composing "finished works" by directly manipulating the musical materials, following a structure or any other predefined constraints, the focus turns to composing dynamic processes. These processes are the composition *per se*, a system that, when enacted, results in a multiplicity of possible outcomes of the music work, determined by rules or delegated to performance through the action of human agents, devices or the environment.

The shift from structure-based to process-based composition introduces the possibility of interaction and provides us with a diversity of approaches that are relevant to our study.

1.4.1 Autonomous music

The development of process-based works entirely determined by probabilistic rules was pioneered by John Cage and Iannis Xenakis in the mid 1950s. Although the final outcome is presented as a fixed and finished work, indeterminism played a defining role in the act of composing. Their works not only are examples of how compositional practice may be formalized in order to be carried out by computers, anticipating algorithmic composition, but also of the diversity of directions that can be taken by introducing elements that are extraneous to musical practice.

[22] Norbert Wiener's theory of Cybernetics (Wiener, 1948) grew out of interdisciplinary meetings that took place between 1944 and 1953, known as the Macy Conferences. Wiener proposed a model for integrating machines into our sensory experiences and creative processes. The existing automata, coupled to the outside world, were equipped with sensing devices, and the equivalent of a nervous system from living systems. Wiener recognized the negative feedback loops for the regulation of behaviour that were characteristic to both living systems and machines and conceived their integration through the transfer of information from the one to the other.

Chance Procedures

Music of Changes (1950-51) from John Cage was the first work wholly generated by chance operations (Cage 1973:57). Resorting to a chart system with 8x8 cells which corresponded to the *I-Ching* hexagram. Cage described sounds systematically arranged in charts and used the *I-Ching* to locate them. His compositional procedures developed into using multiple charts, defining components of sound allowing situations that were not preconceived. Each event was created through the usage of three charts for sound, durations, and dynamics, assuring that Cage would not impose his own intentions on the final form that made up the *Music of Changes*. The sound charts contained sound complexes that were intentionally created, and chance-determined manipulation of durations and dynamics.

Whereas Xenakis employed random operations in order to gain control over the musical structure, Cage's work attempted the opposite and questioned the status of the author by attempting to remove the composer from making decisions, and instead, delegating them to chance. His view was criticized by Pierre Boulez, who believed in the active role for composition and rejected the widespread usage of chance and loss of the opportunity to control events.

> "The most elementary form of the transmutation of chance would lie in the adoption of a philosophy tinged with Orientalism that masks a basic weakness in compositional technique; it would be a protection against the asphyxia of invention the individual does not feel responsible for his work, but merely throws himself by unadmitted weakness, by confusion, and for temporary assuagement into puerile magic."
> (Boulez, Noakes, & Jacobs, 1964)

His point of view by defining such approach as 'chance through inadvertence', as an absence of control through "impotence", and by further characterizing 'chance by automatism' as a "sterile search for combinative devices, into an aggressive refusal of arbitrariness" (Boulez et al., 1964).

However, for this work, Cage's intentions were to remove the authoritarian role of the composer through chance, liberating sound from its referent.

"The music of Changes is an object more inhuman than human, since chance operations brought it into being. The fact that these things that constitute it, though only sounds have come together to control a human being, the performer, gives the work the alarming aspect of a Frankenstein monster. The situation is of course characteristic of Western Music, the masterpieces of which are its most frightening examples, which then concerned with humane communication only move over from Frankenstein monster to Dictator."
(Cage, 1968:36)

Nevertheless, as Robinson points out, Cage regretted the fact that in *Music of Changes* he did not take chance beyond the "sphere of the compositing phase" (Robinson, Bois, Kotz, & Joseph, 2009:90), a direction we'll further discuss in this chapter.

Stochastics

Contrary to Cage's intentions to liberate the composer through chance, Xenakis's aim is to control it. Xenakis believed that *serialism* was not a proper alternative to tonal music. By assigning equal relevance to all pitches serialists removed causality and remained with a complexity where one hears "nothing but a mass of notes in various registers [,] … an irrational and fortuous dispersion of sounds over the extent of the whole spectrum" (Xenakis, 1992:8). *Stochastics*, used by science to explain the complexities of natural phenomenon, were regarded by Xenakis as the adequate tools to control the orchestral sounds of a musical work, rather than the "linear polyphony" used by serialism. The stochastic methods from Iannis Xenakis emerged as an aesthetic that integrates mathematics, logic and sciences into a theory for music composition.

In order to develop his theory, Xenakis needed to devise "the minimum of logical constraints necessary for the construction of a musical process", for which he resorted to the usage of algebra. Sonic events were abstracted through vectors, which can have as many dimensions as the amount of parameters to model. Such formulation enabled him to create n^{th} order vector spaces (1992:155) where each axis would be mapped to a parameter. The scores for *Metastasis (*1955) and *Pithoprakta (*1954) are graphs where the drawn lines represent the evolution of pitch through time. Rather than discrete pitches, Xenakis uses long glissandi to obtain "sonic spaces of continuous evolution".

Xenakis proceeded with the development of rules to control sonic events. For *Pithoprakta* he used the kinetic theory of gases to determine the velocity of each particle, represented as the slope of each line ("temperature"). The average mean of all velocities was distributed according to *Bernoulli's Law of Large Numbers*.

The compositional process of *Achorripsis* (1957) is similar, although for this work every musical element is stochastically determined using *Poisson's* formula – the overall form, durations, dynamics and pitch.[23]

Taking advantage of mechanical devices, such as a tape recorder, Xenakis worked at *GRM*, developing electroacoustic works like *ConcretPH* (1958), *Diamorphoses* (1957-58), *Orient-Occident* (1960) and *Bohor* (1962). Xenakis was not interested in Schaeffer's research agenda, but rather in developing his compositional theory (Harley, 2004:20).

Xenakis adopted Dennis Gabor's *sound quanta theory* (Gabor, 1947) to represent acoustic signals. Rather than regarding sound as a function of time (waveform) or sum of functions with rigorously defined frequencies, as defined by Fourier, Gabor conceived that "any sound can be decomposed into a family of functions obtained by time and frequency shifts of a single gaussian particle"(Roads, 2004:57). Xenakis describes "all sound is an integration of grains, of elementary sonic particles, of sonic quanta." Continuous sonic variation is conceived as an assemblage of a large number of grains disposed in time. These grains can be described in terms of duration, frequency and dynamics.

For *ConcretPH*, Xenakis recorded on tape sounds of burning material. Then, very short chunks were extracted and isolated into single creaks. These segments had the duration of hundredths or thousandths of a second. This material was then assembled to create distinct textures that were then used to form the work.

[23] Iannis Xenakis' process is thoroughly described in *Formalized Music* (Xenakis, 1992:29-38). Further analysis are provided by Linda Arsenault (Arsenault, 2002) and by Agostino Di Scipio (Di Scipio, 1998).

1.4.2 Heteronomous works

Open systems

So far we have discussed the exploration of indeterminism as devised by mathematics, where the outcome of a musical work is expectable within the scope of the possibilities defined by the composer.

The emergence of unforeseen situations can be achieved through the exploration of media serendipity by incorporating uncontrollable aspects of a specific medium as compositional material. *Imaginary Landscape nº4 (*1951) and *4'33 (*1952), by John Cage, are examples of such approach. The unpredictable sounds of broadcasting radio stations are the raw material for *Imaginary Landscape nº4*, a piece conceived of 12 radio receivers, operated by 24 performers (2 for each radio) with detailed instructions for each to modulate their volume and tuning. The score of *4'33"* does not contain any note to be performed. Rather, the sounds made by the audience and surrounding environment are the content of this work, inviting the audience to experience the rich sonic environment that surrounds the performance. Cage's statement – "There's no such thing as silence" – refers to his experience in an anechoic chamber which allowed him to become aware of other less perceptible sounds, such as the sound of the nerve's systematic operation or blood's circulation (Cage, 1968:8). This work can be interpreted as an attempt to create a situation of awareness of the rich sonic environment that surrounds the performance.

Possibilities for interaction

Possibilities for the performers to interact with the musical work became evident in the late 1950s with the works of european composers such as Karlheinz Stockhausen, Pierre Boulez or Henri Pousseur. These works are cited by Umberto Eco, who proposes the term 'open work' in order to describe the then emergent approach to deliberately leave a certain autonomy of choice to the performer, as components of a construction kit, in which the works 'are brought to their conclusion by the performer at the same time as he experiences them on an aesthetic plane' (Eco, 1989:169).

In *KlavierStück XI* (1956) by Karlheinz Stockhausen, the score contains nineteen sections that the performer can successively choose to perform, and provides a set of rules: At the end of each section there is a tempo, agogic or dynamics instruction to be carried out in the following section that the performer chooses; in the second iteration of a section, the octave indications notated must be played; the third repetition of a section results in the end of the performance.

In a similar direction, in Pierre Boulez's *Third Sonata for Piano* (1958), the first section consists of ten scored sections that the performer can rearrange.

Another significant work cited by Eco is Henri Pousseur's electronic composition *Scambi* (1957). This work is unusual in the tape-music medium because it is explicitly meant to be assembled in different ways before listening. Pousseur describes this work "not so much a musical composition [but] a *field of possibilities*" (Pousseur qdt. in Eco, 1989:1-2). It consists of sixteen sections, where each can be linked to two, allowing superpositions, presenting a rich example of the usage of combinatorics.

Eco’s work offered a significant conceptual ground, and was an inspiration for the exploration of indeterminate and procedural approaches to music. As theorized by Eco, a work of art gains “aesthetic validity” with the possibility of having a multitude of perspectives from a single work In the works cited by Eco, however, very narrow possibilities are given to the performers, merely being allowed to define the sequence of events through combinatorics.

Expanding further on the potential of combinatorics, as found in 'open works', and on possibilities for interaction in a performative context, we can refer to *Duel* (1958), a composition by Iannis Xenakis that takes the form of a conflict between two opposing orchestras. For such, Xenakis created probabilistic tables that correlate "direct sonic operations [from each of the conductors that result in] payments one conductor gives the other" (Xenakis 1992:112). *Game theory* is applied to these probability tables in order to guarantee a *zero-sum game*[24]. This work also illustrates how interaction emerges from procedurality through the definition of game rules intended to be carried out during a performance.

As we have previously regarded, John Cage had also been exploring these possibilities, having proposed a wide diversity of approaches, such as the delegation of compositional decisions to chance or technological serendipity. Means to interact with performers can be found in the *Variations* (1958-1967) series. *Variations II* (1961), for

[24] The losses (or gains) of an orchestra coincide with the gains (or losses) of the other.

instance, is rather a composition kit than a score, and consists of eleven transparent sheets (six having drawings of straight lines and five of points). The sheets are to be superimposed freely and perpendiculars to the lines must be dropped, containing the points. The length of these perpendiculars represents the value of each parameter of sound. A single use of all sheets yields thirty determinations.

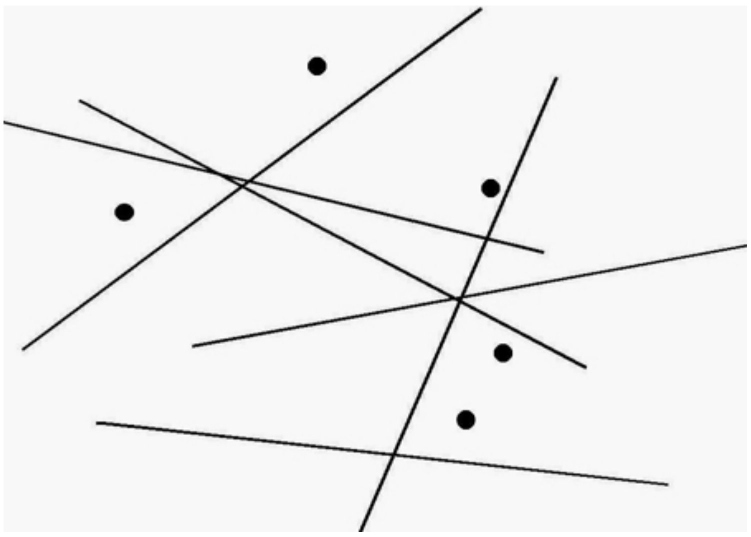


Fig. 2 John Cage, Variations II, 1961

1.5 Live Electronic Music

In the late 1950s John Cage and David Tudor started experimenting on electroacoustic music for Cunningham's Dance Company (Fig. 3), realising the need to improve the relations between sound and movement. Their experiments led to the creation of *Cartridge Music* (1960), one of Cage's earliest attempts to produce live electronic music. Similarly to *Variations II*, Its score consists of a series of transparent sheets on which patterns are drawn. The performer must superimpose the sheets and work out their time structure by observing the ways in which the lines intersect. The sounds are to be generated using cartridges from record players, manipulating them freely with other objects, only being constrained by the score's time structure.

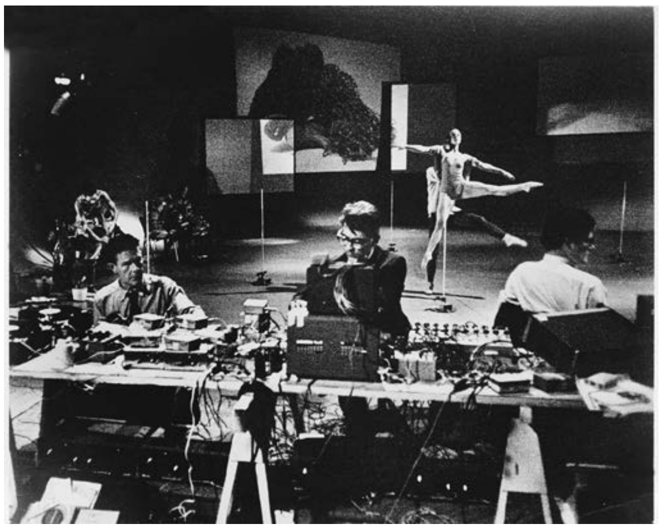


Fig. 3 John Cage, David Tudor and Gordon Mumma performing Variations V (Robinson, 2009:256).

For Cage, the idea of live electronic music was not viewed as a mere technological improvement, in which electronic sounds could be generated at the moment of performance, without resourcing to pre-recorded material, leveraging electronics to traditional instruments. Rather, music was regarded as an 'affirmation of life', reiterating the possibility for a more experimental direction in music.

> And what is the purpose of writing music? One is, of course, not dealing with purposes but dealing with sounds. Or the answer must take the form of paradox: a purposeful purposelessness or a purposeless play. This play, however, is an affirmation of life – not an attempt to bring order out of chaos nor to suggest improvements in creation, but simply a way of waking up to the very life we're living, which is so excellent once one gets one's mind and one's desires out of its way and lets it act of its own accord.
> (Cage, 1968:12)

The work developed by Cage and Tudor was deeply influential to Robert Ashley, Gordon Mumma, David Behrman and Alvin Lucier who also collaborated in the Cunningham Dance Company and later formed the Sonic Arts Union (1966-1976), exploring live electronics. As Behrman points out, their interest evolved towards doing pieces "in which established techniques were thrown away and the nature of sound was dealt with from scratch" (Holmes, 2008:376).

An example of such direction is Alvin Lucier's *I Am Sitting in a Room* (1969), a performance that explores the environment's acoustic resonance, in which Lucier himself narrates a text that is iteratively recorded and played back into the room until only a static drone can be heard.

Gordon Mumma has become a fundamental reference to interactive music practice for having adopted cybernetics ideas from Wiener (1948)[22]. Adopting the term *cybersonics* (Mumma, 1967) his compositional work consists of developing analog self-regulating circuits to interact with in performance. The 'affirmation of life' that is advocated by Cage, is interpreted as a complex of interactions, as found in Wiener's conceptualization of 'living systems'.

For *Hornpipe* (1967), Mumma modified a French horn and built a complex setting of vertical pipes and microphones to capture resonances at different frequencies, connected to a sound modifying circuit. The performer's task is to take advantage of the system's feedback loops and attempt to control it by balancing and unbalancing at different states (Mumma, 1967).

His 'cybersonics' approach to composition implies the development of electronics as a fundamental part of his work, shifting from the composition of sounds to the composition of artefacts that enable the creation of sounds.

> My decisions about electronic procedures, circuitry, and configurations are strongly influenced by the requirements of my profession as a music maker. This is one reason why I consider that my designing and building of circuits is really "composing." I am simply employing electronic technology in the achievement of my art.
> (Mumma 1967)

Having surveyed the specificity of operational modes in music composition, we have established the relationship between music, science and technology, and how this interdisciplinary practice contributed to the development of new approaches to mediate musical ideas and to the enhanced potential to explore new sounds.

From classical antiquity to the mid 1950s experiments carried out by John Cage, from combinatorics to 'stochastics', a diversity of concepts that relate to contemporary computer-based music practice have been underlined.

It is significant to note that the emergence of procedural practice in music composition is not exclusive to the use of computers but rather a subjective diversity that is inherently musical. In fact, these approaches emerged as an attempt to understand the complexities of natural phenomena and rendering them into music. Subsequently, this search contributed to the development of processes for automation, drawing a path towards today's computer-based algorithmic composition.

Another fundamental idea is related to the invention of devices for the creation of new sounds, configuring the possibility to generate sounds and enabling the expansion of the composer's palette. We have identified issues related to the apparent incompatibility between formal and material approaches to composition, a subject that we will tackle in the next chapter.

Finally, we have observed how possibilities for interaction have evolved from procedurality, making the separation between composing and performing less evident in interactive music practices.

(blank page)

2 MEDIATION WITH COMPUTERS: COMPUTABLE MUSIC

In the previous chapter, we have regarded some of the most significant processes used in musical composition. A diversity of works by composers, such as Iannis Xenakis or John Cage, evolved towards the definition of rules and constraints, the attempt to mechanize, automate and delegate musical ideas through the definition of musical procedures.

In this chapter we will observe how these principles were applied to computers, attempting to provide a deeper understanding of the developments that influence today's music practice, and in search of the limitations and possibilities they enable.

The most significant issue these pioneers had to deal with was how to incorporate and extend compositional tools and concepts with such logical machines.

We will begin by regarding formal aspects of computation, such as the *Turing Machine* in an attempt to understand the parallels that can be drawn between composing and programming, developing on the notion of musical works that take the form of computer programs with prescribed musical algorithms. Afterwards we will tackle on the developments on sound synthesis and the diversity of strategies carried out in order to make the computer become a concrete sound machine, offering the composer autonomy to execute (perform) his musical creations.

We will end this chapter with some considerations on interaction that, through the 1960s and 1970s, revealed possibilities for computers to be used in performance.

2.1 On Computable Music

The most significant property of the computer is that it can perform any task that can be defined for any formal system to do. It is a General Purpose Logical Machine. In one sense a computer can do anything. However, that statement begs the question of how we define whatever it is we want the computer to do. The computer can do anything provided that we can define precisely how it should be done.
(Edmonds, 2009:113)

The use of computers for music composition emerged in the late 1950s with the experiments by Lejaren Hiller and Leonard Isacson, Rudolf Zaripov among others (Ariza, 2005:36; Loy, 1989:47; Roads, 1999:830). Pioneering algorithmic composition, their experiments progressed as attempts to formalize musical concepts by means that could be carried out by computers in order to return a score of note parameters to be performed by instrumental musicians.

Methods for the generation of musical material were already very well known, such as combinatorics and chance procedures, however the notion of prescribing compositional ideas into a machine involved many uncertainties – from a minimal set of mathematical and logical operations, these composers had to define and program complete musical works, contributing to an understanding of contemporary computer-based practice. The concept of programming music, as of writing algorithms, inherits Alan Turing's conception of the *Turing Machine*[25], the blueprint of actual mechanical computers as available today.

[25] The *Turing Machine* was developed by Alan Turing on his paper *On Computable Numbers* (Turing, 1936). In this period, the term "computer" was widely known as a job title of a person who carried out calculations in order to answer mathematical questions (Hayles, 2005), and a machine that could perform such calculations was required. The context of the Second Great War required high amounts of complex calculations related to design and use of weapons, deciphering messages or other engineering needs.

A general problem of mathematics was to mechanically validate axioms. David Hilbert wanted to solve this issue and support mathematics with solid and complete principles, eradicating theoretical uncertainties. His objectives, known as the *Entscheidungsproblem* (decision problem) were questioned by Gödel, Turing and Church who argued that logic cannot completely prove all mathematical theorems.

The *Incompleteness Theorem* by Gödel refuted such logical inference, arguing that no axiom can mechanically be evaluated as it would recursively require to prove the validity of its principle axioms *ad infinitum*, therefore, the system would be either incomplete or false (Hofstadter, 1979). Turing replied to this particular issue questioning whether any specific mathematical problem could be solved mechanically, i.e. computable. His work introduced the ∝-*machine*, later known as *Turing Machine* (TM), a new model for non-human computing that served as a blueprint for the electronic digital computer.

The TM is a concept of a device for symbolic manipulation that consists of a 'store', an unlimited memory capacity, obtained in the form of an infinite tape (the metaphor for paper), an executive unit that operates on the 'store', according to a table of behaviours that specified in the tape, and a 'control', that guarantees that the operations where correctly performed (Turing, 1950).

On *Computable Numbers*, Turing limits his theory's numbers to a considerable subset of them, which he termed computable numbers. Such limitation can primarily be justified by the fact that the human memory is necessarily limited too. An infinite number of signs made up of a finite alphabet that, as we all know, can be reduced down to zero and one, has since then banished the endlessness of numbers. Hoffstaedter's description of Zeno's paradox (Hofstadter, 1979:39-47) elucidates this aspect of mathematics.

The tape is divided by cells that take a binary value (any finite number or symbol can be represented as a sequence of ones and zeros, a simplification of language to on's and off's). The 'executive unit' running over the tape can move backwards and forward, and read, write or erase symbols accordingly to the specified operations, changing the machine's state in discrete steps. Turing introduced the notion of logic to perform symbolic processing as part of the machine. The computer is described as a state machine that, in discrete steps, would be transformed from one state to another. It can sequentially take each cell of the symbol, perform a Boolean operation in a one-to-one relation, and write the result as another symbol.

This process characterizes how a computer is operated with machine language instructions (opcodes): We call the number of a computer specific operation, stored in the computer's central processing unit (CPU) to operate on data stored at a specific address in memory. In order to facilitate the act of programming, computer engineers developed *Assembler*, a program that allows translating these commands and data structures memory addresses into words (JUMP, READ, STORE). Afterwards, high-level compilers of programming languages were developed, in order to allow the separation from machine specific code instructions, and have universal commands for operations execution.

Theoretically, the TM provides us with a set of abstractions that are universal, such as the Boolean operations that can be performed with information, making computers physical characteristics irrelevant for programming, but in fact, the first machines had their own specific opcodes that impossibilitated transference of code among machines. Also, as Carvalhais points out, TM is conceived with an infinite tape, which in actual computers does not occur. The computer has a limited storage capacity (Carvalhais, 2010:87).

Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship.

(Leiserson, Rivest, Stein, & Cormen, 2009)

Attempting to determine whether a computer could be used for musical composition, Lejaren Hiller and Leonard Isaacson have taken a series of experiments on modelling compositional choices, as a set of rules, onto it. Using the *Illiac* computer from the University of Illinois (Fig. 4) they implemented composition procedures, such as polyphony and counterpoint rules. Influenced by Claude Shannon's work on *Information Theory*[26] (Shannon, 1948), they also explored procedures based on statistics, such as Markov chains.

From a wide variety of musical material, they reduced compositional practice to algorithmic operations of analysis on the probabilities of notes to occur in musical phrases, generating new musical material that maintains the original genre and style. This method, known as *Monte Carlo*[27] is one of the most successful methods used for algorithmic composition.

[26] Shannon defines information as a probability function with no dimensions, no materiality, and no necessary connection with meaning. Information is regarded as a representation of choice from among a range of possibilities (Hayles, 2008:18), a quantity measured by the probability of certain events to occur. The concern, for information theory, is not the semantics of a message, but rather the potential for using a machine in order to logically operate within a set of established rules. These concepts are not synonymous. Information theory removed semantics from communication in order to be able to operate with it. Also, Shannon limited his theory to *ergodic systems*, music being one of the included types of communication (Ariza, 2005:40).

[27] A thorough description of the *Monte Carlo* method is provided in Buxton (1977).

Fig. 4 Hiller, Lejaren; Isaacson, Leonard, «Illiac Suite» Quartet No. 4 for strings, 1957
© Hiller, Lejaren; Isaacson, Leonard

As a result of these experiments Hiller and Isaacson presented *ILLIAC Suite*[29] (1956), credited to be the first piece of music composed by a digital computer. Hiller and Baker further expanded the techniques that were used in this composition and developed the *MUsic SImulator-Interpreter for COMpositional Procedures* (MUSICOMP) system (Ariza, 2005:44)[30].

Hiller also worked with John Cage, expanding this system by employing subroutines for number selection based on the *I-Ching*[31] and on Mozart's *Musikalisches Würfelspiel*. This collaboration resulted in the creation of *HPSCHD* (1969), a composition that employed seven harpsichords, fifty-one computer generated tapes, eight slide projectors, and seven film projectors.

Around the same period, Iannis Xenakis was exploring stochastic techniques (cf. Ch.1.4). As early as 1962, Xenakis developed the *Stochastic Music Program* (SMP), a score-

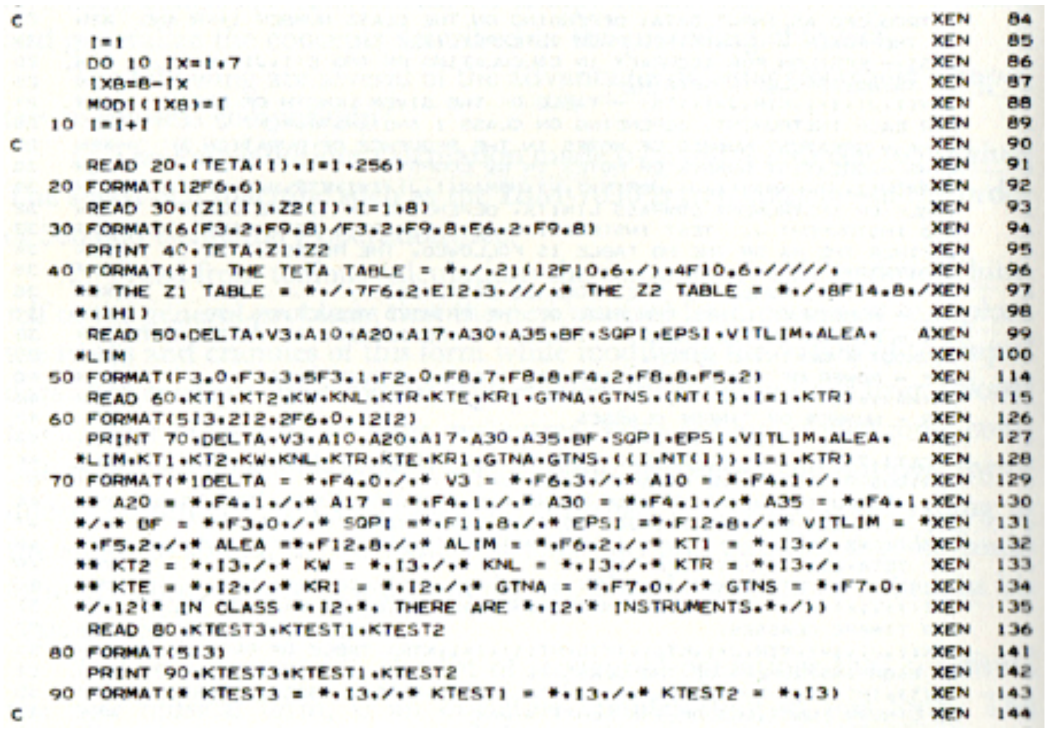
[29] Also known as *String Quartet No. 4.*

[30] Ariza (2011) provides a historical background on Hiller and Isaacson's work, and also the work from David Caplin, Dietrich Prinz, and Harriet Padberg, which was developed in the same period.

[31] We had previously discussed the usage of the *I-Ching* by John Cage (cf. Ch1.4.1).

computing program where he implemented stochastic distributions, written in *Fortran* on an *IBM* mainframe (Xenakis, 1992:144).

Xenakis describes the advantages of using computers in musical composition as of becoming some sort of pilot. Freed from long hours of laborious calculations, the composer can devote himself to the general problems that the new music form poses. For Xenakis, the program's source is an objective manifestation of the composition's form, and even suggests the possibility of dispatching the work to any distant point on earth, to be explored by other "composer pilot", anticipating today's open-source culture.



```
C                                                                       XEN   84
      I=1                                                               XEN   85
      DO 10 IX=1,7                                                      XEN   86
      IXB=8-IX                                                          XEN   87
      MODI(IXB)=I                                                       XEN   88
   10 I=I+1                                                             XEN   89
C                                                                       XEN   90
      READ 20,(TETA(I),I=1,256)                                         XEN   91
   20 FORMAT(12F6.6)                                                    XEN   92
      READ 30,(Z1(I),Z2(I),I=1,8)                                       XEN   93
   30 FORMAT(6(F3.2,F9.8)/F3.2,F9.8,E6.2,F9.8)                          XEN   94
      PRINT 40,TETA,Z1,Z2                                               XEN   95
   40 FORMAT(*1   THE TETA TABLE = *,/,21(12F10.6,/),4F10.6,/////,      XEN   96
     ** THE Z1 TABLE = *,/,7F6.2,E12.3,///,* THE Z2 TABLE = *,/,8F14.8,/XEN   97
     *,1H1)                                                             XEN   98
      READ 50,DELTA,V3,A10,A20,A17,A30,A35,BF,SQPI,EPSI,VITLIM,ALEA,   AXEN   99
     *LIM                                                               XEN  100
   50 FORMAT(F3.0,F3.3,5F3.1,F2.0,F8.7,F8.8,F4.2,F8.8,F5.2)             XEN  114
      READ 60,KT1,KT2,KW,KNL,KTR,KTE,KR1,GTNA,GTNS,(NT(I),I=1,KTR)      XEN  115
   60 FORMAT(5I3,2I2,2F6.0,12I2)                                        XEN  126
      PRINT 70,DELTA,V3,A10,A20,A17,A30,A35,BF,SQPI,EPSI,VITLIM,ALEA,  AXEN  127
     *LIM,KT1,KT2,KW,KNL,KTR,KTE,KR1,GTNA,GTNS,((I,NT(I)),I=1,KTR)      XEN  128
   70 FORMAT(*1DELTA = *,F4.0,/,* V3 = *,F6.3,/,* A10 = *,F4.1,/,       XEN  129
     ** A20 = *,F4.1,/,* A17 = *,F4.1,/,* A30 = *,F4.1,/,* A35 = *,F4.1,XEN  130
     */,* BF = *,F3.0,/,* SQPI =*,F11.8,/,* EPSI =*,F12.8,/,* VITLIM = *XEN  131
     *,F5.2,/,* ALEA =*,F12.8,/,* ALIM = *,F6.2,/,* KT1 = *,I3,/,       XEN  132
     ** KT2 = *,I3,/,* KW = *,I3,/,* KNL = *,I3,/,* KTR = *,I3,/,       XEN  133
     ** KTE = *,I2,/,* KR1 = *,I2,/,* GTNA = *,F7.0,/,* GTNS = *,F7.0,  XEN  134
     */,12(* IN CLASS *,I2,*, THERE ARE *,I2,* INSTRUMENTS.*,/))        XEN  135
      READ 80,KTEST3,KTEST1,KTEST2                                      XEN  136
   80 FORMAT(5I3)                                                       XEN  141
      PRINT 90,KTEST3,KTEST1,KTEST2                                     XEN  142
   90 FORMAT(* KTEST3 = *,I3,/,* KTEST1 = *,I3,/,* KTEST2 = *,I3)       XEN  143
C                                                                       XEN  144
```

Fig. 5 Stochastic Music excerpt, Rewritten in Fortran IV (Xenakis, 1992:146)

However, reading Xenakis *ST* source code (Fig.5), implemented in *Fortran IV*, provides a good perception of the difficulties of composing with programming languages. Procedural programming consisted in linearly writing step-by-step instructions on what to do with data stored in a central structure. From an analytical perspective, it becomes difficult to perceive an overall compositional intention.

With the aim of testing serial music compositional rules with computers, in 1964, Gottfried Koenig began *Project 1* (PR1). His program allowed him to feed into the computer parameters such as instrument selection, time, pitch, registers and dynamics, which when executed would generate a *score*.

Koenig characterizes computer composition as 'the formulation of sets of rules with the aid of a computer with a view to working out musical contexts' (Koenig, 1991). The program embodies a "strategy and the compositional idea behind it". It is important to remark that Koenig described programming as computer composition, as a subjective practice, as it is highly related to the composers’ own understanding of music.

He asserts that composers "are in general inadequately prepared to use a composing program" (Koenig, 1975) because of their lack of capacity for thinking abstractly and self-analysis. The difficulties of programming led him to consider that it would be easier to modify the resulting text produced by the program than correcting the rules and re-writing the program.

The above cases expose a required logical thinking for the conception of correct algorithms and data structures that take the form of computer music programs. The computer operates as a powerful calculating device in the service of music.

The compositional efforts carried out by Hiller and Isaacson, Xenakis and Koenig are described by Horacio Vaggione as *Turing Music* (Solomos, 2005). The emergence of computer-aided algorithmic composition (CAAC), in which the composer takes Turing's formulation of the computer as a powerful symbolic operator to model music information from sets of rules represents an economy of means, allowing the composer to delegate to the machine certain 'laborious' aspects of his work, dedicating his time to compositional concerns. The usage of computers does not represent any significant change in the operational models that are carried out, not even in the intrinsic properties of music elements. The final outcome of these works remains as a set of parameters to be printed out in order to create a score to be performed.

2.2 Abstraction made Concrete

Beyond computers' potential to carry out symbolic operations for the creation and transformation of musical parameters, by the 1950s, they also became capable of generating sound. The significance of such capability lies in the fact that notation itself has become self-sufficient for the emergence of sound; computers have become capable of materializing instructions from abstraction into actual sounds. As Berio and others describe, they have enabled the "composition of sounds themselves" (Risset 1985).

The first experiments in sound generation preceded algorithmic composition. Geoff Hill programmed popular melodies on the CSIRAC computer (Doornbusch, 2004). Nonetheless, the most significant work was carried out by Max Mathews and his colleagues at Bell Laboratories. In 1957, as a demonstration of the *Music I* program, they presented an entirely generated monophonic *étude* called *The Silver Scale* (1957*)*, composed by Newman Guttman. This program allowed the generation of an equilateral triangular waveform to which pitch, amplitude and duration could be specified for each note (Roads & Mathews 1980). In the following years Mathews were fundamental for the research on sound synthesis, developing the most widely used approaches today.

2.2.1 Standard synthesis

On "The Digital Composer as a Musical Instrument", Mathews describes the possibilities opened up by digital-to-analog converters (DAC), with which streams of numbers generated by computers could be directly converted to sound waves, and consequently any perceivable sound could be produced (Mathews, 1963). According to Mathews, the two fundamental problems in sound synthesis were the vast amount of data needed to specify a pressure function, and the need for a simple, powerful language in which to describe a complex sequence of sounds (Mathews, Miller, Moore, Pierce, & Risset, 1969:34).

Attempting to solve these issues, Mathews developed *Music III (1959)* which implemented the concept of unit generators (*UGens*) for sound synthesis and signal processing. These building blocks consisted of algorithmic functions that output signals controlled by their parametric inputs (Roads 1999:787), such as oscillators, filters or mathematical operators. The *UGens* could be interconnected in order to generate more complex signals. This modular approach avoided having to write each and every single sample used in the work (Pope, 1993:25) and represented the potential to "have virtually an unlimited amount of equipment" (Howe, 2009). In order to program a musical work, the composer would have to write an 'orchestra', describing the instruments as *UGen* connection descriptions, and then a 'score' or 'note list’, with the data to control instruments' parameters and start times.

Such ‘top-down’ approach to specify sounds resembles acoustic models for composition facilitating composers’ transition to computer music. The generality of audio programming languages accessible today – such as *CSound*, *SuperCollider* or *Chuck* – are regarded as descendants of *Music-N* and remain faithful to this concept.

Throughout the 1960s, computers were not attractive to the majority of composers, however they permitted a faster and more efficient study of complex sounds, in particular within the field of psychoacoustics.

One of the first researchers to work with these new digital tools was Jean Claude Risset, who found an opportunity to research on the incompatibility between the nature of sound *material* and its *form*. As Risset points out (1976), the richness of musique concréte was constrained by the rudimentary means to control it, and Elekronische, in opposition, offered means at the cost of 'simple and dull sounds'. Computers could offer the best of both approaches by providing means to create and to control rich sounds. Risset began experimenting on *additive synthesis*[32], discovering important timbral properties of sound, such as the importance of inharmonic spectra and the role of their amplitude envelope (Risset 1976). Such discoveries enabled Risset to develop an understanding of sound by combining acoustics, sound synthesis and psychoacoustics that allowed the creation of works such as *Inharmonique* (1977) or *Mutations* (1969).

The potential of Mathew's *UGens* is verifiable by John Chowning's discovery of the musical possibilities of Frequency Modulation (FM). Just like Risset, Chowning was strongly influenced by Mathews' 1963 paper, in particular by the statement declaring computers' "unlimited" sonic possibilities. Chowning began playing with the combination of oscillators

[32] A synthesis technique derived from the Fourier Theorem which consists of describing complex sounds as a sum of sinusoidal components diferenciated by their phase, amplitude and frequency.

and developed Frequency Modulation synthesis. Such approach enabled the creation of very rich sounds using limited resources, in opposition to additive synthesis. His discovery attracted Yamaha, that bought the FM patent and, in 1983, implemented a ‘real-time’ version in the DX7, which is regarded as one of the most successful synthesizers in history (Serafin, 2007).

2.2.2 Beyond standard synthesis

Max Mathews approach to sound synthesis developed on a rather technical perspective. Constrained by computer's limited resources Mathews provided a *high-level* language for composers to be able to take advantage of such techniques. Still today, computer music conforms these acoustic or mathematical models, making a distinction between the "instrument", a model to describe the "sound material", and the "score", to describe musical structure. (Di Scipio, 2003)

Di Scipio argues that "one of the most relevant challenges of today's computer music implies a profound re-working of the modus operandi just described, and inevitably results in a different perspective concerning the relation of material to form, of sound to structure" (Di Scipio, 2003). Such perspective, more concerned with *composing the sound*, can be found in the "*non-standard synthesis*" (Holtzman 1979) methods pioneered by Iannis Xenakis, Gottfried-Michael Koenig and Herbert Brün. The synthesis results from the manipulation of individual digital samples through compositional processes, laying out the foundations for microsound (Roads 2004).

The *Sound Synthesis Program (*SSP) was conceived by Gottfried Michael Koenig as early as 1972 and finished by Paul Berg in 1977. The program allowed the composer to construct individual elements of waveforms and large-scale composition structures from its microstructure. Using amplitude and time-value points, the waveforms were assembled using selection principles inherited from his program *PR2* such as randomness, groups, sequence, ratios and tendencies. By using such principles, the creation of standard waveforms such as sine, triangle, ramp, square waves was difficult to produce. (Luque 2006, Doornbusch 2009)

Regarding SSP, in a 1978 interview, Koenig said:

> My intention was to go away from the classical instrumental definitions of sound in terms of loudness, pitch and duration and so on, because then you could refer to musical elements which are not necessarily the elements of the language of today. To explore a new field of sound possibilities I thought it best to close the classical descriptions of sound and open up an experimental field in which you would really have to start again.
> (Koenig qtd. in Roads 1978)

Sawdust, by Herbert Brün, was also conceived as a program for compositional structuring of waveforms. The composer can specify small fragments of waveforms that are linked, merged and mingled for generating new timbres. Brün has taken a serial approach by constantly mapping waveform lengths to tempered pitch scales, even producing twelve-tone rows and chords for the organization of waveforms (Döbereiner, 2010:28).

Dynamic Stochastic Synthesis by Iannis Xenakis is described on "New Proposals in Microsound Structure" (Xenakis, 1992:242) and, similarly to the techniques developed by Brün and Koenig, consists of generating waveforms from duration and amplitude values using processes conceived by the composer. Xenakis recognized the importance of the variations in amplitude and frequency of even the simplest orchestral sounds (1992:244) and instead of generating complex sounds by juxtaposing finite elements, as devised for instance in *Analogique B* and *Concret PH*, they are constructed with continuous stochastic variations of the sound pressure directly. Xenakis illustrates such variations as a "particle capriciously moving around equilibrium positions along the pressure ordinate in a non-deterministic way, recurring to the use of 'random walk' " (1992:246) and proposes a diversity of methods for stochastic microsound synthesis.[33]

The *POD* (*POisson Distribution*) programs have been developed by Barry Truax since 1972, at the Institute of Sonology at Utrecht. *POD* generates a series of events within the

[33] Xenakis description of these methods (Xenalis 1992:246-247) can be complemented by Sergio Luque (Luque 2006).

bounds of tendency masks using the Poisson distribution[34]. As Truax asserts, "sound and structure become increasingly inseperable" (Truax 2000:119).

2.3 Sound Machines

Developing from Turing's conception of the computer, composers have been capable of modelling compositional ideas, developing techniques that proved their value for supporting a creative activity such as music creation, far beyond the expectations on these devices.

The difficulties of programming, derived from early mainframe machines, were progressively resolved with the maturation and standardization of the computer. Layers of abstraction from machine language were continuously created. From *Assembly* to higher-level languages, such as *ALGOL* or *Pascal*, general-purpose programming languages increasingly allowed more plasticity to develop concepts such as the ones required by music, introducing new structures and procedures[38].

Allied to the potential of algorithmic composition, the possibilities of sound synthesis, pioneered by Mathews, provided a new perspective on the computer as a complete sound machine: a device that enabled the description and generation of sound.

In fact, programming languages for synthesis, such as *Music-N*, offered high degree of flexibility (Roads, 1999:818), in particular since the 1980s due to the development of faster computer processors[39] A fundamental achievement was the development of the *C*[40] language, which allowed portability among distinct machines, exponentiating software development. In

[34] See also the already described work on Xenakis' Dynamic Stochastic Synthesis (cf. Ch.2.2.2).

[38] The basic data models provided (*Strings* and *Numbers*) were extended by enabling the construction of structures; Fortran, for instance, already permitted *Arrays* (lists of structures). Also, new procedures were permitted, such as routine nesting.

[39] A survey on sound synthesis is provided by Smith (1991).

[40] The *C* Language was developed by Dennis Ritchie at Bell Labs, reelased in 1972.

the following years, a diversity of programming languages were created, with particular relevance to *Common Lisp Music* (CLM)[41], *Cmix*[42] and *CSound*[43].

Starting in the late 1960s, research on human-computer interaction enabled a shift of concerns from formal aspects of computation towards new means to relate with the computer. While many composers found themselves at ease with the logics of programming languages, explorations on new devices and interfaces began to emerge, attempting to provide the computer with an interacting body, as found in acoustic and electronic instruments.

Pioneering such direction, Mathews and Richard Moore developed the *GROOVE* (Generated Real-time Operations on Voltage-Controlled Equipment) program (Mathews & Moore, 1970). In this period, computers did not have sufficient speed for the generation of sound in *real-time*[44] but they were already capable of converting digital signals to analog voltage and vice-versa. This enabled the development of *hybrid systems*, in which the computer could send and receive voltages to control analog synthesizers in real-time (Mathews, 1963). It is relevant to note that the research developed at Bell Labs also contributed to the development of analog synthesizers; as Risset asserts[45], Mathews' modular approach to *Ugens* was adopted by the synthesizers built by Robert Moog, Don Buchla or Pedro Ketoff.

The *GROOVE* program enabled the creation, storage and edition of 'functions of time' to output control to voltage-controllable devices such as synthesizers. Such functions could

---

[41] CLM is a sound synthesis language that descends from the *MUSIC-N* family. It was developed in the late 1980s by Bill Schottstaedt at Stanford University. This language is particularly useful to define hierarchical musical due to LisP's recursive nature. A more recent LisP-based programming language is Nyquist, created by Roger Dannenberg (Wang, 2008).

[42] *Cmix, by* Paul Lansky, is a C library that facilitates signal processing and sound manipulation routines, unified by a well-defined API. A score could be specified in the Cmix scoring language, called MINC (which stands for "MINC Is Not C!"). MINC's syntax resembled that of C and proved to be one of the most powerful scoring tools of the era, due to its support for control structures (such as loops). Cmix is still distributed and widely used today, primarily in the form of RTCmix (the RT stands for real-time) (Wang, 2008).

[43] Barry Vercoe's CSound developed from a Music11 version and rewritten in C, was released in 1986 as a shareware. (Manning, 2009:88).

[44] Howe describes the length of time it took to produce the sound and the reduced bandwidth permitted for its realisation (10kHz mono) (Howe, 2009).

[45] Personal presentation at Universidade Católica do Porto 2008/09/10.

be mapped to real-time input controls such as knobs, keyboard keys or buttons. They could also be defined as coordinates of points along with the type of interpolation (step or ramp). Complex functions could be achieved by combining other functions through arithmetic operations. This system's interaction logic resembles current programming environments such as *SuperCollider*, notwithstanding its processing power.

The usage of computers for controlling dedicated synthesis systems, in similarity to the GROOVE program, had become a common practice. The *Sal-Mar Construction* (1970), for instance, was a hybrid analog-digital machine for composition; it was controlled by touch-sensitive switches connected to various modular devices and the output could be sent to any of its twenty-four loudspeakers (Harley, 2009:115). Gottfried Koenig connected his *PR1* to analog sound synthesis, having composed *Terminus II* (1966/67) and the *Funktionen Series* (1967-1969) with this method.

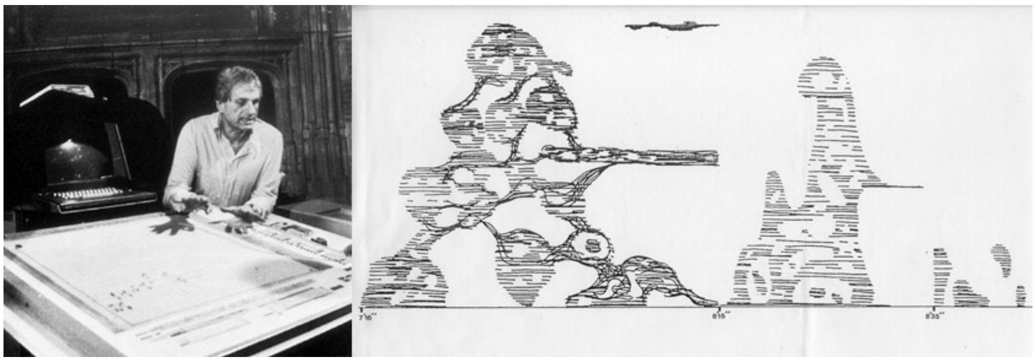


Fig. 6 UPIC *(Unite Polyagogique Informatique du CEMAMu) by Iannis Xenakis,* 1977-1994

Iannis Xenakis also connected SMP to analog systems and, between 1977 and 1994, developed the *UPIC (Unite Polyagogique Informatique du CEMAMu).* It allows the user to create a composition by tracing lines on a graphic pad with an electromagnetic pencil. The sound is generated with *table-lookup* synthesis, mapping the drawn sample amplitude to the frequency by which a wavetable sample is read (Di Scipio, 1998). *Mycenae Alpha* (1978) was the first composition to be entirely realized on this system.

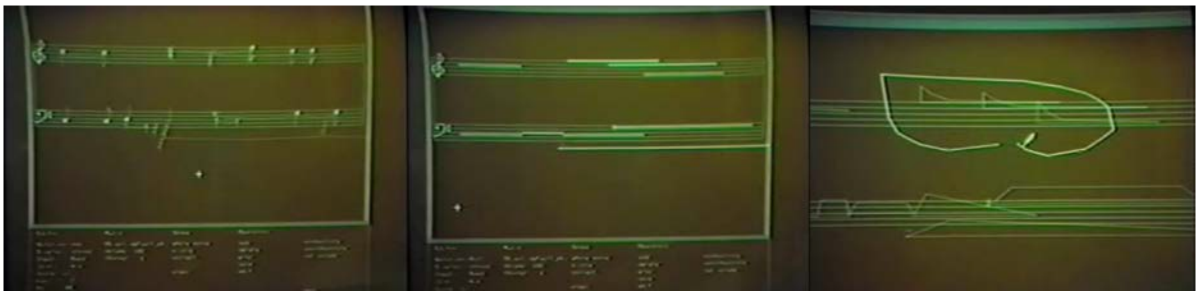


Fig. 7 SSSP (Structured Sound Synthesis Project) by William Buxton, 1978

Interested in developing a highly interactive environment for computer aided composition of music, William Buxton developed *SSSP* (Structured Sound Synthesis Project). His research led to the creation of score-editing tools (Fig.7) that set the paradigm for today's music editing software.

Buxton considered that most systems gravitated between a note-by-note approach, as we have regarded with Mathews acoustic model, and those which dealt with the score as a single entity, such as the "non-standard synthesis" approaches that were carried out by Xenakis. These two extremes, according to Buxton, could be regarded as instances of the same thing, and therefore he attempted to provide a structure that would facilitate the implementation of different high-level external representations of a repertoire of timbres, and the exploration of their multi-dimensional attributes.

Buxton's approach makes an interpretation of Schaeffer's *Sound Object* encompassed within the computational domain, relating the notion of "object" in a striking resemblance to Alan Kay[46] and his *Object Oriented Programming* paradigm[47]. As Buxton describes, an

[46] In the 1960s, artificial intelligence research began developing implementation methods for developing programs with higher complexity and intelligence. One of such methods is *Object Oriented Programming (OOP)*, pioneered by the language *Simula* (1966), developed by Ole-Johan Dahl and Kristen Nygaardand later adopted by Alan Kay. Influenced by the biological-machinic symbiosis proposed by the field of *Cybernetics*, Kay developed his language *Smalltalk*, being a fundamental reference for the early developments of *human-computer interaction.* Buxton developed a similar approach, drawn from Ivan Sutherland's work on *Sketchpad* (1963).

object is "a named set of attributes which will result in sounds having different pitches, durations, and amplitudes to be perceived as having the same timbre [,] simply providing a conceptual framework in which the composer can view his activities" (Buxton et al., 1978). The compositional work would therefore evolve by defining isolated components that could encapsulate a heterogeneous variety of techniques, such as FM synthesis, additive synthesis, waveshaping, fixed-waveform synthesis and VOSIM, and then use them and relate them within higher level objects, constituting a network of interrelated objects. The advantage of such approach lies in the fact that once an object is defined, one does not need to be concerned with its inner logic (*black-boxing*). This object-oriented approach is evident in the graphical interfaces of the *dataflow* programming languages *Max* (1988) by Miller Puckette, or *Kyma* (1987) by Carla Scalleti.

[47] *Object Oriented Programming (OOP)* is a paradigm that uses objects and their relations to create programs. Traditional programming paradigms, such as functional, constraints or logical, separated algorithmic processes from data, stored in a centralized structure. In OOP, objects are abstractions (classes, in general) that contain data (variables) and the procedures (methods) that can occur. Programming consists of defining these "objects as its fundamental logical building blocks" (Booch, 1998:35) and describing the relations that need to be set among them, creating an organized network. Classes are all hierarchically united via inheritance relationships. The relevance of this approach consists of enabling programmers to stop thinking in terms of writing for the abstract machine – Turing's conception of algorithmic – and rather consider programming as an act of modelling real-world concepts. A class mediates a real-world entity and enclosures the possible actions that can take place with it. Programming then, becomes creating taxonomies, hierarchy structures and possibilities for intervention. Alan Kay refers to *OOP* as a "successful attempt to qualitatively improve the efficiency of modelling the ever more complex systems and user relationships made possible by the silicon explosion" (Kay, 1993:3). As Robinson points out, *OOP* became a significant technology, in particular with the programming language *C++*, enabling the general development of software for desktop computers, such as the music domain applications we afford today.

The first era of computer music is defined by experiments that were carried out in order to access how computers could be used for music creation, resulting in the development of software for algorithmic composition and sound synthesis.

The work by Hiller and Isaacson illustrates how a diversity of compositional concepts could be formalized and inscribed into computers. Inheriting Turing's concept of computer, a diversity of procedures were implemented. From simple permutations and chance procedures towards Markov chains their work laid out the foundations of algorithmic composition, further developed by composers such as Xenakis, Brün or Koenig.

Also, a significant aspect for computer music is concerning the possibility to generate sound, as pioneered by Max Mathews in the *Music-N* music software. We have regarded how, from an abstract description written in a programming language, computers were capable of generating and performing concrete sounds; programming languages can be regarded as particular scores, fully prescribing musical works, providing a high degree of autonomy and a rich diversity of sounds at one's disposal.

As computers became fully capable sonic machines, explorations on means to interact with these machines began to emerge. The presented systems, although for real-time, were not conceived for performance, given their dimensions and maintenance requirements, but rather for equipping studios with the intention of improving the composer's interaction with their materials. The concept of taking a dialogic approach with computer-based software was already present in these systems, by allowing the creation of works that could be composed by playing with devices, coordinating a series of parameterizations that were recorded into the program. Nevertheless, the role of computers in the context of interaction remains to be addressed, which we'll look into in the following chapter.

(blank page)

3 INTERACTIVE MUSIC SYSTEMS

The possibility of interacting with computational tools in performance was brought to the forefront with the exponential development of personal computers, linked to the development of programming languages that facilitated algorithmic composition and the generation of sound, either through synthesis or sampling techniques.

Interaction has always been a natural extension to music practice, in particular within the improvisational practices potentiated by the afro-american *Jazz* tradition. Also, the influential work by John Cage already suggested a diversity of possibilities for interaction, opening way to a new generation of composers, such as George Mumma, Robert Ashley, Alvin Lucier, who developed analog circuits for real-time sound generation.

In this chapter we will overview a diversity of approaches to the creation and usage of interactive music systems. As Jordá mentions, it seems difficult to cover all this diversity and multidimensionality of such computational tools. Various taxonomies have been described elsewhere (Drummond, 2009; Jordá, 2005; Pressing, 1987; Rowe, 1993; Spiegel, 1992; Winkler, 2001), however they do little in order to provide an understanding of how these systems actually work and the potential they hold for future developments. Therefore, we attempt to underpin particularities, subjective directions that have been carried out that can contribute to the understanding of the diverse possibilities for interactive music practice.

3.1 'Interactive Composing'

Joel Chadabe, one of the pioneers on the development of interactive music systems, has been exploring the possibilities of human-computer interaction since the late 1960s. Having used large-scale commercial synthesizers for studio-design (Chadabe, 1967), his work has progressively evolved towards processes by which the many features of a synthesizer could be managed during live performance.

In 1981, Chadabe proposed the term "interactive composing" to describe a "performance process wherein a performer shares control of the music by interacting with an instrument that in itself generates new material" (Chadabe, 1984). As Chadabe states:

> "I do not compose pieces, but rather activities, defining a 'piece' as a construction with a beginning and end that exists independent of its listeners and within its own boundaries of time. An 'activity' unfolds because of the way people perform; and consequently, an activity happens in the time of living; and art comes closer to life." (Chadabe, 2001:¶1)

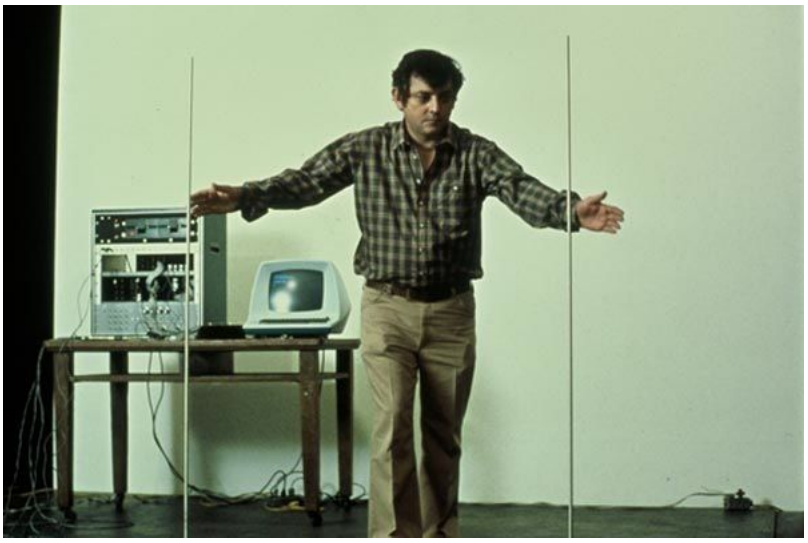


Fig. 8 Joel Chadabe performing Solo at the New Music New York Festival, *The Kitchen*, 1979
© Joel Chadabe

Although one of the first demonstrations using computers for performance was carried out by Peter Zinoview and his associates from *Interactive Music Studios* in 1968 (Roads 1999:685), *Solo* (1978) (Fig.8) is credited as the first performance carried out with a completely digital system. It was an algorithmic composition that reacted to gestures captured by two modified Theremin antennas connected to a *Synclavier*[52]. The composition took its form in each performance as the result of a mutually influential process between the performer and the instrument being played. Partially controlling the work with the computer, the result contained surprising as well as predictable elements. Chadabe describes it as "conversing with a clever friend who was never boring but always responsive"; as "conversing with a musical instrument that seemed to have its own interesting personality." (Chadabe, 1997)

> I pointed out that the compositional algorithm in my composition *Solo* (1978) seemed to make musical decisions independent of my actions as a performer, that I had to react to it at the same time that it reacted to my gestures, and that the term "interactive composing" meant that the composition took its form in each performance as the result of a mutually influential process between myself and the instrument I was playing.
> (Chadabe, 2009)

Adopting the *cybersonics* ideas previously explored by Mumma (cf. Ch.1.5), Chadabe was concerned with the complexities that emerges from opening the system to human interaction in a performative context.

> At this time, however, with interactive media fast becoming a normal part of our culture, these computer-aided performance models may be seen in a new light. From the composer's point of view, at the same time that computer-aided performance provides opportunities for new ways to address the public, it challenges traditional notions as to the skills required for performance. It also challenges the very basis of what we think of as a musical composition.
> (Chadabe, 2009)

[52] One of the first commercially available digital synthesizers, developed by Jon Appleton in 1977.

3.2 Interplay with computers

The arrival of the first microcomputers, in the 1970s, opened the ground for a new generation of composers to develop new approaches to interaction through the usage of these machines. In a context where tweaking cheap electronics was a common practice, computers represented a big conceptual change for music composition, and consequently, new opportunities.

Analog circuits were a real-time action-reaction medium, enabling the simultaneous development of separate circuits that could be activated by switching and routing signals. The qualities of rhythm, harmony, melody and form were abandoned in favour of spontaneity, based on the modification of non-pitched aspects of sound such as the shape of an envelope, timbre, rhythm, filtering, effects (echo, delay, ring modulation, etc.), amplitude and duration (Holmes, 2008:280).

> "You turned it on, flipped a switch, and it just happened in parallel with whatever else was going on: another circuit, a circuit affecting another circuit, a musician playing along, a voltage-controlled device modifying the output of the circuit, and so forth. It was solid state in the conceptual as well as circuitry sense"
> (Holmes, 2008:280).

Computers, however, offered many advantages because of their ability to be configured dynamically, to read and store music information with precision and repeatability, without the need to solder wires. These advantages encouraged composers to learn programming languages such as *Forth*[53] or *Turtle Logo*[54] in order to develop music systems to be used in performance.

One of such composers is David Behrman, who was exploring means to interact with computers and analog circuits. *On the Other Ocean* (1978), is an improvisational work for Flute, Bassoon and electronics, in which Behrman developed a system that consisted of pitch-sensing circuits connected to a *Kim-I* microcomputer that controlled a hand-made analog

[53] An imperative language created by Charles Moore in 1970.

[54] An educational programming language created by Bolt, Beranek and Newman in 1967.

synthesizer. This system would generate long harmonic tones that resulted from the sounds of two musicians, affecting their improvising performance. Eigenfeld points out that the result was less "radical" than Chadabe's, in the sense that the musicians were playing equal tempered, tonal harmonies that were extended by the synthesizer (Eigenfeldt, 2007).

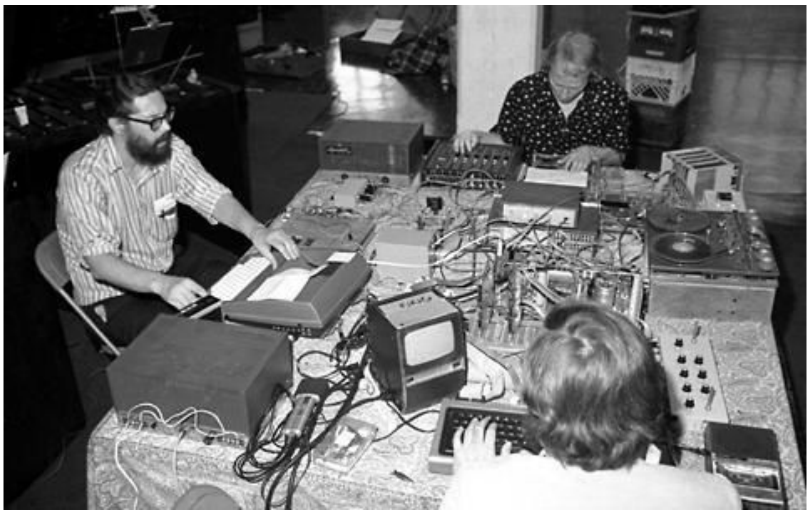


Fig. 9 The League of Automatic Composers
© The League of Automatic Composers

In the same period, Jim Horton conceived the possibility of building a "silicon orchestra" of microcomputers linked together into an interactive network. In 1978 he joined John Bischoff and Rich Gold and formed the *League of Automatic Composers* (Fig. 9). This was the first microcomputer orchestra and the first network band in history. The compositional process of its members consisted of each one developing autonomous sections that would produce their own sound, either by controlling analog synthesizers or generating signals. These sections would then be adapted to the band's setting. Each would define some parameters to be externally controlled by other computers, and also, by sending data to control others. And finally, these sections would be played simultaneously, becoming interacting 'subcompositions'. As Brown and Birchoff assert, connecting the computers, and having them share information, not only unified the distinct "subcompositions" but also added a sense of autonomous direction, of independence of the machines (Brown & Bischof, 2005:381).

In the early performances, they would leave the machines performing unattended and would listen along with the audience, but later, they started interacting directly with the machines, prefiguring today's live-coding practices. Although no details are provided, Bischoff describes the distinctly improvisational character to many of these works, "as the music was always different in its detail. Mathematical theories of melody, experimental tuning systems, artificial-intelligence algorithms, improvisational instrument design, and interactive performance were a few of the areas explored in these solo works" (2005:381).

The Hub ensemble grew from the *League of Automatic Music Composers* and continued exploring the expressive potential of interactive networks. It was formed in 1986 and included members Mark Trayle, Phil Stone, Scot Gresham-Lancaster, John Bischoff, Chris Brown, and Tim Perkins. In this ensemble, the communications had a distinct configuration as they used a microcomputer as a mailbox, thus the name *The Hub*, posting the data used to control each individual system. Any player could then retrieve the posted data and use it.

Bischof and Brown define *The Hub* as "the sound of individual musical intelligences connected by network architectures"(Brown & Bischof ,2005:384). A composition by *The Hub* would be defined by the inclusion of the specification of the modes of interaction to be carried out, the types of information to be shared and the means of sharing it among the members.

Mark Trayle's *Simple Degradation* (1987), for instance, consisted of one conductor generating waveforms simulating the response of plucked strings and then sharing it through the network for the other performers to use it for amplitude modulation. The performers were free to define all remaining properties of the output signal; Stone's *Borrowing and Stealing* (1989) consisted in sharing melodic riffs which could be "transformed in any of a multitude of ways, and replayed" (Brown & Bischof, 2005:385). The resulting riff would be returned to the hub. In *Vague Notions of Lost Textures* (1987) Gresham-Lancaster conceived a *chat* system, by allowing the writing of text messages among members in the hub. Many of these experiments have been revisited by live coding practices today[55].

This pioneering work is characterized by the limited resources composers had at their disposal (microcomputers and hand-made electronic circuits), nevertheless, they displayed a

[55] A critical approach to programming in the context of performative oriented practices is presented in Ward, Rohrhuber, Olofsson, & McLean (2004).

diversity of approaches to interaction with computers, performers and the audience, being still influential in today's music practice.

The presented developments represent a conceptual change regarding the use of computers in music practices as they are opened to the context of performance, differing from their pure logical (disembodied) ancestors. They are dynamically oriented, involved in networking with other machines as well as interacting with humans.

3.3 Intelligent Music Systems

Research on human-computer interaction and artificial intelligence provided new concepts that contributed to the development of programming languages, such as *C++*[56], and paradigms, such as the already discussed *Object Oriented Programming*. These developments contributed to the generalization of music software for desktop computers, such as the *Music Mouse* (1986), an intelligent instrument that attempted to facilitate the creation of electronic music in real-time, developed by Laurie Spiegel at *Bell Laboratories*. Another example is *M* (1986), an interactive composing system based on Markov chains published by *Intelligent Music*, a company that was created by Joel Chadabe (Zicarelli, 1987).

In this context, an interest in the creation of interactive music systems that could afford a higher complexity and intelligence emerged. These systems are computer intensive, and in the 1980s, they could only be developed in research centres such as *Institut de Recherche et Coordination Acoustique / Musique* (IRCAM)[57] or the *STudio for Electro-Instrumental Music* (STEIM)[58]. These research institutions continued to expand the scope of possibilities for compositional practice, allowing the development of resources, knowledge

[56] *C++ (C with classes)* was developed by Bjarne Stroustrup in 1979, extending the *C* language with high-level features for program organization.

[57] http://www.ircam.fr/

[58] http://steim.org/

and technologies for music creation that only became affordable for composers in the following decade.

3.3.1 'Creative Machines'

One of the most significant examples of interactive music systems for music performance is George Lewis' *Voyager*, an "interactive musical environment" that analyses the performance of human improvisers in real-time to develop an automatic composition. Its developments began in 1986[59], at *STEIM* and continued until 1995 (Panken & Lewis, 2009).

A major concern in Lewis' work is to "de-instrumentalize" the computer, treating it not as an instrument, but as an independent improviser with personality of "his own". For such, the computer system is devised in order to create a vast number of processes, an orchestra "without necessarily involving a central authority". It cannot be controlled but rather influenced through a *dialogic* discourse and its responses are not predictable, raging "from complete communion to utter indifference". The output is purely generative, with the exception of stored elemental material, such as scales and durations.

George Lewis uses probabilities to select from tables of stored melodic and rhythmic material, and these probabilities are again modified by the successions actually played out.

> Instead of asking about the value placed (by whom?) on artworks made by computers, Voyager continually refers to human expression. Rather than asking if computers can be creative and intelligent—those qualities, again, that we seek in our mates, or at least in a good blind date—Voyager asks us where our own creativity and intelligence might lie—not "How do we create intelligence?" but "How do we find it?" Ultimately, the subject of Voyager is not technology or computers at all, but musicality itself.
> (Lewis, 2000)

Lewis provides scarce details on the technical implementation of *Voyager*, being more concerned with the cultural and aesthetic aspects that surround his work. On "*Too Many*

---

[59] According to Jordá, Lewis' program *Chamber Music for Humans and Nonhumans*, developed at *IRCAM* in 1982 evolved into *Voyager* (2005:68).

Notes", Lewis states that "interactivity has gradually become a metonym for information retrieval rather than dialogue", regretting the rapid development of standardized modes of human-computer interaction instead of exploring the possibilities computers enable for a musical discourse (Lewis, 2000).

As previously considered by George Mumma, Joel Chadabe, among others, Lewis asserts that the development of a musical system is a compositional act by itself. This position helps us to understand at what point programs like *Voyager* are not universal, but instead represent the particular ideas of their creators.

3.3.2 Artificial Performer

Another proposal for an Interactive Music System such as *Voyager* is Robert Rowe's *Cypher*. His work has become a widely recognized reference for its detailed documentation and systematisation, discussed in detail in his books (Rowe, 1993, 2001).

> Interactive computer music systems are those whose behaviour changes in response to musical input. Such responsiveness allows these systems to participate in live performances of both notated and improvised music
> (Rowe, 1993:1)

The architecture of *Cypher* is derived from Marvin Minsky's *Society of Minds* (1988). Minsky's theory develops towards the possibility of artificial intelligence by regarding the mind as a society of agents with circumscribed abilities. According to Minsky, cognition can be modelled by assembling autonomous agents, "any part or process of the mind that by itself is simple enough to understand", into "agencies" without regard to what each of its parts does by itself, constituting a higher-level system. (Minsky, 1988:326)

As such, Rowe formulates that interactive music systems are situated systems of autonomous agents that continuously interact with the outside world. They "receive information from sensors, perform calculations based on that information, and send commands to synthesizers or other sound processors (effectors) in response" (Rowe, 1993:258).

His compositional work consists of providing musicianship to such systems by programming musical concepts that allow the machine to recognize and act upon human musical activity, becoming an artificial autonomous player. Rowe followed the premises of artificial intelligence in an attempt to provide enough information to the computer so that it can be a partner in performing. *Cypher* is described as an *expert system*[60] that generates a solution using a knowledge representation system: collections of condition-action pairs.

In his work we identify an interest in providing the computer with musicianship, in close relation to AI research, in order to create an artificial performer. It is important to note that Rowe claims that:

> No necessary isomorphism to human cognitive processes is being emulated [.] There is no claim that these systems are intelligent, creative composers, but rather, the implementation of certain processes, the delegation of compositional ideas in the form of code
> (Rowe, 2001:237).

The definition of interactive systems, provided by Rowe, is restricted to systems "which posses the ability to 'listen'" (Jordá, 2005:79). Although *machine listening*[61] represents an important possibility, it "is not the essence of what we should understand by interactive music system" (Jordá, 2005:80).

One must also note that the technologies that were being used deeply restrict the creative and aesthetic possibilities of musical practice. Both *Voyager* and *Cypher* are systems that develop on the ability to listen to the performance of improvising musicians using *audio-to-MIDI*[62] converters, in order to inform the computer about the material that is being

[60] A computer system that emulates the decision-making ability of a human.

[61] Computational techniques that extract features from sound emulating human listening capabilities.

[62] In 1983, upon a common standard agreement between commercial manufacturers, *MIDI* (Musical Instrument Digital Interface) was developed. This standard facilitated the vendor-independent usage of distinct devices, such as synthesizers, samplers and controllers to communicate within a common protocol. It offered simple and inexpensive means to extend the possibilities for music instrument networking (Loy & Abbott, 1985).

performed[63]. As Wishart asserts, such technology restricts sound properties to a note-based paradigm fostering preconceptions about electronic music that endures to present days (Wishart, 1996).

An example of such constraints can be found in the practice of the *Hub*. Their members changed their custom communications protocol to MIDI, attempting to make their processes more accessible to other musicians. This protocol deeply changed the role of communications. Custom format messages such as waveforms or texts were not possible, and instead of each member accessing data from the hub, they would send private midi messages to other members. *Waxlips (*1991) by Perkins "was an attempt to find the simplest Hub piece possible". The rules were that each player would only send and receive requests to play one note.

> Upon receiving the request, each should play the note requested, and then transform the note in some fixed way to a different note, and send it out to someone else. The transformation can follow any rule the player wants, with the one limitation that within any one section of the piece, the same rule must be followed (so that any particular message input will always cause the same new message output)
> (Brown & Bischof, 2005:387).

3.4 Computers as Instruments

Contrasting with the view of computers as autonomous intelligent systems, the notion of using the computer as a musical instrument became more evident in the early 1990s[64]. The possibilities to process sound in real-time attracted many composers who were interested in the potential of the computer for improvisational practices, establishing a strong link to free

[63] (Rowe:12), for instance, describes on a variety of techniques for higher precision detection of pitch and gesture control.

[64] Related to this perspective, we have already discussed the work by Max Mathews in previous chapters. Also, throughout the 1980s, software such as the *Music Mouse* by Laurie Spiegel or *M* by Joel Chadabe had been developed.

jazz and European free improvisation, as carried out by composers such as Lawrence Casserley, Tim Perkins, Joel Ryan, Richard Barrett, Paul Obermayer, among others.

As Perkins asserts, clever algorithms don't guarantee the perceptually important aspects of the music (Perkins, 2009:163). In view of this, many composers directed their concerns towards the actual physical properties of sound and the means to control them. As such, the focus turned to the use of the computer as a sound-generating device, and the development of interfaces to control them in real-time.

STEIM is particularly relevant in this field, promoting a human-focused approach to technology and asserting that it has to be tailored to the individual. The *Hands* (1984) by Michel Waisvisz (Fig.10) is an iconic example of an interface that enables a rich sonic control while providing a visual reference of the activity that is being carried out during the performance.

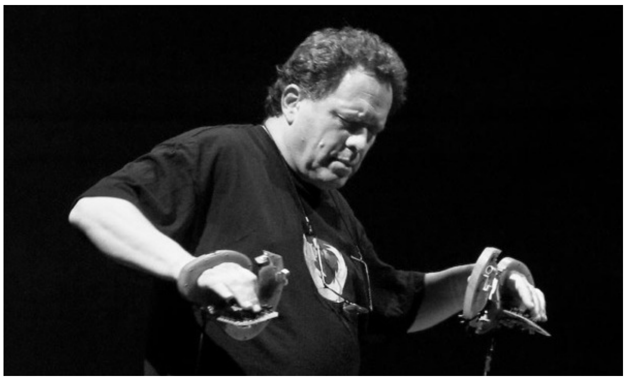


Fig. 10 The Hands, by Michel Waisvisz (1984)

The technology developed by *IRCAM*, such as the *Max* family environments, has become one of the dominant computer music interactive environments[65] for facilitating the creation of musical instruments (*patches*) and its easiness to connect external devices.

---

[65] The first version of *Max* (originally called *Patcher*) was implemented by Miller Puckette (1988) as a graphical application for routing and controlling external synthesizers in real-time. It has undergone through a series of versions to control IRCAM's DSP hardware (Max/FTS "faster than sound"). It was first used to control *4X* synthesizer, and in 1992, IRCAM made it available as the *Signal Processing Workstation* (ISPW), a system that contained modules for pitch-tracking sound, reverberation, pitch-shifting, harmonization, modulation, sampling, filtering and spacialization. (Rowe:89). The *Max* software was released to the public in 1991 by *Opcode Systems*. In 1996, Puckette released *PureData* as an *open source* environment that enabled synthesis and signal processing in real-time. Its signal processing capabilities were implemented into *Max* as an add-on, called *MSP* (Max Signal processing or Miller S Puckette), released in 1996.

A significant example of this approach can be found in the work developed by Casserley, who since the 1970s, had designed several prototypes for electronic instruments and vastly collaborated with saxophonist Evan Parker. On *A digital signal processing instrument for improvised music* (1998), Casserley discusses the concepts and design of his computer processing system, initially developed at *IRCAM* with the *IPSW*, presenting detailed specification and providing diagrams and *patches*.

The metaphor of the computer as an instrument that has infinite sounds poses its difficulties. A fundamental concern is related to the lack of gestural affordances by computers. This issue is twofold: on the one hand it concerns the perception of causality regarding produced sound; the computer does not have a 'body' that one can actually perceive as it is being acted upon. Performing with traditional instruments encompasses a visual component that evidences an effort, a virtuosity with every note that is played.

> The laptop musician broadcasts sounds from a virtual non-place; the performance feigns the effect of presence and authenticity where none really exists. The cultural artefact produced by the laptop musician is then misread as "counterfeit," leaving the audience unable to attach value to the experience. The laptop performer, perhaps unknowingly, has appropriated the practice of acousmatic music and transplanted its issues.
> (Cascone, 2000)

On the other hand, it relates to the need to enhance a more intricate relation between the composer and his work. As Dean asserts, the computer's standard interfaces, like the mouse and the keyboard, don't allow the multiple and overlapping possibilities for interaction that composers desire (Dean, 2009). Also, Paine argues that it is critical to research into new instruments that facilitate the subtlety and nuance that traditional istruments provide.

These environments' data flow model (*dataflow programming*) inherit the approaches of analog circuits and allow users to create blocks of code, known as *patches*, by graphically adding boxes (that represent signal generators and processors, operators, graphical controls interfaces) connected by wires.
The composer creates instruments by manipulating *patches* in real-time, feeling that he is not programming at all. Puckette refers that the notion of patching GUIs was influenced by the *Oedit system* (1987) by Richard Steiger and Roger Hale (Puckette, 2002).

The argument is that acoustic instruments "provide a set of affordances that have facilitated modes of engagement that extend to profound 'embodiment relations' that encourage expression on a highly abstract but simultaneously visceral and rewarding basis" (Paine, 2009). An interest in an *enactive* view of perception (Varela, 1991)[66] emerges, emphasizing the role of sensory-motor engagement in musical practice (Magnusson, 2009; Paine, 2009; Wessel, 2005). As developed by Varela, the mind is necessarily embodied and the cognitive function makes no distinction between perception and action.

This direction however, attempts to extend traditional music practice, deriding other aspects that the computational media provide, such as the ones explored by *The League of Automatic Composers*, for instance.

In fact, the core of computational media is a symbolic system that that establishes a hermeneutic relation with the world. In this sense, music only exists to the extent that it has been described to the machine to the upmost detail. Musical interfaces, such as MIDI controllers, keyboards, among others are mere mapping decisions.

Battier and Schnell propose the term "composed instrument" (Schnell & Battier, 2002), highlighting the fact that computers are as much an instrument as they are a score, as they can carry a vast amount compositional material to a performance. This dualistic perspective of the computer places it in a unique situation for music practice.

At present, although research institutions like STEIM continue dedicated to the investigation of gestural interfaces (Ryan, n.d.n.d.), research on this field has become much more decentralized. Many proposals can be found represented in the *New Interfaces for Musical Expression* conference series (NIME), and in recent years, ubiquitous computing has deeply contributed to interactive practices by providing diverse mobile and sensing devices that facilitate their exploration for music practice.

66 "We propose the term enactive to emphasize the growing conviction that cognition is not the representation of a pregiven world by a pregiven mind but is rather the enactment of a world and a mind on the basis of a history of the variety of actions that a being in the world performs" (Varela, 1991:9).

3.5 Interactive Music Systems and Emergence

I think it is technically possible and musically desirable to achieve a broader understanding, if not a reformulation, of what is meant by ‘interaction’.
(Di Scipio, 2003)

Interaction is often regarded as an activity that places the performer in a direct relation with the machine, as a "man/machine interrelationship" (Di Scipio, 2003) where the role of the performer is of crucial importance for the outcome of the musical work. In such interrelation, the performer sends control data to a computational system that, through algorithms, outputs a result, i.e. generates sound. The design of these systems assumes a *feedback loop* between the output sound and the performer, establishing a relation that, without the latter, would result in the halting of the system.

A distinct conception of interaction is drawn by Agostino Di Scipio and Jonathan Impett. These composers take advantage of complex systems’ theory by exploring the musical possibilities of emergent behaviours.

3.5.1 On Emergence

A musical composition constitutes an emergence because it is not reducible to its parts or elements or even to the operations that were realized during its composition. Therefore we can neglect here the word "Causality". […] The musical work will constitute itself as an emergence (a singularity), and not as a simple globality, since it manifests itself not as that which constitutes the product from the parts of a whole, or even as the epiphenomenon of its components, but as simply as that what it is, without any reduction.
(Vaggione & Soulez, 2005:55)

From a mathematical perspective, there are certain behaviours from which we cannot derive an analytical[67] answer to a problem and from which it is impossible to predict "how things tend to go" (Galanter, 2003:7), therefore, the necessity of differentiated approaches.

Systems theory is a model that explains *emergent* behaviours that result from the interrelations of its components. Derived from Bertallanfy's general systems theory (GST) and cybernetics' study on self-organization and reproduction (in particular, the work by Von Neuman), systems can be defined as a network or set of interacting agents or components, from which patterns emerge.

In this context, the notion of *emergence* refers to properties generated from the interactions of the system's components, and are not constituents of the components themselves, and eventually instantiate a new functionality on the system (Hayles, 2008:243). Cellular Automata (CA)[68] are the most widely known procedures to explain the notion of emergence, given the complex results that are achieved from its basic rules.

The term 'complex' stresses the fact that until the pattern that generates a certain result is now known, the resulting behaviour seems erratic and complex, rather posing a problem of cognition. In fact, very 'simplex' things are required to create complex behaviours and computer music has taken advantage of this.

As for aesthetics, the difficulty lies in getting valuable results. A complete formal implementation, as appealing as it might seem, does not "guarantee that such algorithms applied to music are going to make musical or aesthetic sense" (Davis, 2010) , which might

[67] Classical mechanics, for instance, provide methods that, given some properties of an object, we can determine with extreme accuracy the evolution of states of that object, through time. That object's behaviour is linear and deterministic, and the laws of classical mechanics provide an analytical solution.

[68] CA were conceived by Stanislaw Ulam and John von Neumann in an effort to study the process of reproduction and growths of form. Originally implemented on computer by von Neumann as part of his research on self-replication and reproduction, CA became known in the 1970s with Conway's Game of Life. It consists of a grid of square cells, where each of which takes a boolean value (either it is alive or dead). The rules are very simple: any alive cell that has less that two neighbours, dies of isolation; if it has two or three neighbours, survives; and if it has more than three neighbours, dies for overpopulation; any dead cell that has three neighbours becomes alive, by reproduction. From this basic set of rules, many patterns (species) emerge, such as the glider, the spaceship, the pulsar, the beehive, the blinker, among many others.

explain the lack of diversified examples for music practice. Nonetheless, many approaches have proven worthy to apply, such as the application of *cellular automata* by Iannis Xenakis (Hoffmann, 2010), *fractals* by Horaccio Vaggione (Risset, 2005), *nonlinear functions* by Di Scipio, *genetic algorithms* (Brown, 2004) or *L-systems* (Manousakis, 2009).

3.5.2 Composing Musical Interactions

> I think it is technically possible and musically desirable to achieve a broader understanding, if not a reformulation, of what is meant by 'interaction'.
> (Di Scipio, 2003)

A conception of interaction proposed by Agostino Di Scipio is described as a shift from "*interactive music composing* to *composing musical interactions*"(Di Scipio, 2002). His approach developed from the idea that mathematical methods, such as the ones found in *chaos theory*[69], could allow him to develop an *holistic* approach to music composition, in which the macro-level articulation of musical structures (*"composing with sounds"*) and the micro-level timbral properties of sound (*"composing the sounds"*) could emerge as the result of a compositional process (Di Scipio, 2001). As we have already regarded, this separation between form and material, or composing *'with'* and *'the'* sounds is a concern that continues open to discussion, partly due to acoustic compositional model continued with standard synthesis[70].

As such, in collaboration with Ignazio Prignano, Di Scipio devised *functional iteration synthesis (FIS)*, which represents a "non-standard" synthesis method with similarities to the ones developed by Brün, Koenig and Xenakis in the 1970s.[71]

[69] Originating from Henry Pointcaré's studies in 1880, chaos theory offers a diversity of processes that display chaotic behaviour. These are difficult to predict over time and are highly sensitive to initial conditions, although they are deterministic, following a strict sequence of cause and effect (Galanter, 2003:6).

[70] See Chapter 2, where standard and *non-standard* synthesis is discussed. See also (Di Scipio, 1994).

[71] The connection between Xenakis and Di Scipio, regarding granular synthesis and cybernetics is developed by Solomos (2006).

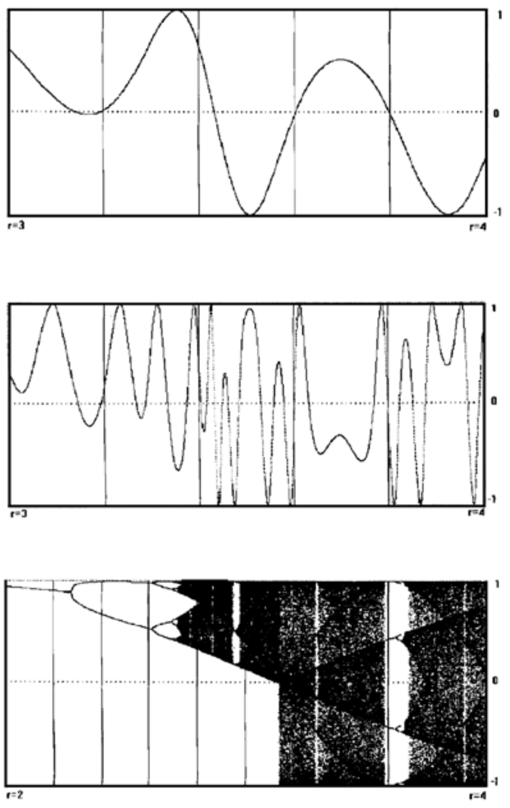


Fig. 11 Representation of FIS generated waveforms (5th, 7th and 100th iteration) (Di Scipio, 2001)

This method takes advantage of the mathematics of "chaos theory"[72] as its internal sound-generating engine, and consists of recursively applying the result of a sound function *n* times[73] (Fig. 11). The use of iterated nonlinear functions enabled him to generate a wide variety of sounds from a very limited number of parameters, ranging from very smooth curves to very intricate oscillations, with dense spectrum, eventually reaching noise. The sensitivity to initial conditions of these functions allowed him not only to generate audio samples, but also the musical structure itself, for it's unfolding in real-time.

These functions were extensively used by Di Scipio for the creation of "*Sound & Fury*" (1995-1998), a series of five works where each explored a different approach to the same process, such as tape pieces, live computer music concert pieces and sound installations.

[72] Or the study of nonlinear factors which cause simple systems to exhibit complex behaviour.

[73] A detailed description of FIS is provided by Di Scipio (2001, 2010).

Di Scipio's explorations led to the recognition that the possibilities for interaction do not lie in the human agent, by means of controlling the machine-generated events in real-time, but rather in the process of "iteration" itself. His compositional works consists of conceiving algorithms that are capable of autonomously responding to external stimulus, setting them to interact with each other. Therefore, Di Scipio proposes the term “composing the interaction”.

> This is a substantial move from interactive music composing to composing musical interactions, and perhaps more precisely it should be described as a shift from creating wanted sounds via interactive means, towards creating wanted interactions having audible traces. In the latter case, one designs, implements and maintains a network of connected components whose emergent behaviour in sound, one calls music.
> (Di Scipio, 2003)

This approach is particularly evident on *Audible Eco-Systemic Interface (*AESI) project. Evoking the cybernetic approaches carried out by Mumma and Lucier (cf. Ch. 1.5), Di Scipio explores the concept of 'ecosystem'[74] and creates a network of independent and autonomous agents linked among themselves by a number of dependency rules, from which behaviour is not strictly determined, but rather emerges from ongoing interaction. A particularity of such 'ecosystem' is that sound is the interface through which the components interact, by extracting features and low-rate control signals.

> The notion that a computer reacts to a performer’s action is replaced with a permanent contact, in sound, between computer and the environment (room or else). The computer acts upon the environment, observes the latter’s response, and adapts itself, re-orienting the sequence of its internal states based on the data collected.
> (Di Scipio, 2003)

Similarly to Di Scipio, Jonathan Impett also explores the possibilities of emergence in musical practice. Impett defines music as a dynamical complex of interacting situated embodied behaviours. These behaviours may be physical or virtual, composed or emergent. All interact in the same space by a process of mutual modelling, redescription, and emergent restructuring (Impett, 2001).

---

[74] As found in the General Systems Theory by von Bertalanffy or in the Cybernetics' negative feedback loop by Norbert Wiener.

Impett's Meta-Trumpet system consists of an instrument, fitted with physical sensors, which, similarly to Lewis' *Voyager*, uses a pitch to MIDI converter, and a computational system that processes incoming data and generates musical material. According to Impett, the aim of the system is to extend the inherent aspects of performing the trumpet to become material for algorithmic composition and the means of direct control of such system (Impett, 1994).

In this chapter we have overviewed a diversity of approaches to interactive music systems, attempting to provide a general understanding of this subject, identifying the major trends and its more paradigmatic intervenients. As computers ceased to be closed procedural devices and became connected to the real world, composers recognized the potential of sharing control them in a performative context.

A general trend has been the provision of musicianship to the computer so it can autonomously generate and play musical material in a performance. Adopting concepts from artificial intelligence, the computer has become a *creative* entity that adopts the metaphor of an autonomous performer or intelligent instrument.

Another trend concerns the improvement of the computer as an instrument, in which the development of interfaces becomes critical.

Our concerns were directed towards individual particularities that could offer new directions for our own practice, and therefore we attempted to underpin subjective directions and proposals that have been realised. In this view, intervenients such as the *Hub* or the *League of Automatic Composers* offer an enriching perspective, focusing more on the uniqueness of the computational medium and creatively exploring it beyond the established musical discourse. This perspective is also adopted by composers such as Agostino Di Scipio. Critically questioning the understanding of interaction as limited to a man / machine relationship, Di Scipio explores the notion of ecosystem and emergence, expanding his work to interrelations between software components, the environment and the audience.

Interested in such perspective, in the next chapter we will explore with more detail aspects that relate to the computational medium and its defining characteristics.

4 MEDIATION WITH COMPUTERS: CODE CULTURE

This chapter was partially published (Cardoso, 2013).

The role of computers in the arts has progressively moved away from technological issues towards social, aesthetic, political or humanist concerns. In the context of computer music, the efficiency and quality of signal processing, the representation of data structures, or the constraints in human computer interaction, progressively gave way to questions related to the creative and expressive potential of computers for artistic creation.

In this chapter we investigate diverse approaches to computation within musical practice and relate them to what is characterized as an ongoing computational shift from traditional approaches, using rule-based algorithms to interaction (Vaggione, 2001), in an attempt to get a clarifying perspective of how computation can be regarded in the context of computer-based interactive music.

4.1 Computation as production

The idea of computation as production corresponds to a view of software development that can be considered as mainly instrumental, that is, devised as a tool, helping us to do what it needs efficiently, as means to a specific end. This view implies an *a priori* conception of a function, use or purpose that prescribes specific principles, rules and tasks.

The development of software such as *Ableton Live*[75], *Reaktor*[76], *Reason*[77], or *dataflow* programming environments like *Max/MSP*[78] or *Pure Data*[79] are examples of successful applications widely used for interactive music practice that deeply facilitate musical creation. The synergies between *Ableton Live* and *Max/MSP*, for instance, have merged into a highly developed system, where the former's mixing and sequence playback, and parametric control through elaborate graphical interfaces, is extended to the latter's algorithmic composition potential and programmable filtering and processing capabilities. The domination of commercial market, with seldom exceptions within research and technology institutions or academic research, makes these technologies an "ubiquitous cultural presence that largely define[s] the nature of the music being produced" (Dunn, 1992:61).

In such context, programming languages are regarded as a relatively straightforward "*instrumentalization*" (Feenberg, 1991), first and foremost as tools for building efficient applications (Rohrhuber & Campo, 2009). This productive frame of computation evolved from the early 1970s research on human-computer interaction[80], concerned with developing means to make computers useful for general public use, creating *friendly* environments and promoting *goal oriented* applications. On *Remediation*, Bolter and Grusin develop the idea that scientists and technologists considered they were making computers more "*transparent*" and more "*natural*" (Bolter & Grusin, 2000:32). By doing so, they established the paradigm for today's computer usage in which software is presented as virtual version of real-world objects, not only in behaviour but also in visual resemblance and interaction paradigms. As Penny explains, "transparent" means that the analogy on which the software is created is foregrounded (typewriter, drawing table, piano, etc.), and the computation is hidden (Penny, 2005:55). The user forgets that one is using a computer, intuitively engaging with the

[75] *https://www.ableton.com*

[76] *http://www.native-instruments.com*

[77] *http://www.propellerheads.se*

[78] *http://cycling74.com*

[79] http://puredata.info

[80] The new machines that were beginning to be commercialized in the 1970s would only be successful if usable. Douglas Engelbart invented the mouse, and Alan Kay with the Xerox/PARC team, developed graphical user interfaces that allowed the replacement of the textual command-line interface for objects in the computer screen. The known as 'windows' interfaces could be manipulated with a pointing device, like Engelbart's mouse.

software's resemblance to the original object it *remediates*. One can execute the available commands, but the computational engine is hidden in a '*black-box*'.

This concept of *remediation* is applicable to programming environments such as the *Music-N* family or *dataflow*. The separation of the score from the synthesis engine in *Music-N* programming languages is a structural imposition that mediates the acoustic[81] paradigm, for which "non-standard" synthesis can represent a counter-example. The *dataflow* programming languages, like *Max/MSP* or *Pure Data*, are graphical interfaces that, down to a certain level, open the computational engine, enabling the construction of patches from a set of objects to be connected[82]. Nonetheless, the structure of the engine itself is not open (Smith & Wakefield, 2009b). Creating patches, rather than actual programming, resembles the logic of operating with analog electronics, by treating sound as an undifferentiated stream of data that travels through a circuit to be set.

> Computer software whose interface is designed in this manner frees the user's attention for domain-related activities. Rather than having to think in terms of the system s/he is using, the user can remain focused on domain-related concepts, thus freeing attention for domain-centered activities. The user forgets that s/he is using a computer, with its large array of I/O requirements—all of the complexity is under the hood, so to speak. The computer, as such, effectively disappears in its use.
> (Hamman, 2002)

As such, software imposes an interaction mode restricted by the available *commands* and structures, making this *transparency* a possible constraint for creative expression. By adopting software that entails pre-defined solutions to a series of problems, the composer accepts the tools as they are given and appropriates them in his compositional activity. As Herbert Simon observes, "solving a problem simply means representing it so as to make the solution transparent" (Simon, 1996:132). [83] Accordingly, there is the belief that technology is

---

[81] See discussion in Chapter 2.2, regarding Mathews' *UGens.*

[82] Smith and Wakefield recognise that this modular flexibility is the condition upon which *dataflow* languages draw, by starting up on a blank canvas with open-ended possibilities to connect and encapsulate modules (W. Smith & Wakefield, 2009b).

[83] Even today's lower-level programming languages, like C++, are developed in integrated development environments (IDE) with many affordances, such as code editors, debugging tools, build automation tools, dynamic documentation and helpers.

‘neutral’ (Veak, 2000),that it provides the most efficient solution to a problem that is “technical” and that it improves over time, where each new released version of *software* solely introduces more and better functionalities.[84]

As a consequence, there is the idea that composers do not need to understand technology’s inner workings. In this sense, Trevor Wishart alerts us to the fact that the easiness to generate satisfying sounds can lead to a culture of ‘*neophilia*’; meaning that, artists cease to explore the medium due to the market’s cycle of constant introduction of new products (making past ones obsolete) in order to open up selling opportunities (Wishart, 2009). Even more critical, and directed to the computer itself, is Friedrich Kittler, who argues that some elementary functions that are essential for the computer to work are recorded in *the hardware*, being inaccessible to exploration under the argument of safety (Kittler, 1995). This already represents a domestication of computers, enclosing its creative potential.

This view of computation as production tends towards a model of interaction with software that has proven efficient and fruitful, as evidenced by the vast and diversified offer of products in the market. However, we can also identify trends that attempt to provide a more adequate support to artistic practices that, beyond a mere productive frame, promote an exploratory approach to computation as a creative medium.

4.2 Computation as Creative Medium

An expanded understanding of computation in the context of artistic practice is evidenced through a renewed interest in the potential of programming languages. Since the late 1990s, artists engage in programming activity, motivated by experimenting approaches that are not available in commercial tools. New environments were created, such as *SuperCollider*[85] (McCartney, 1996, 2002), *Processing*[86], *openFrameworks*[87], or *ChucK*[88].

[84] The notion of *legacy support.*

[85] http://supercollider.sourceforge.net

[86] https://processing.org

[87] http://www.openframeworks.cc

Most of these were developed by artists with computational knowledge, motivated to support their own work (McLean, 2011:116)

These approaches bring to the fore an understanding that by making code's inner entities and structures accessible to the artist, an expressive potential is opened-up and common or preconceived schemas are avoided. These structures function as a flexible and abstract modular kit from which composition specific material is constructed.

Programming ceases to be regarded as a technical activity, with ends to the creation of user applications (Rohrhuber & Campo, 2009), but rather an environment to explore creatively, directed towards the prototyping of ideas, developing artworks, performing music, creating artefacts for a diversity of uses and contexts. The act of programming becomes a way to explore the possibilities at hand, to gain a better understanding of the problem itself, through the *prototyping* of ideas in a familiar environment. As such, the *design-then-do* production logic[89] is replaced by *dialogic* approaches[90].

In this context, the work is developed with no *a priori* planning, and each step of the programming activity is evaluated before proceeding to the next, in a *feedback loop* acting and reacting to the perceived.

While establishing the relation between this *dialogic* mode of programming and other artistic practices, we can recall the experiments with painting carried out by Kandinsky and Klee, or even Pollock's action-painting. Sherry Turkle uses the term '*bricoleur*' as a metaphor to describe this dialogic practice:

---

[88] http://chuck.cs.princeton.edu

[89] This approach consists of "working within a rule-driven system that can be mastered in a top-down, divide-and-conquer strategy, as taught in the Harvard programming course" (Turkle & Papert, 1990:136).

[90] The term *dialogic* is used in reference to the regulatory *feedback loop* that occurs between human and machine, as defined by cybernetics (Wiener, 1948).

The bricoleur resembles the painter who stands back between brushstrokes, looks at the canvas, and only after this contemplation, decides what to do next. Bricoleurs use a mastery of associations and interactions. For planners, mistakes are missteps; bricoleurs use a navigation of mid-course corrections. For planners, a program is an instrument for premeditated control; bricoleurs have goals but set out to realize them in the spirit of a collaborative venture with the machine. For planners, getting a program to work is like "saying one's piece"; for bricoleurs, it is more like a conversation than a monologue.
(Turkle & Papert, 1990)

So far, we have outlined two distinct modes of relating to computation. On the one hand, a *production* oriented practice, in which one engages the programming activity with a very defined idea of the problem at hand, and of the desired output. On the other hand, through an experimental approach, uncertainty becomes a desirable characteristic and the computational medium is used as a fundamental part of the creative process, in a closer affinity to artistic creation.

4.3 Interactive Programming

We would begin by writing an initial algorithm that captures a rough imagination, a conjecture of how the sound could be characterized. Then we would modify this description until it became, possibly in a surprising moment, a sudden realization of something that evokes a memory of that particular sound. The surprising moment is not so much the result of a random coincidence, but of the way in which program-text, synthesis process, sound and perception interact.
(Rohrhuber et al., 2007)

The exploratory approaches to computation that we have described are strengthened by the resurgence of interactive programming environments, opening up the possibility of writing and compiling parts of a program in real-time, while it is executing, hence replacing the traditional production steps of writing, then compiling, and finally using the software.

This approach became viable by using "just-in-time" (JIT) compilation,[91] an implementation technique that can be found as early as the 1960s. Languages like *Smalltalk* (Kay, 1993), *Self*, *Basic*, *LISP* or *FORTRAN* used *JIT* compilation as a means to improve the time and space efficiency of programs, although it was regarded as completely unnecessary, given the context in which languages were used. For music practice, however, the relation to time is particularly relevant. By removing the compiling time, i.e. the temporal delimitation between development and action, one is able to change not only the parameters of the music program, but the program itself, at runtime. Such changes bring a new and desired situation for electronic music, in which the description of sound, or notation, is immediately perceived as sound. This is particularly relevant for performance as it opens up new expressive possibilities, being no longer constrained to merely setting parameters of a preconceived program. Another fundamental change is related to the fact that the computer's state (stored properties and methods) is maintained when new parts of the program are executed, since there is no necessity to restart it and reset the program to its original state.

> Extending the capacity of a program to be redefined at run-time embodies a shift away from the notion of computer as a bounded tool for a bounded task toward the notion of an always-on workspace or environment. Run-time augmentation is thus relevant to composition in general, since radically decreasing the latency and maximizing the overlap between action and perception may increase interactive fluidity and reduce conceptual load.
> (W. Smith & Wakefield, 2009a)

The use of interactive programming environments for music practice dates back to the early 1980s, among composers who wanted a means to facilitate the rapid realization of musical ideas, in "a dialogic creative process emblematic of an improviser's way of working" (Lewis, 2000). The programming language *Forth*[92] was attractive to these composers, and became the most widely used language for interactive music. Tim Perkins, for instance, describes that the rehearsals of *The Hub* involved the continuous modification of their code in real-time, without shutting down the musical network interaction, although code would be stable for performances (Tim Perkins qtd. in Rohrhuber, Collins, McLean, & Ward, 2003).

---

[91] For a brief history of *just-in-time* computation see (Aycock, 2003).

[92] *Forth* is a real-time programming language developed by Chuck Moore, in the 1970s, with the intent to allow more direct user and machine interaction. See http://www.forth.org/.

The first documented live-coding experiment, was carried out by Ron Kuivila in 1985, using his *Formula*[93] programming language (Collins, 2011). This interactive approach is also present in the *data-flow* programming environments, enabling the switching between *edit* and *use* modes in the construction of *patches*.

In the beginning of the Century, *live coding* practices re-emerged, taking advantage of interactive programming approaches for the development of live computer music and visual performance, by writing live algorithmic compositions, while using code as a conversational medium (Rohrhuber et al., 2003; Ward et al., 2004)[94].

As Rohrhuber points out, in spite of the many antecedents, the appearance of live coding was not self-evident at first. In the implementation of *SuperCollider* for instance, James McCartney[95] "decided to keep the '*interpreter'* running during sound synthesis without mentioning it in the release notes" (Rohrhuber & Campo, 2009). Nonetheless, interactive programming practices in the musical context became more consequent as computers became faster and more capable of generating sound in real-time.

One of the earliest examples of such practice is the *PowerBooks UnPlugged*[96] ensemble, created in 2003. They use laptops as complete musical instruments, limited by their default configuration. *Built-in* speakers are used to output sound and wireless connection is used to share algorithms and distribute sounds over any other computer that is accessible in the network, creating a differentiated spatialization situation. They also take advantage of the computer's autonomy to move through the performance space, mingling with the audience. The *Just-in-Time* library (2011), for *SuperCollider*, was developed by Rohrhuber in order to improve the ongoing interaction with the computational model of

[93] A programming language based on Forth

[94] See TOPLAP (The [Temporary | Transnational | Terrestrial | Transdimensional] Organisation for the [Promotion | Proliferation | Permanence|Purity] of Live [Algorithm | Audio | Art | Artistic] Programming).

[95] The author of SuperCollider.

[96] See http://pbup.net/s/. *PowerBooks UnPlugged* is formed by Julian Rohrhuber, Alberto de Campo, Echo Ho, Hannes Hoelzl, Jan-Kees van Kampen and Renate Wieser.

sound. Also, this multi-user live coding practice led to the creation of the *Republic* system for sharing and modifying *codelets*[97] through the network.

Another early example is the *Slub*[98], a collective that "shared a desire to make music and enthusiasm for programming, and resolved to combine them" (McLean, 2011:138). Since 2000 they have developed their own software for performance, considering themselves programmers who make music. Examples of the software they have developed are *Feedback.pl* (McLean, 2004) , a live coding environment that self-modifies its own source code in order to maintain the state of the program in the process of compilation; *Tidal*, a language designed for live coding of musical patterns (McLean, 2011:79); *Texture (Fig. 12)*, a visual programming language based upon the *Tidal* language (2011:107); or Griffiths' game-like live coding environments for music, such as *Al-Jazeri* (Collins, 2011).

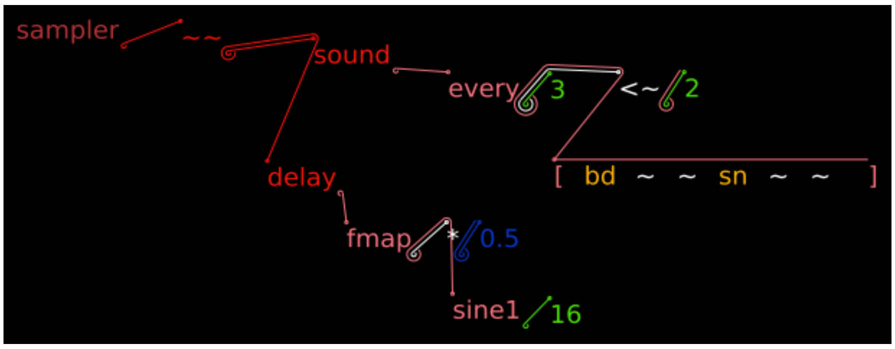


Fig. 12 *Texture* visual programming language by Alex McLean

[97] Rohrhuber & Campo use the to describe little synthesis programs (Rohrhuber & Campo, 2009). See also Hofstadter's definition (Hofstadter, 1995:105).

[98] Formed by Adrian Ward, Alex McLean and later joined by Dave Griffiths.

4.4 Rethinking Code

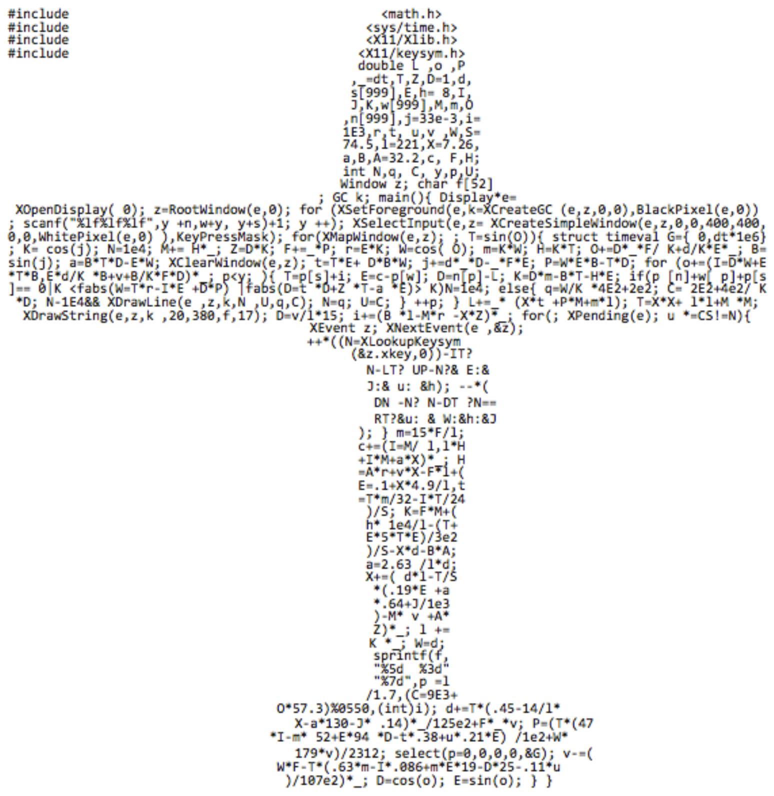
"All writing is 'right': it is a gesture of setting up and ordering […] ideas"
(Flusser, 2011:6)

The understanding that humans become a fundamental part of the computational environment, as established with interactive programming, suggests an understanding of code that does not coadunate with the traditional conception of computation by algorithms but, rather, as part of language in general, opening the domain of programming to considerations within social sciences. The emergent field of software studies, for instance, attempts to address code beyond its mere functional dimension, recognizing its social, political and aesthetic dimensions (Berry, 2011; Fishwick, 2006; Marino, 2006)

In the context of software art we find a diversity of explorations of aesthetic features of code. *Perl poetry*[99], for instance, is primarily directed towards human interpretation, although written with code. *Obfuscated code*[100] develops on indecipherable source code that is expected to run in some sort of surprising way (Fig. 13).

[99] For details on the Perl language and its afinities to latural language see Cox & Ward (2008)

[100] Regarding aesthetic explorations of code of see (Montfort, 2008)



```
#include                                           <math.h>
#include                                         <sys/time.h>
#include                                         <X11/Xlib.h>
#include                                        <X11/keysym.h>
                                                double L ,o ,P
                                               ,_=dt,T,Z,D=1,d,
                                               s[999],E,h= 8,I,
                                               J,K,w[999],M,m,O
                                              ,n[999],j=33e-3,i=
                                              1E3,r,t, u,v ,W,S=
                                              74.5,l=221,X=7.26,
                                              a,B,A=32.2,c, F,H;
                                              int N,q, C, y,p,U;
                                             Window z; char f[52]
                                           ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                          XEvent z; XNextEvent(e ,&z);
                                         ++*((N=XLookupKeysym
                                               (&z.xkey,0))-IT?
                                                 N-LT? UP-N?& E:&
                                                 J:& u: &h); --*(
                                                  DN -N? N-DT ?N==
                                                  RT?&u: & W:&h:&J
                                                ); } m=15*F/l;
                                                c+=(I=M/ l,l*H
                                                +I*M+a*X)*_; H
                                                =A*r+v*X-F*l+(
                                                E=.1+X*4.9/l,t
                                                =T*m/32-I*T/24
                                                 )/S; K=F*M+(
                                                 h* 1e4/l-(T+
                                                 E*5*T*E)/3e2
                                                 )/S-X*d-B*A;
                                                 a=2.63 /l*d;
                                                 X+=( d*l-T/S
                                                  *(.19*E +a
                                                  *.64+J/1e3
                                                  )-M* v +A*
                                                  Z)*_; l +=
                                                 K *_; W=d;
                                                  sprintf(f,
                                                  "%5d  %3d"
                                                  "%7d",p =l
                                                 /1.7,(C=9E3+
                                     O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                                       X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                                    *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                                       179*v)/2312; select(p=0,0,0,0,&G); v-=(
                                     W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                                      )/107e2)*_; D=cos(o); E=sin(o); } }
```

Fig. 13 winning entry in the 1998 ‘International Obfuscated C Code Contest’ (IOCCC), a flight simulator written by banks

Other examples can be found in the creation of *weird* or *esoteric* programming languages[101] such as *INTERCAL*[102], in which, for a certain proportion of statement lines of the code, one has to be polite and include terms such as '*PLEASE DO'* in order for the program to execute.

The *Quoth*[103] (Fig. 14) music environment, by Craig Latta, uses natural language commands to describe and manipulate objects, recalling Joseph Weizenbaum's *Eliza* (1966).

[101] http://esolangs.org/wiki/Timeline_of_esoteric_programming_languages

[102] INTERCAL (Compiler Language With No Pronounceable Acronym) was developed by Donald Woods and James Lyon in 1972.

[103] http://netjam.org/quoth

```
- "Where am I?"
  "You're in the library," replies the note.
- "Play 4 times."
  "You lost me at '4'," replies the note.
- "I want to teach you to 'play 4 times'."
  "Great! Welcome to my mind," replies the note.
```

Fig. 14 Example of a session using Quoth, by Craig Latta

These examples illustrate the potential of regarding code from a linguistic perspective, reinforcing the notion of code as something subjective and malleable, plastic.[104]

> To whom are these people writing? For they are not writing past a conclusion to another human being. Rather they write with and for the apparatuses. […] It is another writing, in need of another name: programming.
> (Flusser, 2011)

On the one hand, the simplification of language to binary codes simulates the structure of our nervous system. The *Boolean* algebra converted to voltage streams of *ons* and *offs* represents nerve synapses on an artificial body that can be extended by connecting peripherals, such as sound cards, displays and control devices. On the other hand, the *Turing Machine* (Turing, 1950) is the minimal construct that allows us to consider code as a an abstraction away from the machine that forces a start-from-zero into most basic mathematical and logical *if-then* propositions (Mcwilliams, 2009).

From machine language to assembly and over layers of bootstrapped compilers, we end up with computers that are capable of manipulating symbols. Instead of looking at computers as programmable tools, in which we operate with their symbols in order to instruct, we can regard them as structural elements of language, that, just like words, constitute the materials to project ideas. In this sense, code is not just a medium. It offers the

[104] Regarding code from a linguistic perspective, see also (Berry, 2011; Cramer, 2005; McLean, 2011; Wardrip-fruin, 2006)

possibility for these ideas to materialize into action: they are '*executable statements*' (Cramer, 2005).

From this viewpoint, programming can be regarded as a peculiar kind of writing, directed towards both human and machines, as if it was a conversational act in the everyday sense.

The live-coding practice of the *PowerBooks Unplugged* ensemble illustrates this idea of coding as a conversational activity. By broadcasting *codelets* through the network of computers, performers and machines are both indispensable for interpreting, transforming and executing those texts in order to create music.

A relevant aspect within interactive programming is that we are not merely creating artifacts, but rather maintaining a discourse with a computer system. Starting with the *default* objects and functions provided by the programming language, we progressively extend our scope of action, describing new *entities*, with new *properties* and *methods*, and creating worlds with their own rules[105].

As early as 1979, Terry Winograd had already recognised that computers are not primarily used for problem solving, but are instead "components in complex systems" (1979). On "Understanding Computers and Cognition" (1986), Winograd and Flores further describe computers as designed in language and equipment for language, consolidating the link between programming and the issues of knowledge representation undertaken by cognitive science.

> [Computers] will not just reflect our understanding of language, but will at the same time create new possibilities for the speaking and listening that we do-for creating ourselves in language (Winograd & Flores, 1986:102)

They argue that we get a very distinct perspective on computers by regarding them as plastic structured entities that are shaped by interaction, and by considering that the programmer and his medium form an inseparable complex that can be regarded as

---

[105] Such perspective can be understood by the notion of 'language game', developed by Ludwig Wittgenstein on *Philosophical Investigations* (Wittgenstein, 1958), where language is viewed as a cultural activity characterized as the act of explaining the meaning of a symbol through action, by pointing to real-world situations (Wittgenstein, 1958).

autopoietic (Winograd & Flores, 1986:102).This term is adopted by Humberto Maturana and Francisco Varela (1980), for describing as a quality of living systems, in the sense that they are not only *self-organized* but also *autopoietic*, or *self-making*, informationally closed and self-referential, where knowledge cannot be transmitted from the environment, but rather constructed by the system itself, taking place by variation-and-selection or trial-and-error.[106]

We therefore identify a renewed interest in considering the act of programming as a natural extension of cognitive processes, emphasizing the notion that the programmer conceives a certain reality or world by using the computer to mediate his thought, by designing processes, structures, possibilities for action – thus programming as an act of *self-reflection* and *self-making*.

This view holds great similarities to the requirements of realtime interactive composition.

The program must easily adapt to the composer's dynamic decision-making; it must be regarded as a mechanism with which to interact, not a mathematical abstraction which can be fully characterized in terms of its results. Also, as Rohrhuber and Campo asssert, by performing with code one finds limitations on the available time for typing and on the cognitive complexity of algorithms; Also, the system must be efficient in order to allow realtime evaluation of such code, and be capable of handling errors gracefully (Rohrhuber & Campo, 2009).

[106] For a summary on cybernetics see also (Heylighen & Joslyn, 2001).

Winograd and Flores' remarks have been taken into consideration, as we can regard with the renewed interest in Prototype-Based Programming[107] and with the emergence of new generation programming languages such as *Perl*, *Python*, *Ruby* or *Javascript*.

At present, a broader view of computation that takes into consideration the social, aesthetic significance of computers is becoming more evident in a wide variety of fields.

The concerns are less in building software applications and turn essentially to the exploration of artifacts "that act, that move, that work" (Brooks, 1987). The recognition of interaction in computation is also the source of ontological concerns in computer science, as presented by Goldin and Wegner in Interactive Computation (Goldin & Wegner, 2007; Goldin, Wegner, & Smolka, 2006).

[107] Attempting to support the notion of computers as complex interactive systems, Winograd proposed the application of the concept of *prototype* (Hofstadter, 1979:358; Wang, 2008; Winograd, 1979:397; Winograd & Flores, 1986:115) in programming languages, implemented by Henry Lieberman (see also Harkins, 2011; Lieberman, 1982) for his *Act1* programming language. OOP *inheritance (*Every *instance* of a *class* shares a common behaviour implementation) is criticized under the argument that, since instances exist, there is no need for the *class* / *instance* duality. Instead of using classes, one uses *prototypes*. With *classes*, the defined behaviours are only valid within its instances, as a closed system. In order to extend its behaviour one would resort to *inheritance*, extending the *class* from which the object would have to be instantiated.
With *prototypes*, by creating a new object, the concepts of behaviours are *cloned* as default, taking advantage of the general knowledge encoded the original one. The advantage is that these objects can still reuse concepts from others. However, through *delegation*, an object can borrow concepts from an external one. Therefore, each object becomes a dynamic and subjective unique entity. Prototypes change the ontological relation with programming allowing a more dynamic discourse between the programmer and his software.
Also, due to the fact that the objects are dynamic entities, methods always accept unknown entities as their arguments and try to operate with them. If the method fails to perform the operation, it usually returns an error without halting the system. With *prototypes* this is not assumed as a programmer's error, a fault within the program, but rather a trial and error approach in the discourse between entities of a system.

> These compositional instances, to reiterate, are not envisaged here in the frame of the traditional approach to algorithmic (automatic) composition: they are instead seen in the light of the ongoing paradigm shift from algorithms to interaction, where the general-purpose computer is regarded as one component of complex systems (Winograd 1979), and where the composer, being another component of these complex systems, is imbedded in a network within which he or she can act, design, and experience concrete tools and (meaningful) musical situations.
> (Vaggione, 2001)

Computer music does not need to be reduced to established general-domain approaches. Acknowledging the constraints of viewing programming languages as mere tools for the production of artifacts, with a preconceived use or purpose, a distinct perspective can be explored by regarding computation as an expressive medium to engage in a creative activity.

The exploration of new directions and territories that are being carried out, attest that the act of programming as a cultural activity, where code is regarded as something aesthetic, subjective, linguistic, dialogic, conversational, as an extension to knowledge representation. In these terms, we are given the possibility to integrate how machines "act and perceive into our own sensory experiences and creative processes", where becoming machine becoming human "will eventually look less like a hybrid and more like a united whole" (Chayka, 2012), a promising potential for computer music practice.

5 THR44: AN APPROACH TO INTERACTIVE MUSIC

Music before everything else,
and, to that end, prefer the uneven
more vague and more soluble in air
and nothing in it that is heavy or still.
(Verlaine's Art Poétique, late 19th Century in Eco, 1989)

In this section, we present the musical outcomes and the software (*Thr44)* that resulted from this research. The purpose of this work consisted in discovering methodologies that could contribute to the improvement of interactive music practices. We begin by addressing some aspects regarding the context of its creation, such as personal considerations and circumstantial factors, and by defining the objectives and criteria that guided our work. We will then proceed by presenting the musical context within it was developed, and thereafter we discuss the implementation details of *Thr44*, focused on the possibilities that it can provide to interactive music practice. We will conclude this chapter with a summary on our findings.

5.1 Possibilities for Composing Interactive Music Systems

In the previous chapters, we observed that the use of computers for music creation offers a wide variety of possibilities for the representation of musical concepts, their sonic expression and control. We have attempted to identify fundamental issues and prospects related to the usage of computers, in particular within the context of interactive music systems.

These systems are not conceived in view of traditional approaches to compositional practice. Such approaches are directed towards the creation and interpretation of musical works in a restrictive sense (Blum, 2001) that are fixed and finished. These systems can rather be described as *activities*, as processes encoded in technologized musical environments that are to be carried out during their presentation, resulting in one particular expression among a vast field of possibilities. These works are variables, defining a framework for action (Ribas, 2011).

> The activity of composition has changed from the production of works to the construction of technologised musical environments in which music happens, and that emphasise music as a situated and interactive occasion rather than something that lends itself necessarily to repeatability and mediated reproduction. […The] focus of creative work is shifted from the production of reproducible sounding works to the construction of objects and environments in which music, or sonic art, is facilitated.
> (Green, 2006)

Barry Truax, in 1976, described distinct levels at which the composer can operate with interactive music systems (Truax, 1976).[108] Within this context, musical creation can be described as an activity that extends towards a multi-disciplinary practice that entails a diversity of modes by which one relates to the computer.

One of such modes is related to the creation of low-level entities and structures that enable the computer to operate with musical concepts: *meta-composing*. As we have previously discussed, the recognition that the choice of software deeply determines the aesthetic possibilities of artistic creation has led to an increasing interest from artists into the development of their own systems. This tendency is evidenced by the growing amount of open-source interactive music environments and respective libraries that have emerged in the past decade. In this perspective, we reiterate the relevance of programming languages as a defining aspect of the creative process within music practice.

[108] The following levels are described by Truax (1976): at the *sonic* level, one specifies the structures and models that allow the creation of acoustic material; at the *syntactic* level, one composes with the sounds, articulating the relations between the sonic material; at the *semantic* level one evaluates the well-formedness of those structures; finally, at the *pragmatic* level one interacts with the system, registering the user behaviour.

A second mode lies in the creation of actual sonic entities, taking advantage of previously created structures in order to describe the sound material ('composing the sound') and its musical articulation ('composing with sound'), i.e. this mode consists of developing the means by which musical material is worked, in order to develop overall musical form taking smaller units and components. Such work relates to sound design (synthesis and signal processing) and algorithmic composition.

A third mode is directed towards the development of strategies for interacting with these sonic complexes, which ranges between two contrasting approaches. At one end, drawn from a procedural dimension of music, one can develop generative processes to be used in performance, resorting to Markov chains, graphs, genetic algorithms, non-linear functions or any other strategies that permit sonic activity. At the other end, one can use the computer as a virtual instrument and impose a direct control of sound. In this view, human-computer interaction (HCI) provides us with a diversity of models that facilitate such relations.

A last mode is the moment of presentation of the work. At this level, the compositional activity can be intertwined with performance, open to distinct communicational possibilities and the exploration of values such as spontaneity or expressiveness.

These distinct modes support the idea that composing computer-based interactive music must be regarded as a multi-disciplinary practice that opens up a vast field of possible directions to investigate.

In the next sections we will proceed with this research by drawing the paths we consider more significant for the improvement of our practice in the context of interactive music.

5.2 A personal perspective

The technical and compositional decisions that are made are strongly biased by personal and aesthetic attributes that must, therefore, be exposed. Our musical interests are placed within the context of experimental electronic and electroacoustic music, with an emphasis on improvisation. In fact, the musical creations we have developed prior to this study consisted of performing with laptop computer, using Max/MSP to process the sound of other performers, or integrating and playing double bass to generate our own material.

Concerning the aesthetic nature of sonic material, we are interested in the spectral and textural qualities of sound, its varieties of motion and fluctuations in time, its possibilities for timbric manipulation, being less concerned with the "note paradigm", in which pitch and amplitude are regarded as primary parameters (Boulez, 1987). As we will notice in the following sections, the sonic materials and processes that are privileged in our work are linked to *microsound* and develop from sampling acoustic instruments, or generating small synthesis fragments of noise, pulses, frequency and amplitude modulation synthesis, applying granular synthesis, and recursively processing the generated and sampled material.

Another aspect that informs this work is the recognition of its *acousmatic* character, in the sense that, in our performative work, we do not make an effort to expose the sources or causes of sonic events through visual or action cues. Rather, they can sonically remain mysterious and ambiguous, subjected to the auditory perception of the audience. As Smalley asserts, sonic events can be perceived as surrogates of such cues, relating to a range of exeriences outside the context of the work, whether explicit, implicit or remote from reality (Smalley, 1997).

Additionally, and framed within an improvisational context, the confrontation of our decision-making with a complex dynamic of extraneous constraints and affordances is critical. A fundamental aspect lies in the possible relationships that can be established with other musicians, which in the majority of situations encompasses their aesthetic choices, instruments, techniques and conceptions, sometimes only known in the act of live performance.

Another aspect is related to the resources and conditions that are inherent to the place where the performance occurs. Libraries, galleries, universities, cultural associations, bars, among others, impose physical and acoustic constraints to the performance outcome.

And finally, one needs to consider the social and cultural context, in particular the discourse that is intended with the audience – the communicational character of this discourse can be prior outlined, but it is the relation that evolves *in situ* that becomes critical for the development of this work.

These aspects offer an enriching opportunity for those who are interested in the possibilities of computer music in the context of experimental and improvisatory practices.

5.3 Composing Interactive Music Systems

In the previous chapters, we have surveyed the possibilities offered by computers for music creation, recognising that one of the promising approaches to interactive music lies in the actual creation of computational systems themselves. In this view, rather than using existing general-purpose tools, and recognizing that software is highly determinant for the aesthetic, artistic, musical outcome, we assert that by dwelling with programming languages we can develop a more subjective music practice.

Regarding the creation of interactive music systems, a common approach lies in a two-step development where the design of the system is planned beforehand, i.e. composed, and then performed with a fixed set of parameters or controls. Such approach draws a separation between *thought and action* where composition is viewed as making provision for the interactions that can be carried out in performance (cf. Ch.4.1).

A distinct approach has been recognized in the practices of *live-coding,* established as the research of the potentialities of algorithms as a means to express mental ideas (cf. Ch. 4.2), as a conversational medium, avoiding "safety nets" of previously written snippets of code. These practices have taken advantage of interactive programming and *JIT* compilation. The ability to change a program while it runs, and not only its parameters values, enables the composer to establish a less mediated relation with the program itself. The programming language becomes part of the performance activity itself, the main interface.

This approach enables the expansion and flexibility of decision-making process in the context of live performance, however one must be cautious about its constraints. Recalling the arguments provided by Rohrhuber, in the act of writing code during performance one is limited to the time available for typing and by the cognitive complexity of the algorithm being written, as well as its consequences as it is interpreted by the computer (Rohrhuber & Campo, 2009).

Instead of assuming one of these opposing approaches, one would rather gather the strongest points from each. In this perspective, one can consider that by developing a computational framework, one would be able to continuously formalize and mediate compositional ideas. Such system does not need to be timely constrained within the moment of performance. Interactive programming paradigms enable us to develop a continuous dialogue with a growing computational system, whether in a work exposition or in a studio-like setting. We believe that such direction would allow a more dynamic and flexible context, not only expanding the possibilities of decision-making in performance, but also taking advantage of the complexity of generative approaches and of the expressiveness of physical or graphical user interfaces, that require long development cycles to be tested and implemented.

5.4 Objectives and Criteria

Based on the observations in the preceding sections, our objective is oriented towards the creation of sonic works, or activities, that result from the development and use of the computational medium. The purpose of this work is to design and experiment diverse possibilities in order to assess how one can compose interactive music. In particular, by testing what strategies and concepts enable one to improve compositional practice in an interactive context. By improvement we mean facilitating the musical decision-making process in a diversity of contexts such as a performance, installation or studio composition; that is to say, to be able to engage in a creative process, expanding one's inner concerns, without being limited to pre-defined compositional strategies imposed by pre-existing software.

In order to fulfil such objective the following criteria have been taken into account:

Versatility

By versatility we mean easiness in the description of compositional ideas. The system must have a representational potential that, on one hand, does not impose preconceived models (for instance, limiting sound description to the use of Unit Generators), and on the other hand, facilitates the expression of musical ideas. From this perspective, the system must provide a variety of components (high and low level), and their usage must have as little accidental complexity as possible, so one is not concerned, for instance, with memory pointers or disk addresses, but rather with the musical possibilities that can arise from interacting with those components.

Transparency

One must not only be capable of describing and executing processes, but also be able to recover and perceive them; to understand their notation, their inner structure, behavior and interconnectivity.

Flexibility

Directly related to versatility and transparency, flexibility expresses the system's susceptibility to modification or adaptation. Taking into account our aesthetic context and the diversity of situations that can emerge in our practice, it is required that the system has the ability to adapt with the minimum possible effort and time. It must be flexible, enabling one to dynamically create, edit, and maintain it.

Expressiveness

We privilege interaction in order to extend compositional decisions to the act of performance, assuming that the ability to convey intentions or emotions is of particular relevance. We are not only concerned with the description of musical ideas, but also with the means to communicate them.

We assume the subjective nature of the present work, and therefore, there is no intention in presenting it as an application or a tool *per se*. Rather, the objectives of our work are to identify and experiment strategies that could improve our own compositional practice, and therefore we are more interested in evaluating the application of these strategies in our own musical practice. As such, criteria of efficiency, general usability or pedagogy are not considered of relevance. Nevertheless, this work is publicly available and we have extensively used it, allowing us to assess the correspondence to the criteria designated in this section.

As discussed in the introduction of this study, we have adopted an action-based methodology and therefore the musical activity we engaged in provided the grounds and evaluation context from which we iteratively defined future objectives to carry on with our research.

5.5 Musical Activity

We succinctly describe the ideas and principles we have developed in our musical activity as we assume they will provide some insight on the implemented framework and its principles.

5.5.1 Variable Laptop Orchestra

The first performances that were carried out within the context of this research were as a member of the *Variable Laptop Orchestra* (VLO) from CITAR[109] (Centro de Investigação em Ciência e Tecnologia das Artes), a laptop ensemble constituted by the members of this study's research programme[110]. Its main goal was to create an ecletic environment for CITAR researchers to experiment and confront their ideas. This collective was very diversified in terms of their approaches to music practice and backgrounds, raging from experimental and improvisation-based musicians to composers more concerned with written pieces.

[109]. http://artes.ucp.pt/citar/

[110] André Rangel, Joana Gomes, João Cordeiro, Miguel Cardoso, Pedro Patrício, Ricardo Guerreiro, José Luis Ferreira and Vitor Joaquim.

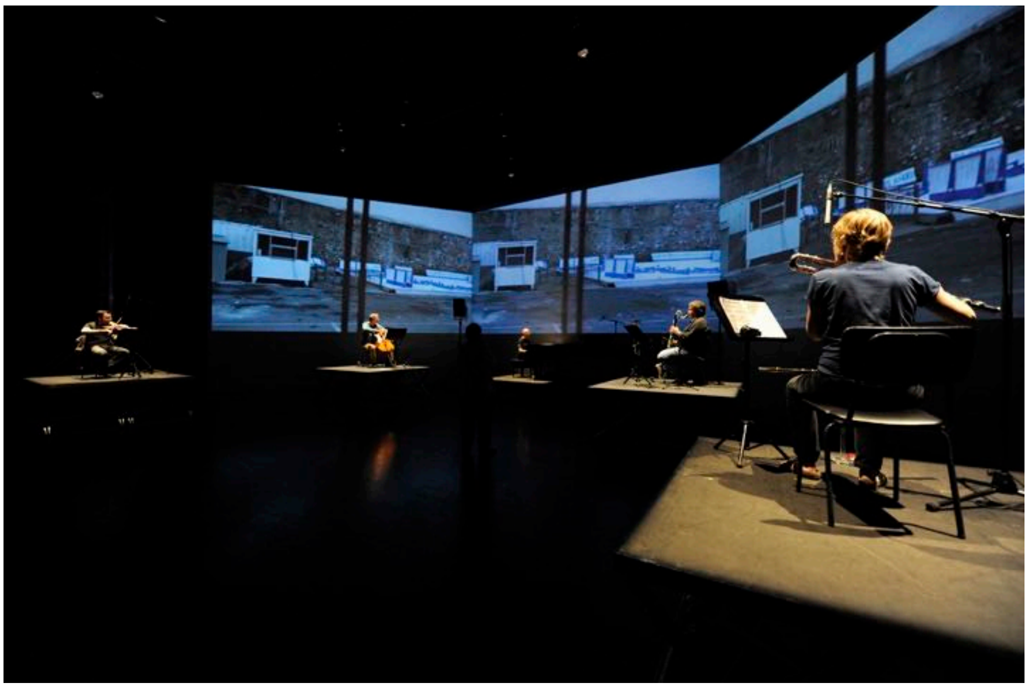


Fig. 15 Transfronteiras

One of the first evident trends was the possibility to work beyond the context of improvisation. In this perspective we have collaborated with José Luis Ferreira on a musical piece titled *Limits / Capacities* (2008), having performed at "Isto não é um concerto", an event that was produced by Paulo Ferreira Lopes at Centro Cultural de Belén[111]. We have also performed at "Transfronteiras", in CulturGest[112] (Fig.15), an event that was produced by José Julio Lopes and Orchestrutópica[113], in which we collaborated with Paulo Ferreira Lopes on his composition *De Profundis* (2006).

For such events, our role was to explore electronic sounds over scored instrumental sections. Although we were improvising, we had already a defined notion of what the instruments were going to play, allowing some advanced planning. Also, the collaboration with the composers permitted some insight on the concepts, techniques and aesthetic concerns they manifest, providing some sense of direction.

[111] http://www.ccb.pt

[112] http://www.culturgest.pt

[113] http://www.orchestrutopica.eu

We were interested in exploring signal processing techniques and in developing means to empower the acoustic instruments by extending their sound beyond their natural limits, a direction that was ideal for these pieces. In our actions, we created low frequency melodic phrasing, excited the harmonics of the acoustic instruments or sustained the notes far beyond the instrument's capacity; In more tense sections we multiplied the instruments sounds, creating voicings that would imitate their source's original articulation and then dissolve in conformity with the note's resolution.

VLO had also the opportunity to perform with Carlos Zíngaro at the Black & White Festival, held at Universidade Católica Portuguesa. A recording of this event is provided in the accompanying media device (cf. Appendix 2).

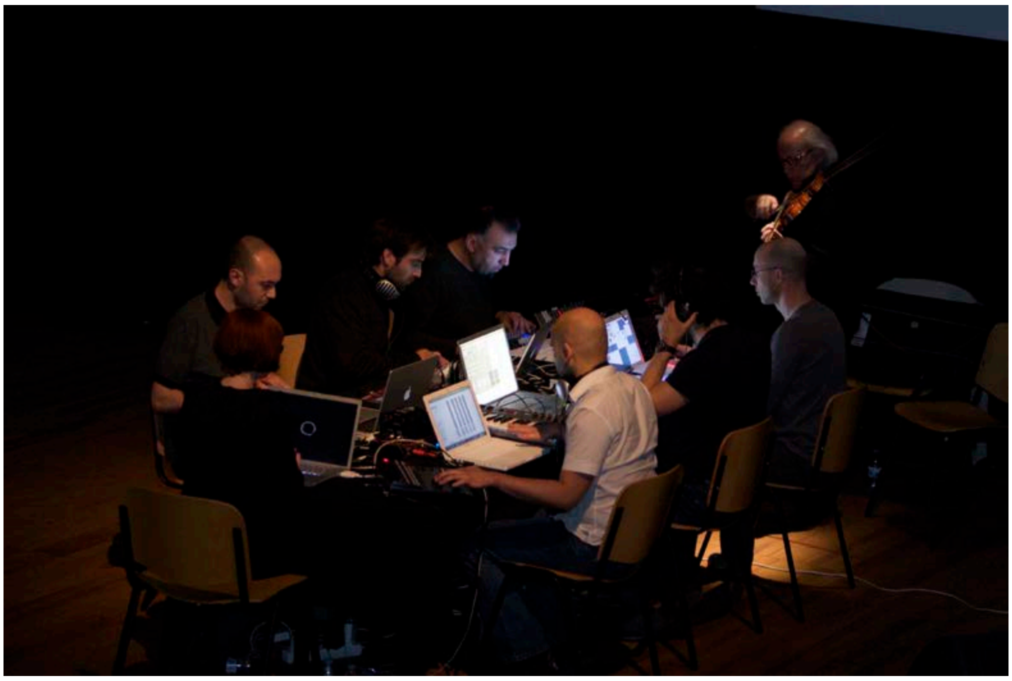


Fig. 16 Variable Laptop Orchestra and Carlos Zingaro

For this event, the formation of VLO consisted of six laptop performers processing Carlos Zingaro's violin sound. The signals from these laptops was then mixed by José Luis Ferreira. Also, visuals were created by Joana Gomes.

A key aspect for this performance consisted in the constant switching between predominant laptop performers who where continuously exploring distinct approaches for the generation of sonic material. The resulting form of the work evolved from the layering of autonomous blocks that progressively found protagonism and dispersed due to Ferreira

Lopes' work on the mixer. The conversational character between performers is reinforced by their disposition on stage (Fig. 16), being all capable of seeing each other.

Within our intervention, we continued to work on signal-processing techniques. We would create multiple hybrid voices or low-pitch drones by capturing and repeating the phrasing of the violin, for instance, changing its pitch a fifth or an octave above or below, or sustain certain notes. Another approach was to diffuse masses of sounds in order to create the perception of movement from the violin towards the audience, or creating action-response situations with the violin, triggering short energetic gestures of processed violin sounds.

5.5.2 Reinold Friedl Ensemble

Invited by Granular, and integrated in the *Metasonic* cycle, we had the opportunity to participate on a three-day creative residence taught by Reinhold Friedl that resumed with a performance as a member of *Reinhold Friedl Ensemble*[114], carried out at Goethe Institut.

This collaboration was significant for our research given Friedl's manifest interest on spectral, textured, timbre-based sounds[115] and the means by which he explores them through performance. Such interests can be perceived in his proposals as artistic director of *Zeitktratzer*, having collaborated with Carlsten Nicolai on *Zeitkratzer Electronics* (2008), or with Lou Reed on *Metal Machine Music* (2007). Also, he proposed *Xenakis Alive* (2007) which implies an aesthetic link to Iannis Xenakis' stochastics and electronic music. On *Inside Piano* (2011), for instance, a vast spectrum of sounds that give form to the musical work emerge as Frield explores extended techniques for the piano, implying a plasticity and richness that we are interesting in pursuing in our musical practice.

Throughout the residence, Friedl proposed a series of actions that deeply relate to improvisation pratices, such as imitation games, reinforcing the need to respond to certain

[114] The performance was carried out with Reinhold Friedl on "inside piano" and direction, Ulrich Mitzlaff and Miguel Mira on cello, Hernâni Faustino on DoubleBass, Pedro Lopes and Miguel Cardoso on electronics.

[115] We refer to timbre-based as oposed to note-based (cf. Ch. 2.2.2).

stimulus with very well defined gestures. For instance, he would define that a certain piano pattern should be rythmically repeated by the ensemble by only varying the timbre or dynamics, which could generate aesthetically relevant bursts or masses of sound. By this period, our research had already provided a diversity of methods for the creation of a rich palette of sonic materials, however, our ability to act fast and dynamically in performance was constrained by the need of sampling other musicians' sounds and by the lack of control of sonic events, given the limited set of interfaces we afforded.

A key aspect relating to the computer's instrumentality and autonomy was underlined, leading to the exploration of two complementary directions. At one end, we began to create sound through sound synthesis processes, and, at the other, we started to investigate possibilities for richer performance control, which led to the development of *SCPad!*, an iOS software that will be discussed later in this chapter (cf. Ch 5.6).

5.5.3 2+n

2+n is a duo we have created with Ricardo Guerreiro in order to explore common interests within interactive music practice. We were both concerned with the expressive potential of computers for music practice in a live context, as carried out by the late 1970s laptop bands such as *The Hub* or by the recent live-coding scene, as carried out by Julian Rohrhuber and Alberto de Campo on *PowerBooks Unplugged*. In these lines of action, we began this project by attempting to establish a musical discourse between performers and with the audience through the exclusive use of our laptops, using an interactive programming language to generate and control sonic processes. Each of us developed our software autonomously, but our laptops were connected in order to share the generated sound, its respective source code[116] and the software's internal active parameters. We also conceived of opening the duo to external collaboration (thus, the *n* variable in the project's name) and have already had the pleasure to perform with Gustavo Costa, Henrique Fernandes, António Jorge Gonçalves and Joana Fernandes Gomes.

The most significant change in relation to previous musical activities, lied in the fact that by constraining the project to the exclusive use of laptops, performing without pre-

[116] The fact that both members of 2+n use *SuperCollider* facilitates the sharing of computational resources.

recorded samples, we were enforced to describe and control all sounds algorithmically, resourcing to sound synthesis, which posed a challenge regarding the means to create aesthetically expressive and relevant sounds.

This concern has led our practice towards the exploration of microsound. The possibility to consider sound as an aggregation of masses of particles that can be algorithmicaly shaped offers a great malleability, a great plasticity for the generation of sounds that can range from rythmic patterns to dense continuous tones or textures with rich timbre. In previous chapters we have already regarded a diversity of programs that follow this approach by operating in the domain of the sample, namely *Sawdust* by Herbert Brün, *SSP* by Goedfried Koenig or *ST* by Iannis Xenakis (cf. Ch2.2.2). Also, research on microsound has been carried out by Curtis Roads (2004), Alberto de Campo (2011), Carlos Caires (2006), Agostino Di Scipio, Horaccio Vaggione (Solomos, 2007) among others. Musical works such as *Volt Air (2001-2003)* by Roads, or *24 Variations* (2001) by Vaggione illustrate the aesthetic possibilities of such direction.

Our initial effort consisted in generating a variety of waveforms and envelopes that would serve as our compositional material. Rather than using pre-recorded samples for the creation of grains, the source material consisted of a diversity of synthesis algorithms. We employed common techniques such as subtractive, additive or cross synthesis, frequency and amplitude modulation or physical modelling. Also, we also wrote sound functions to prescribe grain envelopes such as *square*, *pulse* or *sinc* (cf. Ch. 5.6.4 *GrainEnv*).

A diversity of material was created by algorithmically manipulating grains, mapping their source waveforms to synthesis algorithms and adjusting their shape, duration, envelope, filtering and spacialization[117]. Adopting an interactive programming approach (cf. Ch.4.3) and taking advantage of the *Odef* object we have implemented (cf. Ch.5.6.4), we created instruments and scores, employed computational procedures such as iterations, recursions, and biased randomness to generate phrasings. The microsound processes described by Roads, such as glisson, grainlet, trainlet or pulsar synthesis provided an enriching starting point. The resulting material, such as little crackle, creaks and fry sounds, or short glissando sequences, can be perceived in the first minutes of *n=0* (cf. Appendix 2).

[117] Although the provided recordings are in stereo, all our work is developed using *ambisonics*, taking advantage of *BFormat UGens* for *SuperCollider* developed by Josh Parmenter.

As soon as we established a formal algorithmic control of our compositional material, we started to explore means to dynamically interact with it in performance. Vaggione argues that algorithmic approaches to microsound pose a compositional problem on “how to generate true singular events, and how to articulate them in the larger sets without losing the sense (and control) of these singulatiries” (Roads, 2005). For Vaggione, direct intervention on sonic material is fundamental, making it incompatible with algorithmic approaches. In his compositional practice he rather employs the term *micromontage,* being described as that of a pointillistic painter. The development of *SCPad!* (cf. Ch.5.6.4) has enabled us to address this issue by enabling the simultaneous manipulation of a vast amount of parameters from various objects. Although we don’t control each grain individually, our approach is a compromise between Vaggione’s pointillistic approach and overall algorithmic control. We can control a complex grain cloud, defining tendency values for grain’s density, amplitude, frequency, waveforms and envelopes, among others. We can simultaneously trigger diverse routines that can generate morphologies, singularities, that are perceived in differentiated ways. In fact, Reynold Friedl was a fundamental reference for the development of *SCPad!*. Such interface allows an optimal control and fast switching of sonic material, enabling the creation of multiple layers of structures of sound at different levels.

As we became autonomous in the generation of sound, we turned to the processing techniques that had developed for VLO and began to iteratively process our own musical material, being able to create unique feedback loops. Such actions can be heard at 13:00 minutes and onward of *n=0* (cf. Appendix 2).

We also began to develop strategies that allowed the creation of situations in which the material could be autonomously generated but over which, if desired, one could claim back its immediate control. Adopting Xenakis’ understanding of sound parameters as sets of vectors in a multidimensional space we developed graphical representations of particle systems, simulating behaviour by applying physical forces such as attractors and repulsors, that we then mapped to sonic parameters. We have also explored the usage of networks for the creation of non-linear situations by mapping to networks’ nodes a series of parameters. Details of such work will be presented in this chapter (cf. Ch 5.6.4).

As implied by the exposed material, our musical practice evolves from a reductionist view of sound as the perception of air waves that reach our ears. Emiting those waves we find voltage variations of an electronic device that operates in analogy with numeric streams emmited from a computer. In this view, our practice develops from the mathemacial definition of time-value functions that are aggregated, intertwinned, recorded, added, divided or multiplied until converted into sound. They are assigned a vector and subjected to forces that imprint behaviour, becoming part of a complex stream of other sounds utterly controlled by us.

Having exposed some of the ideas and principles that we have developed in our musical activity we expect to have provided some lines that can facilitate the comprehension of the concepts that were implemented in the *Thr44 framework*.

5.6 The Thr44 Framework

5.6.1 Overview

Thr44 is an experimental framework that abides to the criteria and objectives described above in order to contribute to the development of methodological and operative strategies for interactive music practice. The difficulties in fulfilling these criteria are not so much related to the generation of musical material itself, but rather to the means to articulate, organize, integrate, relate and expand musical ideas within a dynamic context such as a musical performance, attempting to improve its improvisational and expressive potential.

Such possibilities encompass a set of concepts and principles that attempt to integrate sound design, human-computer interaction, algorithmic composition and performance into a single activity, defined as the composition of an interactive music system.

We consider that by following an interactive programming paradigm, and by incorporating existing models such as prototype-based programming and its delegation mechanisms, we can take advantage of their potential for dynamically structuring and organizing modular components and behaviours. Such components can then be combined and used, increasing our decision-making in a musical performance.

Rather that a closed and static program that is conceived in order to provide every conceivable solution to interactive composition, *Thr44* is a framework that offers a package of computational objects and enables the continuous work on its components at various levels, not only facilitating the prototyping of musical ideas, but also their continuous incorporation into a growing environment, as a cumulative open work.

5.6.2 Considerations on implementation technologies

The real-time implementation of the classes presented in the *Thr44* package was developed on *SuperCollider*[118] (*SC*), an interactive programming language for music composition (McCartney, 2002). There are many other language environments – namely *Chuck*[119] (Wang, 2008), *Common Music*[120], *Nyquist* (Dannenberg, 1993) or *Kyma*[121] – *but SC* is free, open-source, and supported by a vast research community. *SC* provides many necessary audio and scheduling primitives and allows diverse programming approaches (functional, imperative, object-oriented, prototype-based, among other). It also implements many high-level abstractions (dynamically typed, single-inheritance, event dispatch, garbage-collected). Since *SC* source is open, it can easily be changed or extended. *Server UGens*, for instance, can be developed in *C* (Kernighan & Ritchie, 1988). In this sense, a relevant project is *Faust*[122], a functional programming language developed by Yann Orlarey that was specifically designed for real-time signal processing and synthesis and that allows the development of *UGens* for *SC*.

[118] http://supercollider.sourceforge.net

[119] See http://chuck.cs.princeton.edu

[120] See http://commonmusic.sourceforge.net

[121] See http://www.symbolicsound.com

[122] http://faust.grame.fr

SCPad! was developed on *C++* (Stroustrup, 2008), using the *iOS*[123] *SDK*[124] and *openFrameworks*[125], an open-source toolkit for creative coding. It contains a wide variety of libraries in a single package that can be used for *Windows*, *Macintosh*, *Linux*, *Android* and *iOS* operating systems. It facilitates graphical representation and provides communication support for a variety of devices, such as webcams and physical computing (for example, Arduino[126]). It also supports many communication protocols (*Open Sound Control*, *sockets* and other web services) and data model formats (*JSON*, *XML*).

Although some technical details of our work are related to such technologies, we consider that the overall principles and approaches taken in our study remain valid for other technologies. In fact, and as opportunely referred, many of our implementation options were influenced from approaches taken from other programming environments and contexts.

5.6.3 System architecture

The *Thr44 framework* is integrated in the existing architecture of *SuperCollider* (*SC*).

SC consists of two separate applications, of which the *Server* is a clean and efficient real-time synthesis engine, and the *Client* provides a language interpreter, where algorithmic composition is developed. These two applications communicate through network messages using *Open Sound Control (OSC) (Wright, 2005).* Multiple instances of *Servers* and *Clients* can be used simultaneously.

Within the SC *Client*, one can either develop classes, that are required to be compiled, and thus, to restart the interpreter, or develop scripts that can manually be interpreted at runtime.

[123] It is of significance to note the deep technological changes that are currently happening, in particular on what it concerns *tablets* with *multi-touch* technologies. In 2010, *Apple* released *iPad* targeting the need for a device to read books in digital formats. Surely, it opens new possibilities to access *multi-touch* technology for creative musical expression, but this device restricts the development and distribution of software to its developer platforms and languages (app Store). The new products launched in recent years open the scope of technological choices. As an example, we can refer *Android OS*, used in a variety of *tablets*, or *Microsoft Surface*.

[124] https://developer.apple.com/ios8/

[125] http://www.openframeworks.cc

[126] http://www.arduino.cc

The real-time implementation of the Thr44 framework consists essentially of *SuperCollider* classes. In the next section we present the way in which they are organised.

The SCPad! application is built using the *model-view-controller* (MVC) paradigm. It consists of a general controller that sends and receives data from *SC* and is responsible for creating and destroying the distinct required graphical user interfaces. It communicates with *SC* through *OSC,* using JSON and XML formatted strings.

5.6.4 The Thr44 Implementation

In this section we discuss some of the implementation details of *Thr44*, attempting to provide an understanding of the strategies and approaches that have been carried out. The diverse possibilities that this work permits cannot be fully addressed, however, we attempt to provide the key features that can contribute to the improvement of interactive music practice.

Its source code is accessible at:

- https://github.com/Thr44

A full description of its objects is available at:

- http://www.344server.org/phD/docs/Help/Thr44.html

Also, a version of these resources has been forked and is available at the accompanying media storage device (cf. Appendix 2).

Main Structure

The Thr44 framework is organized as follows:

Core – Containing the major objects that are fundamental for the usage of the framework, such as the *Odef* object.

Components – Consisting of structural objects developed in order to enhance our compositional practice. Among these components we find *interaction* related components, such as the *SCPad*, mathematical abstractions like the Network ***dataModels*** classes, and other *utilities* such as the Buffer related classes.

Interaction: *SCPad*

dataModels: *Thr44Network*, *PetriNet*, *Thr44ParticleSystem*

utilities: *GrainEnv*, *BufferList*, *BufferUtils*, *ContinuousBuffer*, *Recorder*

Other – Additional classes and scripts that result from performing with the system, such as instrument descriptions, configurations, among others.

Odef

One of the main points of this project is the development of strategies for increasing versatility, transparency and flexibility in the description and articulation of distinct components of the system. This concern has led us to the study of *object-oriented programming*; in particular, to the recognition that the *prototype-based* paradigm, often associated to interpreted languages, could provide us with a viable solution for improving the structuring and organisation of musical material.

Concerning performance, in which the decision-making timeframe is of critical importance, one requires strategies to facilitate the relationship with the system, exposing its state and possibilities for action, constituted by a diversity of heterogeneous objects, their functions and variables. Performing with programming languages entails a series of operational difficulties, such as the need to create a mental map of the constituents and of the state of the system. Additionally, a common issue in musical performance lies in the loss of a reference to an active sonic entity, being unable to control or stop it, and forcing one to restart the program. Similarly frequent, but less critical, is the need to read-through our code in search of an entity's name, or of a forgotten value.

```
a={SinOsc.ar (220, 0.2)}.play;
a=nil;
```

Fig. 17 By executing these two lines of code in *SuperCollider*, the *SinOsc* would no longer be accessible to control, except by directly querying the server, or restarting the environment.

In order to address these issues, we have developed *Odef*, an entity that works as a wrapper to a multitude of distinct entities.

```
Odef (\name, {
    arguments
    function body
});
```

A first example, implementing FM synthesis:

```
Odef (\FM, {
     arg carrier=400, mod=20, modDepth=100;
     Out.ar (0, SinOsc.ar (carrier + SinOsc.ar (mod, 0, modDepth)));
});

Odef (\FM).play;

Odef (\FM).set (\carrier, 2, \mod, 1000, \modDepth, 1000);
```

As such,

(i) We can declare any type of process, ranging from a mathematical operation to a sonic process:

```
  Odef (\sinDown, {
          arg gain=1;
          25.do{
              arg i;
              {
                  var sound, env;
                      env=EnvGen.kr (Env.perc (0.01, (4096/44100)/ (28.1-i)),
doneAction:2);
                  sound=SinOsc.ar (9000- (110* (i+20)), 0, 1)*env;
                  Out.ar (0, sound);
              }.play;
          0.05.wait;
     }); // A Task iterating 25 times;

     Odef (\addition, {
          arg n0=1, n1=2;
          n0+n1;
     }); // A mathematical operation as example of a generic function;

     Odef (\oscil, {
          arg freq=440;
          Out.ar (0, SinOsc.ar (freq));
  }); // A sinusoudal oscillator;
```

(ii) The *Odef* entity functions as an abstraction that allows differentiated procedures to share common methods: *play*, *stop*, *set*, *get*. The *set* and *get* methods allow the changing and accessing of the parameters that are declared as arguments:

```
Odef (\oscil).play;  //start oscillator;
Odef (\oscil).get (\freq);           //return 440;
Odef (\oscil).set (\freq, 880); //change its frequency;
Odef (\oscil).stop;  //stop the oscillator;

Odef (\addition).play; //returns 11
```

(iii) The function body is not closured so that we can access any variable from the global scope of the environment:

```
//variables from global scope:
q= ();
q.a=3;
q.b=1;
~staticValue=4;

//creating an addition function, where its internal variables
read the global scope:
Odef (\ addition, {
    ~staticValue+q.a+q.b;
});

Odef (\addition).play; //returns 8

q.a=10;
Odef (\addition).play; //returns 15
```

(iv) All Odef instances that were declared can be accessed. In performance, for instance, one can easily find the general constituents of the system that are being used at the time.

```
Odef.all;
//returns an Object with all the Odefs  e.g. ( 'oscil': an Odef )
```

(v) We can introspect each Odef:

```
Odef (\oscil).asCompileString  //returns
                                   Odef (\oscil, {
                                      arg freq=440;
                                      Out.ar (0, SinOsc.ar (freq));
                                   });
Odef (\oscil).key  //returns oscil;
Odef (\oscil).argNames  //returns [ freq ];
Odef (\oscil).argValues // returns [440];
Odef (\oscil).objectType //returns "soundFunction";
```

This approach is not only useful for recovering the contents of the object, but also to be accessed by other computational entities that can question the *unknown* object and behave appropriately. An example object that takes advantage of such method is SCPad, which will later be described.

(vi) Odef also implements *SuperCollider* event system, providing a means to listen to specific actions, such as for instance, the moment when a sound process finishes playing.

```
odef.addDependant ({
    arg odef, status;
    if (status=='stopped', {

         … do something

    });
});
```

(Vii) In order to generate multiple copies of the Odef, we have implemented the following methods:

clone: This method creates a new independent *Odef*. Its internal contents can be modified without affecting the original entity.

```
Odef (\oscil).clone (\clonedObject)
```

getInstance: This method creates a new instance that is mapped to the original one. If the parent *Odef* is changed, its child instances will inherit its properties. This method is useful for the creation and manipulation of multiple instances of a single *Odef*.

```
        Odef (\oscil).getInstance (\instance0)
```

The development of *Odef* evolved from Alberto de Campo's insight, providing knowledge for understanding the possibilities of interactive programming and, particularly, the implementation details of *JITLib* (Rohrhuber & Campo, 2011).

A full and detailed description of *Odef* can be found in the accompanying media device.

SCPad

An environment like *SuperCollider* provides many ways to work with external data, enabling communication between a diversity of interfaces such as *Human Interface Devices* (HID) or *graphical user interfaces* (GUI) and protocols, namely *Musical Instrument Digital Interfaces* (MIDI), *Open Sound Control* (OSC), among others.[127] However, the question is not so much of a technical nature (on the means to send and receive data), but rather oriented towards the means to relate the interfaces with the sound generating software with a minimum effort and maximum expressiveness.

A general approach to such interfaces encompasses a separation between the programmed sound engine and the interface itself. The interface is static and unidirectional, solely capable of sending its current state; it requires the performer to provide information on the appropriate mappings that can be established between these entities in order to control sound. Recent applications for mobile devices, such as Robert Fischer's TouchOSC (2009) (Fig. 18) can be framed within this view.

[127] (Kersten, Baalman, & Bovermann, 2011) provides a more detailed description of the communication protocols that are implemented in SuperCollider.

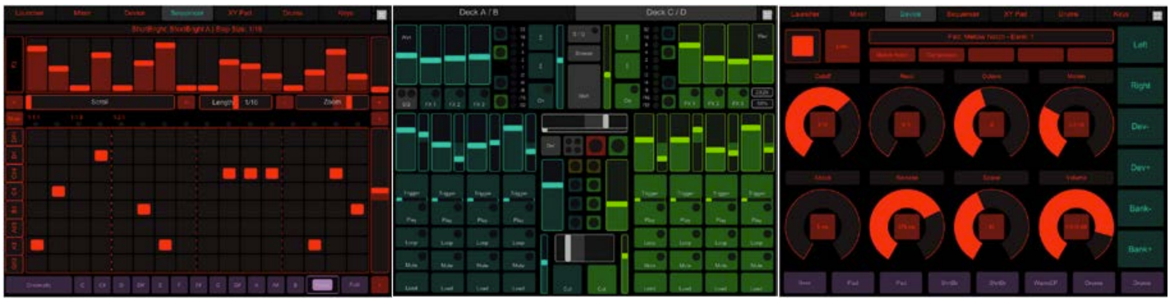


Fig. 18 TouchOSC by Robert Fischer (2009)

As previously discussed (cf. Ch. 3.4), another approach is evidenced by the interest in the creation of expressive intelligent instruments, or composed instruments (Magnusson, 2009; Schnell & Battier, 2002). In this view, the interface already provides compositional decisions, namely arpeggios, rhythms, scales, among others, blurring the distinction between system design, composition and performance, as Magnusson points out (2009).

The potential of ubiquitous computing is that virtually every device can become a compositional tool, allowing the appropriation of technology that was not originally intended for a certain purpose. As an example, there is a growing interest in the development of musical interfaces for mobile devices due to their *multi-touch* technology, their processing power and communication capabilities. We can take advantage of such devices in order to develop dynamic interfaces that are capable of adapting themselves to external changes *on-the-fly* and generate appropriate controls. A flat screen does not provide the proprioceptic response one would desire, however such direction allows us to have a dynamic interface in which one can control a vast amount of parameters simultaneously.

SCPad! is a software package that dynamically generates graphical user interfaces (GUI) on an *iPad*, conforming to its *multi-touch* technology. It consists of a *SuperCollider* class that enables seamless communication between our framework and the *iOS* program. Rather than statically mapping controls to sonic processes the have previously been defined, as one does with *TouchOsc*, this interface dynamically reflects the objects and properties that one instantiates on *SuperCollider*.

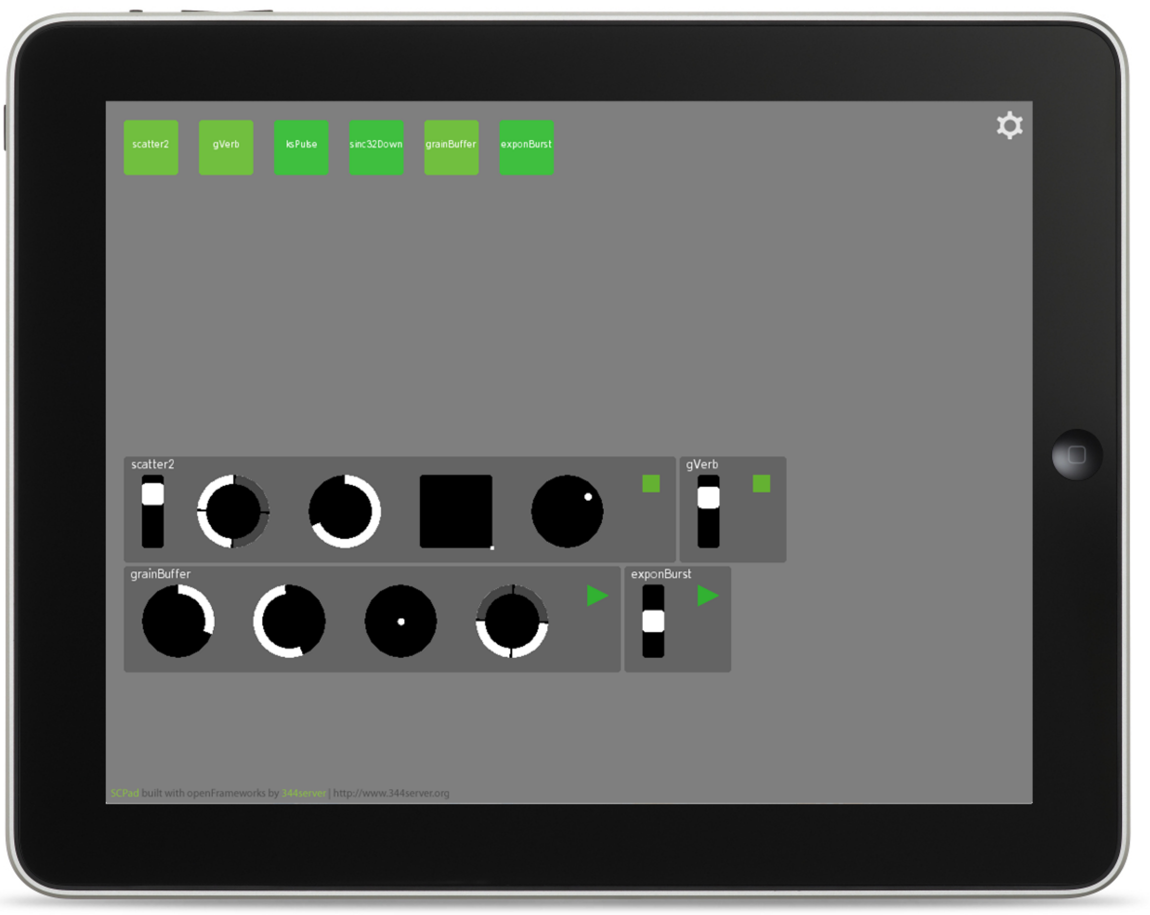


Fig. 19 SCPad! (2013)

As previously described, the *Odef* is an object that enclosures compositional internal logics and functions as a generic *interface*[128] for common methods, such as *play*, *stop*, or *set*. *SCPad* is built on the *Odef* object in order to seamlessly listen to its available instances, and respective arguments, in order to dynamically generate the appropriate graphic control.

128 The term interface refers to as a programming language entity that serves as an abstract specification of methods to be implemented.

An example of Frequency Modulation Odef, with carrier, modulator and modulator depth as parameters, and its respective GUI (Fig. 20):

```
~scPad=SCPad ("192.168.1.92");

Odef (\FM, {
    arg carrier=400, mod=20, modDepth=100;
    Out.ar (0, SinOsc.ar (carrier + SinOsc.ar (mod, 0, modDepth)));
});


Odef (\FM).clear; //removes the Odef
```

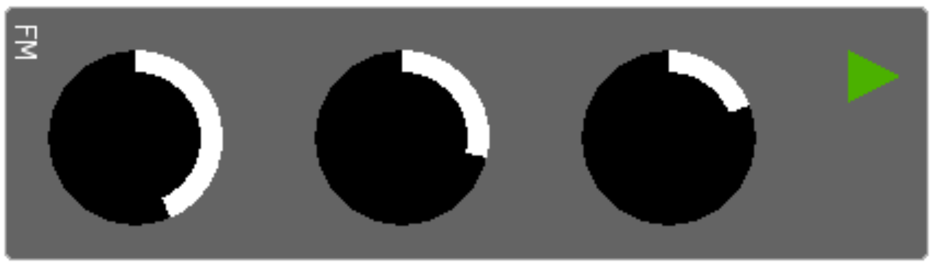


Fig. 20 An example of the interface that would be generated (carrier, modulator and modulator depth)

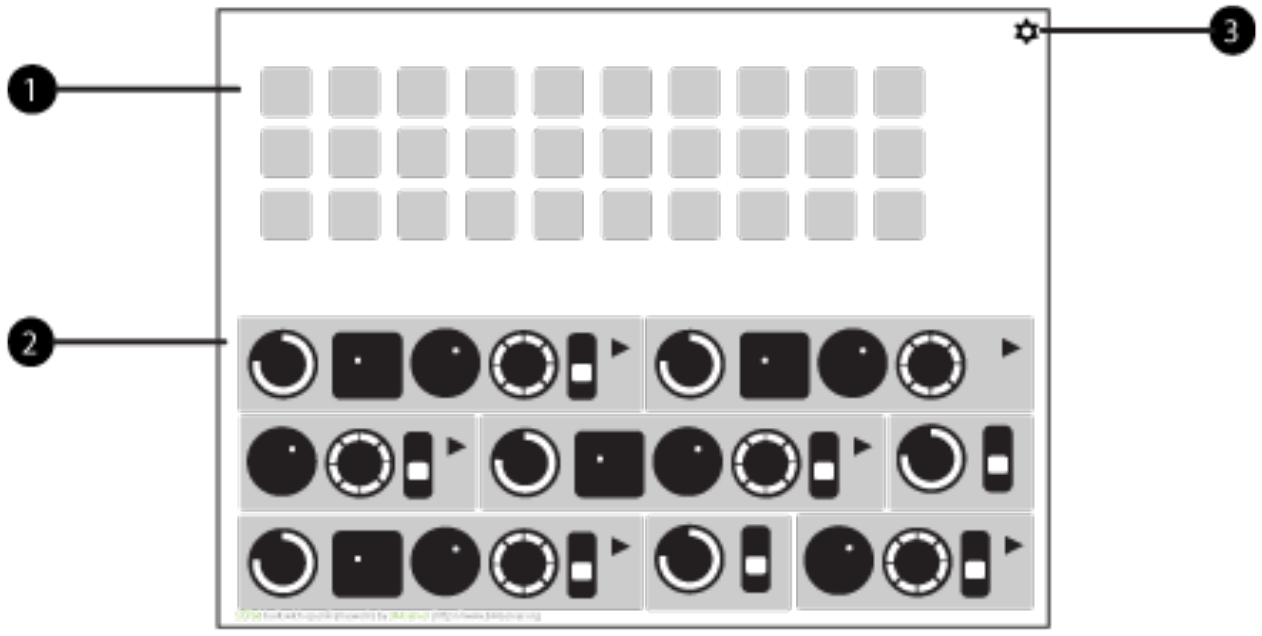


Fig. 21 Diagram of SCPad! Interface organisation

The *SCPad!* interface is divided into two major sections:

The *Odefs* section *(*1) contains a set of buttons that represent *Odef* instances. These can either be defined as *triggers*, that run process when tapped, or as *selectors*, which toggle the appropriate parameters representation. In order to access the graphic control of the parameters of *trigger Odefs* one can press-and-hold the buttons.

The *parameter group* section (2) displays the group of graphic controls that are mapped to the Odef parameters.

In order to establish the connection to SuperCollider, it also contains a settings panel (3).

While *SCPad* has already some predefined controller configurations, one can always override that information and assign specific controllers to *Odef*. In Fig.22, we illustrate a set of controllers we have already implemented.

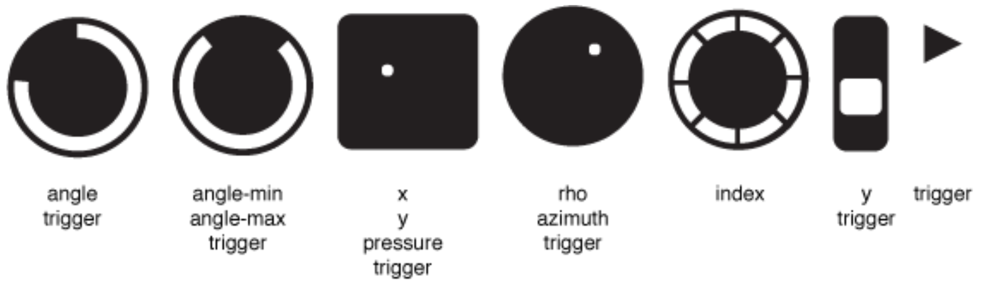


Fig. 22 Illustration of the implemented controls: *circularSlider, circularRange, ScatterXY, ScatterRadial, circularSelectSlider, slider, play*

Continuing with the example of FM synthesis Odef, we present how GUI contros could be manually defined:

```
//Example of an Odef to use with SCPad:
 (
Odef (\FM, {
    arg carrier=400, mod=20, modDepth=100;
    Out.ar (0, SinOsc.ar (carrier + SinOsc.ar (mod, 0, modDepth)));
});

//definition of controls:
Odef (\FM).addControl ("circularSlider", [\freq]);
Odef (\FM).addControl ("ScatterXY", [\mod ,\modDepth, \play]);
Odef (\FM).addControl ("Play", [\play]);
)

//Definition of control method:
Odef (\FM).setControlMethod (\select);
```

In order to specify the range and curve of the controller that is expected to be mapped onto the Odef's parameters, a *ControlSpec*[129] can be used, enabling a diversity of numeric translations (linear, exponential, sine, among others).

```
ControlSpec.specs[\azimuth] = ControlSpec (-pi, pi, \pan, 0.001, 1, units: "");

ControlSpec.specs[\rho] = ControlSpec (0, 8, \linear, 0.001, 1, units: "");
```

Although parameters can generally be defined as numeric values, *Lists* of objects, such as *Buffers*, can also be used. In this example, we specify a set of grains[130] to be used in an *Odef* and a circular selector as its controller. The -> has been adopted in order to be able

129 *ControlSpec* is an input specification object, native to the *SuperCollider* Language.

130 The GrainEnv object is an utility object that we will later describe, consisting of a library of sound functions.

to specify a default value within a list of possible values that need to be represented in the GUI control.

```
      Odef(\scatter, {
          arg envbuf=[GrainEnv.sinc16Env]->
[GrainEnv.sinc8Env, GrainEnv.sinc16Env, GrainEnv.sinc32Env, GrainEnv.exponEnv];
      ...
      });
      Odef(\scatter).addControl ("circularSelectSlider", [\envbuf]);
```

SCPad! source code can be retrieved online[131] or at the accompanying media device (cf. Appendix 2). A compiled version has been submitted to Apple's *App Store*[132].

Networks

This research is focused on the development of strategies that can increase the compositional decision-making in contexts such as performance, where one finds limitations such as the cognitive complexity of algorithms or the available time for typing. To this end, we have implemented a series of abstract mathematical classes for networks and physics that enable the creation of complex situations such as the below documented PetriNet and Boids.

Networks, or *graphs*, are mathematical structures that are used to model relations between Objects. Networks are constituted by collections of nodes and by collections of edges (or relations) that connect pairs of nodes. The relevance of such structures consist in the possibility to model complex topologies, allowing the exploration of non-linear generative processes such as Markov chains or stochastics. *Thr44Network.sc* was created to model graphs as a means to explore complex and non-linear situations for musical performance and composition.

[131] https://github.com/Thr44/SCPad/

[132] https://itunes.apple.com/tr/app/scpad!/

The following example illustrates how to create a network of 75 nodes, randomly interconnected, and its resulting graphical representation (Fig. 23). From this abstract structure, one can map a series of sound processes or parameters, and navigate through its nodes, opening the possibility for generating non-linear sonic narratives.

```
~network=Thr44Network.new ();
~node=Thr44Node.new (~network.getNewNodeID ());
~network.addNode (~node);
75.do ({
    |i|
    ~node=Thr44Node.new (~network.getNewNodeID ());
    ~randomNode=~network.nodeList[ (~network.nodeList.size-1).rand];
    ~relation=Thr44Relation(~node,~randomNode,~network.getNewRelationID());
    ~network.addNode (~node);
    ~network.addRelation (~relation);
    });
~network.log ();
~network.gui;
```

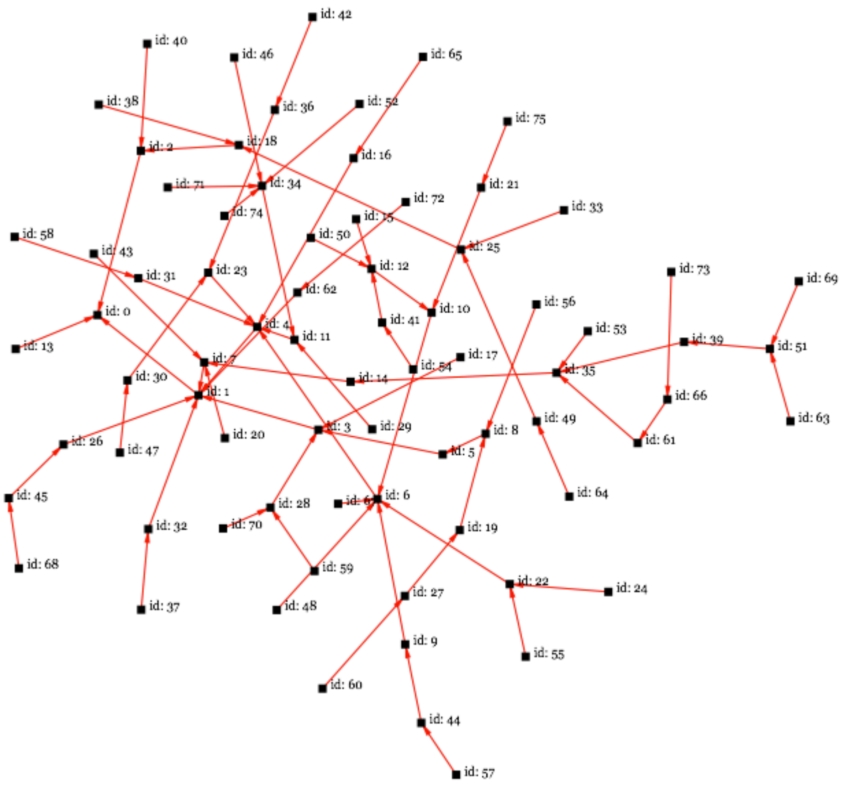


Fig. 23 Graphical representation of a *Thr44Network*

PetriNet

The implementation of PetriNets (PN) models (Petri & Reisig, 2008) in our work evolved from a collaboration with Ricardo Guerreiro and António Rito Silva. Guerreiro's interest in network models for musical performance and composition[133] led to the conceptualization of an application of PN for live computer mediated electroacoustic music. We have implemented their concept in *Supercollider*, which allowed the further development of the network structures presented above.

This work was presented in SuperCollider's Symposium 2012 and its creative potential was explored in *2+n* (Cardoso, M., Guerreiro, R.).

A PN is a graph that consists of two types of nodes (transitions and places), that are connected by directed relations. It functions by adding tokens to places, and when all the inputs of the transition have enough tokens to satisfy arcs weight, the transition is triggered. The input tokens are consumed and tokens are produced in the output places (Fig. 24).

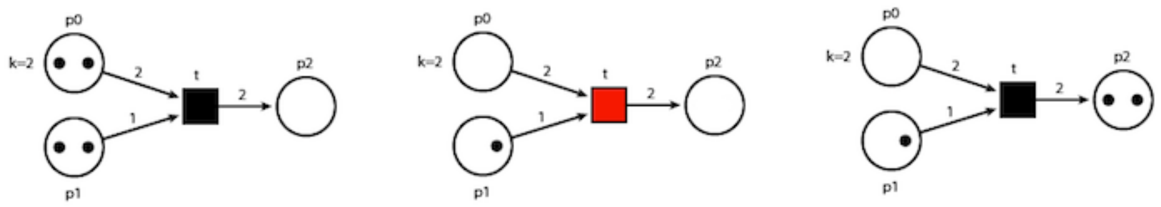


Fig. 24 PN consumption and production of tokens.

When a PetriNet class is instantiated a new PN graph is dynamically created using a random procedure. This procedure ensures the creation of a correct PN graph comprising a certain number of places and transitions, bounded by a defined minimum and maximum, and connected with a given connectivity index.

[133] See Guerreiro discussion (Guerreiro, 2015:121).

```
	//create petrinet:

	p=PetriNet (numPlaces:6, numTransitions: 8, connectivityIndex: 0.2);
	//present graphical interface:
	p.gui;
```

Each transition of the PN is associated to an external object that, by default, is randomly selected from a *List*. The objects of this assigned *List* share a common interface (*Odefs*).

```
	// define some sound processes:
	 (
	Odef (\resonz, {
	    var sound, env, trig;
	    env=EnvGen.kr (Env.sine (12,  0.2), doneAction:2);
	     sound=Resonz.ar (Pulse.ar (Rand (30,90), 0.5), rrand (100, 10000), rrand
(0.1, 1))*env;
	});

	Odef (\sines, {
	{
	    SinOsc.ar (rrand (200,3200))}.dup * EnvGen.kr (Env.sine (12,  0.2),
	doneAction:2);
	});

	Odef (\saws, {
	{
	    Saw.ar (rrand (1600,3200))}.dup * EnvGen.kr (Env.new ([0.0001, 0.01, 0.3,
rrand  (0.001,0.01),  0.0001],  [rrand  (6,12),  rrand  (0.1,1),  0.001,  rrand
(6,12)],'exponential'), doneAction:2);
	});

	b = Buffer.read (s, Platform.resourceDir +"/sounds/a11wlk01.wav", bufnum:1);

	Odef (\grainy,
	{
	    var bufnum = 1, trate, trigger, dur, env, out;
	    trate = LFNoise0.kr (LFNoise1.kr (0.3).range (0.1,1)).range (0.1,21);
	        dur = 0.1 / trate;
	        trigger = Impulse.kr (trate);
	        env = EnvGen.kr (Env.linen (3, Rand (6, 18),  3, 1.0), doneAction:2);
	         out = TGrains.ar (2, trigger, bufnum, Rand (1.0,3.0), LFDNoise1.kr
(0.1).range (0,BufDur.kr (bufnum)), dur, TRand.kr (-0.99,0.99,trigger), 0.5, 2);
	        Out.ar (0, out*env);
	});
	)

	// make these objects available to our PN
	~list=[Odef (\resonz), Odef (\sines), Odef (\saws), Odef (\grainy)];
	p.addOdefs (~list);

	//starting network flow:
	p.play();
```

When a transition is fired it launches its related object that is set for a period of time. When the timeout occurs, the process is deallocated, and the PN transition finishes its execution creating tokens in its outbound places. To allow this behavior we have extended the classical PN model with a time factor that seems to be fundamental, for our musical purposes, in dealing with the potential coexistence of multiple sound strata that our parameter controls.

```
//set a waitTime of silence after each sound process finishes:

p.waitTime({rrand(2.4,7.5)});
Also, one is able to interact with the PN, by adding or removing tokens.
p.getPlaceFromId(6).addTokens(1);
p.getPlaceFromId(8).removeTokens(1);
```

Furthermore, our PN version randomly selects its initial state, i.e. the place from where it starts, and may not have a final state, thus allowing a non-linear cyclic network flow.

A video of the presented example is provided (cf. Appendix 2).

PetriNet BasicBuilder

In order to facilitate the construction of PN, we have also implemented a BasicBuilder with methods such as insertSequence, insertLoop, fork, join, and closePath. The following example illustrates the creation of a PN using the basic builder.

```
(
    var petri, result, builder, endPlace, numPlaces;
    petri=PetriNet ();
    builder=PetriNetBasicBuilder (petri);
    numPlaces=3;

    result=builder.buildSequence (numPlaces);
    endPlace=result.endPlace;
    petri.initPlace=result.initialPlace;
    result=builder.insertSequence (numPlaces, petri.initPlace);
    petri.initPlace=result.initialPlace;
    builder.addLoop (numPlaces, endPlace);

    petri.log ();
    petri.gui ();
)
```

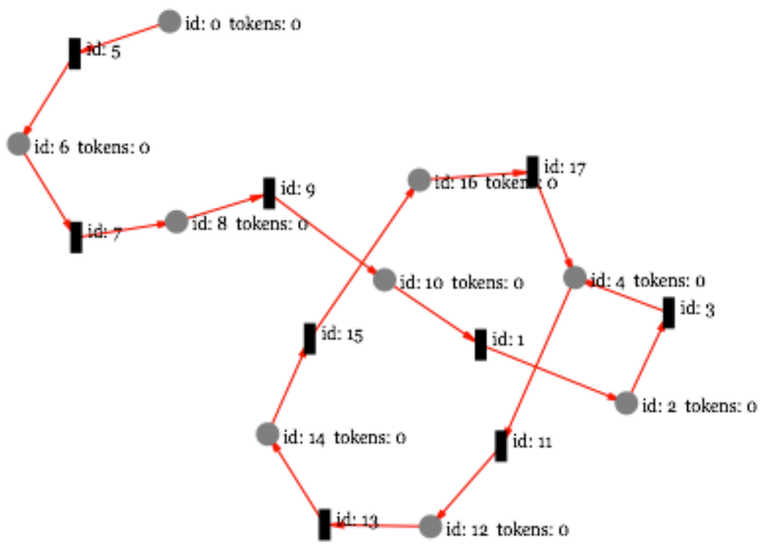


Fig. 25 Resulting representation of a PN

As the network flows rely on the consumption and production of tokens, deadlock and starvation situations can be detected, making PNs critical for the analysis of concurrent entities' behaviours. However, from our musical perspective, such dependency led to difficulties in finding adequate models, making it unsuitable for our purposes. Nonetheless, this research allowed the implementation of enhancements to the *Network* classes, namely the methods for modelling these structures or the strategies to map their *nodes* and *relations* to sonic entities.

Particle System

The implementation of networks as an abstraction for modelling a diversity of structures suggests the creation of ecosystem behaviours such as collisions, springs, explosions, flocking birds, among others, that could be mapped to sonic entities in order to control their parameters autonomously. As such, we have implemented Thr44ParticleSystem, a class that manages particles and allows setting forces to these entities. In the example below, we use the network created in the previous section as a topology to create particles and forces among them.

```
~pS=Thr44ParticleSystem.new ();
~network.relationList.do{
    arg relation;
    var dist=50;
    ~pS.addSpring (relation.getNode0 (), relation.getNode1 (), 0.6, dist);
};

~pS.fillMatrixWithRepulsors (0.4, 100);
~pS.calculateParticles ();
~pS.gui
```

In order to create new forces, one can extend the *Thr44PForce* and implement the appropriate calculations.

```
Thr44PForceName : Thr44PForce{

    *new{
         arg …arguments;
         ^super.new (node0, …arguments).init (node0, …arguments);
    }
    init{
         arg node0, …arguments;
         this.node0=node0;
         …Assign variable values from arguments
         ^this;
    }
    calculate{
         perform required calculations on nodes position and
acceletation: vx, vy, vz, ax, ay, az

    }
}
```

These forces can then be assigned to Thr44ParticleSystem:

```
    ~pS.addForce (node0, node1, …arguments);
```

One can map these nodes to other objects by injecting a function to the nodes, that will be executed by calculateParticles ():

```
~network.nodeList.do{
    arg node, i;
    node.applyAction ({
         arg element;
         element.postln; //prints out a Thr44Node
    });
};
~pS.calculateParticles ();
```

Boids

In the following example we present an experiment for sound spatialization using the *boids* algorithm.[134] As described by Craig Reynolds, *boids* is a model for coordinated movement among particles that simulates the behaviour of flocks of birds, or schools of fish. Such model is often cited as an example of principles of artificial life and emergence. This example serves to illustrate how this work facilitates the creation of generative behaviours in order to control some aspects of sound; in this case we apply the boids implementation to ambisonics saw generator instances:

```
      Odef (\saw, {
          arg frq=100, rho=0.3, azimuth=pi, elevation=0;
          #w,x,y,z = BFEncode1.ar (Saw.ar (frq, mul:LFDNoise3.ar (0.3)*LFDNoise3.ar
(0.3)), azimuth, elevation, rho, 0.1);
          BFDecode1.ar (w, x, y, z, [-0.5pi, 0.5pi]);
      });
      b=Thr44Boids.new;
      b.applyAction={
          |boid, odef|
          var rho, azimuth, pos;
          pos=RealVector[boid.pos[0]-200, boid.pos[1]-200, 0];
          rho=pos.norm/200;
          if (rho.isNaN, {
               rho=0;
               pos=RealVector[-1000, -1000, 0];
          });
          azimuth=pos.angle (RealVector[0, 1, 0]);
          if (pos[0]*pos[1]<1, {
               azimuth=azimuth+ (2*pi);
          });
          odef.set (\rho, rho, \azimuth, azimuth); //we apply position to the Odef
      };
```

[134] See http://www.red3d.com/cwr/boids/

```
20.do ({
    |i|
    var odef;
    odef=Odef (\saw).clone ("saw"++i); //we clone the original Odef 20 times
    odef.set (\frq, rand (100)+20);
    b.addOdef (odef);
    odef.play;
});

b.gui;
```

Sound Utilities

Early stages of our work were essentially oriented towards improvisational strategies, which led us to recognize the need for the creation of objects for real-time audio sampling and manipulation. These would allow a more expressive dialogue with other musicians and their sonic material on significant moments of their performance.

Instead of operating with incoming sound as a continuous stream as if it were analog audio flowing through connected boxes that apply direct transformations and output the signal, one can rather treat sound as a database of discrete segments of particular morphologies.

BufferList

BufferList facilitates the creation and organization of collections of buffers by applying filters, keywords or other procedures, making them easily accessible by any other process of the system.

```
    BufferList.new(); //singleton, always returns the same BufferList
    BufferList.new().gui; //creates a GUI of the list

    BufferList.new().addBuffer (name, buffer, [keywords]);

    BufferList.new().getBufferAt (index);

    BufferList.new().getBuffersWithKeyword (keyword);
```

```
      b.addBuffer ("ok", Buffer.new (), ["teste", "other"])
      b.addBuffer ("ok2", Buffer.new (), ["teste", "other2"])

      b.getBuffersWithKeyword ("other")

      BufferList.new().getBufferWithName (name);
      BufferList.new().getSelectedBuffer (); //returns the buffer that was
      selected in the GUI

      BufferList.new ().removeBufferAt (name, buffer);
```

ContinuousBuffer

One of the first strategies to achieve an expressive dialogue with other musicians, and their sonic material, on significant moments of their performance consisted in the creation of a recorder from which, at any moment, we can extract samples of a passed event. This represents a perceptual shift on the act of sampling from an action that attempts to capture upcoming events towards an action that retrieves significant passed events that can be immediately used.

ContinuousBuffer records audio on two phased buffers and enables us to retrieve segments from them. These samples could be immediately used as material for algorithmic procedures.

```
      ~contBuf=ContinuousBuffer.new ();
      ~contBuf.setCurrentInput (0);
      ~contBuf.startSnapshotBuffer (20, true);//will store the last 20 seconds of
audio

      b=~contBuf.getSnapshot (2); //returns a Buffer with the last 2 seconds
recorded
      b.plot;
      ~bufUtils=BufferUtils.playBuffer (b);
```

BufferUtils

This class contains higher-level methods to operate with sound files.

```
      BufferUtils.writeToFile (buffer, fileName);
      BufferUtils.playBuffer (out, buffer, rate: 1, loop: 0, server);
      BufferUtils.toMono (buffer);
```

GrainEnv

This class contains a series of grain functions that have been used for granular synthesis and waveshaping.

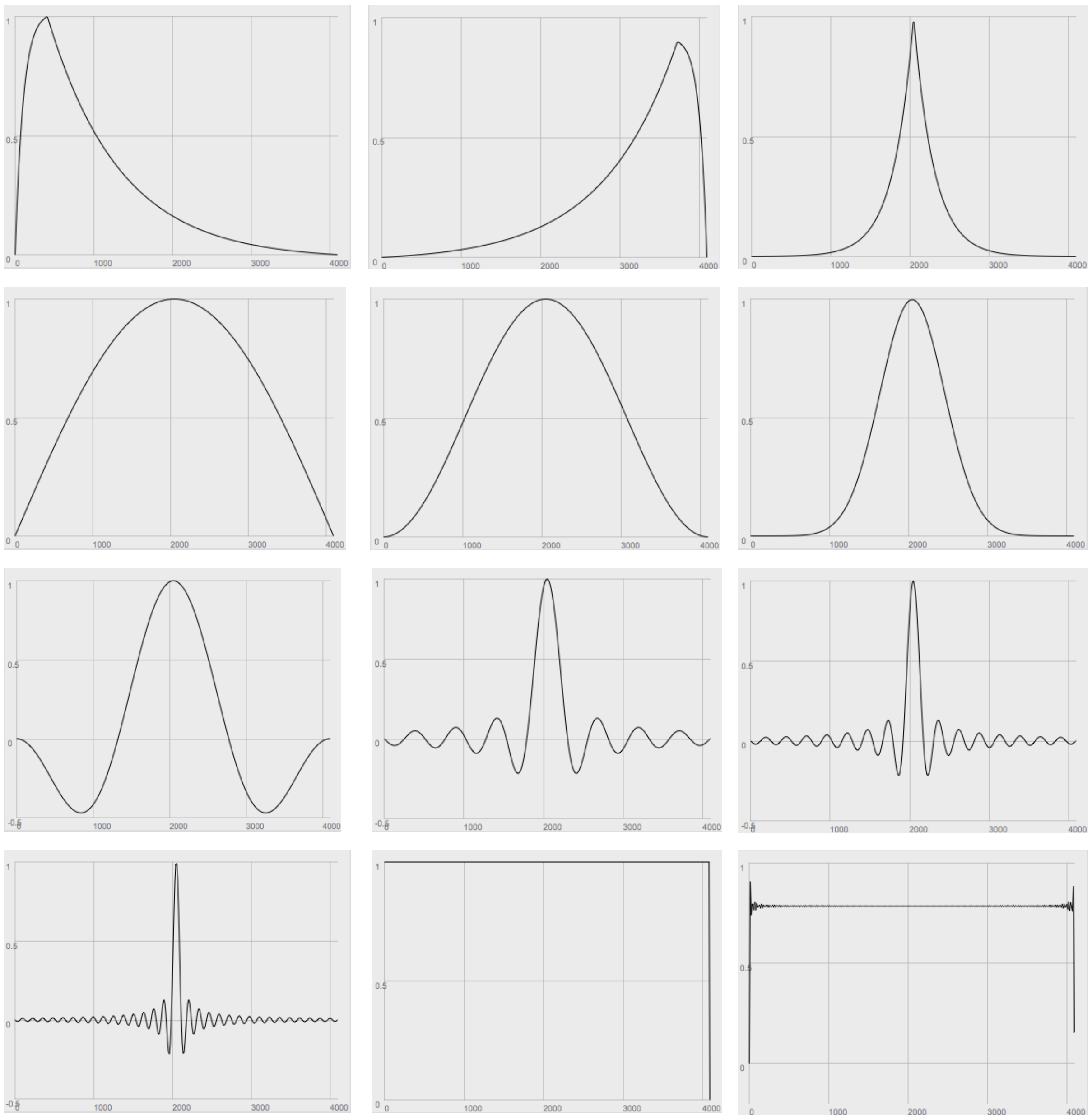


Fig. 26 Implemented grain envelope functions: Expodec, rexpodec, expon, welch, hanning, gaussian, blackman, sinc8, sinc16, sinc23, rect, pulse

5.7 Summary

The *Thr44 framework* was developed in order to design and experiment on approaches that can contribute to the improvement of the relations between the composer and his work, in the context of interactive music practice.

This work was developed under the assumption that one could develop a more subjective and dynamic context for music creation by engaging in a computational practice and by relating interactive programming approaches, such as live-coding, with the beforehand development of a set of objects and behaviours that could be articulated in during a performance. Strategies and paradigms, like for instance prototype-based programming, would allow us to create such entities as modular components to be used and modified *on-the-fly*, facilitating not only the prototyping of ideas but also improving the scope of compositional decisions that could be made in a performance.

We have established a set of criteria that guided the development of this work, identified as *versatility* (the easiness for the description of ideas), *transparency* (related to the perception and cognition of the system), *flexibility* (the system's susceptibility of change and adaptation) and *expressiveness* (the ability to convey intentions or emotions; to communicate).

Regarding *versatility*, we have to recognise that such criteria is more related to the choice of the musical environment that is to be used rather than to the actual development of the framework itself. The choice of *SuperCollider* deeply facilitated the implementation of this work, however, as we progressed with its development and use, we found limitations in its *IDE* (integrated development environment), such as its limited abilities for code hinting and completion. Also, *SuperCollider*'s library is class-based, raising difficulties in the organisation of the system's data structures in the context of real-time, interactive music (cf. Ch. 4). As a consequence, a particular issue that relates to the implementation of *Odef* lies in the fact that we have created a language interpreter that still requires further testing and development in order to become a stable entity. These issues still affect the versatility of our work, opening possibilities for future work.

Considering the requirement of *transparency*, we have just begun to understand the advantages of interactive programming and to draw the significant aspects that it can contribute to interactive music practices. As an example, the use of reflexive mechanisms (the ability to ask the system for the identification of its variables and functions) enables one to access and perceive every process running in the system improving the discourse that can be established with the computer. Such approach has been frequently used in this work, in particular within the implementation of the *Odef*. We are not only able to inspect the existing *Odef* instances for their statuses, available parameters or source code, but also locate and identify them at any moment (by calling the *all* method one can get an index of every available *Odef*).

The efforts towards *flexibility* can be evidenced by the articulation that has been established between the *Odef* and the *Networks*' and *Particles*' structures. At one end, the *Odef* enables the description, modification and execution of processes in real-time, enclosing it in an ever-accessible placeholder. On the other end, the Networks' and Particles' classes allow the creation of complex generative processes that are not adequate to implement in the context of a performance, but can easily be accessed, modelled and mapped to the dynamic sonic entities enclosured in *Odef* instances to control their behaviour. In this chapter we have provided some examples that illustrate this approach, such as the work developed with PetriNets.

Attempting to improve on the *expressive* potential of this work, we have developed the *SCPad!* as an extension of the *Odef*. Instead of mapping sonic processes to graphic user interfaces (GUI), a task that is usually carried out manually, the *SCPad!* dynamically creates and modifies its GUIs by constantly listening to the instances of Odef that one has instantiated. A similar approach could be carried out to improve gesture in direct relation with *live-coding*. In the same sense, other *intelligent* systems can be developed to assist other aspects of compositional practice in performance.

Concerning the limitations of this work, the implemented objects pertain to a personal aesthetic that others may find too narrowing for their own practice. However, all the materials that have been developed are open source and can be accessed on *Github*, a platform that favours collaborative software development, which facilitates our work's appropriation and further development.

Addressing the technologies that we have used, *SuperCollider* continues to be our choice of language for real-time implementation, however we long for a language that fully implements the *prototype-based* paradigm and for an IDE that can provide better coding support, namely code-completion and hinting. Also, an interesting improvement would consist of finding the means to write 'white-box' unit generators, controlling sounds at the level of the sample at run-time, without resourcing from an outside language[135]. As for the development of *SCPad!* we would favour the use of *Objective-C*, which would not only facilitate the application's maintenance, but also take full advantage of *iOS SDK*.

We have designed a computational framework for music practice that certainly presents an interaction logic of its own. Our compositional strategy develops on a heuristic search for interesting code snippets for sound and behaviour that cumulate through a continuous discourse with the computer. From language we create sonic entities that are set in a particular ecosystem, subjected to our own physical laws; we create interfaces that enable us to interact with these entities through actions that range from overall tendency values to each sonic element's detailed description.

Recalling Truax's levels[136] at which the composer can operate with interactive music systems, this work enabled a vast scope of action that we'll certainly carry on pursuing. Although we consider this framework to be a personal creative work, our aim is that it can contribute to the discussion of computer-based interactive music practices. The learning experience that involved the creation of *Thr44* is beyond the discussed implementation details and therefore its source code has been made publicly available[137].

---

[135] Such as the *C* programming language used for the *UGen* implementation in *SuperCollider*.

[136] Cf. footnote 108, page 84.

[137] https://github.com/Thr44/

CONCLUSION

I dream of instruments obedient to my thought and which with their contribution of a whole new world of unsuspected sounds, will lend themselves to the exigencies of my inner rhythm.
(Varèse, 1917, qtd. in Holmes, 2008:3)

This work proposed an investigation on computer-based music creation, in particular within the context of interactive music. Our goal was practice oriented, directed towards the establishment of a methodological framework that could facilitate our compositional activity.

As computers become more pervasive, a diversity of tools proliferate, making sonic creation seem easier than ever. At present, very little knowledge of sound theory is required to create a satisfying aural experience. However, if one is attempting to extend one's creative capacity and access differentiated experiences with interactive music, these tools reinforce prescribed models and solutions by making their inner processes opaque and inaccessible. Rather than a producer, one feels like a consumer of an interactive work.

In this perspective, this work evolved as a contribute to the research on computer-based interactive music by attempting to identify and experiment on new methodologies that could improve compositional practice to be a more subjective, dynamic and transparent process.

Our objectives were pursued by following an action-based methodology, and therefore, this work was driven by the creation of musical projects by means of developing and using an interactive music system. Such direction implies a questioning of the models, techniques and concepts that are fundamental within this context in order to identify trends that can contribute to the improvement of computer-based interactive music practice.

Acknowledging that computer music inherits conceptualizations that emerged long before the advent of mechanical computers, we began by surveying the relationship between music, science and technology.

A particular aspect concerns the formalization of processes, often regarded as some technical matters exclusive to the domain of computer science. We have identified a diversity of approaches that employed combinatorics and chance procedures aiming to facilitate the generation of sonic material. Also, the intention of delegating compositional ideas to mechanical devices was already present in the middle ages, naturally progressing towards the creation of music-machines and the conceptualization of automata for composition. The influence of science is particularly evident since the turn of the 20th Century as composers have become more aware of surrounding phenomena and attempted to materialize them into musical works. The romantic view of the composer as a genious creating masterpieces, addressing values of harmony, equilibrium and stability, is abandoned in favor of scientific logics, as pursued by Schenberg, Schillinger or Messiaen. This perspective is also present in the formal approaches to composition that were carried out by Iannis Xenakis, who employed stochastics for the control of complex masses of sound particles, or by John Cage, whose efforts were directed towards the awareness of everyday sounds as an enriching manifestation of life.

Another aspect that contributed to the redefinition of compositional practice is reflected in the exploration of technological means to register and reproduce sounds. Whereas previously constrained to the manipulation of symbols that mediated sonic events that would later manifest in a performance, this new direction opened the possibility to operate directly on sonic material, enabling the autonomy of compositional practice and the expansion of the long established set of orchestral sounds to a new repertoire of timbres.

Such technological changes enforced a reconceptualization of the nature of musical materials. The attributes of the note, such as tone, duration and amplitude, and the organization strategies that were characteristic in a tradicional musical score lost their primacy in favor of the complexities of timbre. In fact, the incompatibility between form (composing with sound) and material (composing the sound) prevailed in subsequent years, particularly with Pierre Schaeffer's notion of *Sound Object* in opposition to *Elektronische Musik*.

Turning towards the computational medium, we regarded how these machines contributed to a reaffirmation of formal approaches that had previously been carried out. The exploration of computers as symbolic operators not only represented an economy of means, freeing the composer from laborious work, but also empowered their scope of action, enabling the development of complex formulations such as the implementation of Markov chains or Stochastics.

Exploring the possibilities for computers to generate sound, we have identified distinct formulations such as the mathematical approach to sound as a function of pressure variation (Mathews et al., 1969), sound as concatenation of grains (Gabor, 1947) or psychoacoustic approaches (Risset, 2003; Smalley, 1997) that contributed to a deeper understanding of sound.

The use of computers enabled a unique situation, that from the abstraction of programming languages one can generate concrete sound, allowing the confluence of the description and the actual sound in the same entity, partially eliminating the separation between form and material. From algorithmic composition to sound design, we regarded how the incorporation of computers in musical practice enabled the development of computational tools capable of generating complete works.

Directed towards the topic of interaction, we regarded how the exploration of the negative feedback loops proposed by *cybernetics (Wiener, 1948)* represented a paradigmatic shift to the means by which music is conceived. The recognition of an interactive design cycle in composition, rather than a singular one-way determination, opens way to a diversity of approaches to interaction. As computers found their way onto the stage, a diversity of approaches emerged. In this perspective, we have regarded a diversity of proposals in which the computational medium takes the form of an artificial performer, an intelligent instrument or an autonomous ecosystem. We draw attention to the pioneering experiments that were carried out in the early 1970s by the *League of Automatic Music Composers* as they were fundamental in the exploration of communicational aspects between computers and human performers, and to the ecosystemic approaches to interaction that were realised by Agostino Di Scipio that reinforced the need to explore a deeper relationship with the audience and the environment.

In search of more adequate models for music practice and further exploring the possibilities for interaction with the computational medium, we have looked into recent trends in programming languages. Paradigms such as *just-in-time* programming open up the possibility to change a program while it runs whereas *prototype-based* programming brings forth the idea of computation as an extension of our mind, as the reflection of our own reasoning process. Allied to this perspective, the materials that are used in our work are language itself, an abstraction from which one describes concepts that materialize into music.

Driving this research, we have engaged in the creation of a musical work, from which the *Thr44* framework was devised. This work has enabled us to provide a deeper understanding of computational media and its use for interactive music practice. We have identified a field of possibilities within interactive programming and the principles of object-oriented composition that allowed the design and implementation of a computational system that was created and experimented while creating a musical project.

The title of this study – ***Composing Interactive Music Systems*** – attempts to reflect a perspective in which the development of a system for music practice is in itself a compositional act, not only emphasizing the relevance of the computational media in this activity, but also questioning the notion of composition itself.

In this perspective, the role of the composer blends with the performer, the developed work can rather be understood as an activity. Interaction emerges as a by-product of procedurality; the composer does not compose the music, but rather the interactions and interdependencies that manifest themselves as sound – an heuristic model where compositional decisions are made through an iterative cycle of implementing and testing musical ideas. In this perspective, the composer creates the possibility for action that can be enacted in a continuum between composing and performing.

This point of view places programming languages at the core of our activity. By describing procedures to be carried out by the machine to execute, code extends towards algorithmic composition, signal processing or interaction design. In this sense, computer music encompasses a vast trans-disciplinary scope of activities into a single one.

We are not regarding the computer as a means to an end, nor as a mere productive tool that provides a wide variety of applications and devices to ease musical creation, and consequently assuming the affordances and constraints it prescribes. Instead, we reiterate the idea that one can relate to the computer by considering it as a medium for creative exploration (cf. Ch.4.2), as an extension of our mind, enabling their integration into our own sensory system.

> All these things head in the same direction, towards a despecification of the instruments, materials and apparatuses specific to different arts, a convergence on a same idea and practice of art as a way of occupying place where relations between bodies, images spaces and times are redistributed.
> (Jacques Rancière qtd. in Stockburger, 2012)

With the presented work, we hope to have contributed to the discussion on interactive music, in particular to the relationships that are established with the computer in order to improve the discourse that can be established with other musicians, the audience and the environment. We not only attempted to provide a theoretical ground for such practices, but also advanced an implementation of a computational system – the *Thr44 framework* - that proposes some alternatives to the dominating practices.

Deriving from the limitiations of the *Thr44 framework*, the implemented work can still undertake a diversity of improvements, although we consider that, according to the set objectives, we achieved satisfying results. As such, it is our intention to continue researching on the development of musical environments for interactive music practices.

We limited this study to sonic phenomena in a performative context, and have not explored other contexts related to sound art, namely physical computing or visualisation. Although we recognize the potential of audio-visuality in interactive practices, and have collaborated with visual performers in our performances, it is a direction we intend to address in future research.

Finally, we were essentially concerned with the development of strategies to facilitate our compositional practice, and therefore the musical outputs were relatively limited. Having found an adequate methodology for our practice we can now focus on extending that work by increasing our performative activity.

> Art, and above all, music has a fundamental function, which is to catalyze the sublimation that it can bring about through all means of expression. It must aim through fixations which are landmarks to draw towards a total exaltation in which the individual mingles, losing his consciousness in a truth immediate, rare, enormous, and perfect. If a work of art succeeds in this undertaking even for a single moment, it attains its goal.
> (Xenakis, 1992)

BIBLIOGRAPHY

Ariza, Christopher. (2005). *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL.* (PhD), New York University.

Ariza, Christopher. (2011). Two Pioneering Projects from the Early History of Computer-Aided Algorithmic Composition. *Computer Music Journal, 35*, pp. 40-56.

Arsenault, Linda. (2002). Iannis Xenakis's Achorripsis: The Matrix Game. *Computer Music Journal, 26*(1).

Aycock, John. (2003). A brief history of just-in-time. *ACM Computing Surveys, 35*(2).

Battier, Marc. (2007). What the GRM brought to music: from musique concrète to acousmatic music. *Organised Sound, 12*(3).

Berry, David. (2011). *The Philosophy of Software: Code and Mediation in the Digital Age*. Basingstoke: Palgrave Macmillan.

Blum, Stephen. (2001). Composition. The New Grove Dictionary of Music and Musicians: Oxford University Press.

Bolter, David Jay, & Grusin, Richard. (2000). *Remediation, Understanding New Media*. Cambridge, Massachusetts: The MIT Press.

Booch, Grady. (1998). *Object Oriented Analysis and Design with Application* (2nd ed.). Santa Clara, California: Addison-Wesley.

Boulez, Pierre. (1987). Timbre and Composition. *Contemporary Music Review, 2*.

Boulez, Pierre, Noakes, David, & Jacobs, Paul. (1964). Alea. *Perspectives of new music, 3*, pp. 42-53.

Brooks, Frederick P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE computer*, pp. 1-14.

Brown, Andrew R. (2004). An aesthetic comparison of rule-based and genetic algorithms for generating melodies. *Organised Sound, 9*, pp. 191-197.

Brown, Chris, & Bischof, John. (2005). Computer Network Music Bands: a history of the League of Automatic Music Composers and The Hub. In A. Chandler & N. Neumar (Eds.), *At a Distance: Precursors to Art and Activism on the Internet*. Cambridge, Massachusetts: The MIT Press.

Budón, Osvaldo. (2000). Composing with Objects, Networks, and Time Scales: An Interview with Horacio Vaggione. *Computer Music Journal, 24*(3), pp. 9 - 22.

Busoni, Ferruccio. (1911). *Sketch of a new esthetic of music*.

Buxton, William. (1977). A Composer's Introduction to Computer Music. *Interface, 6*.

Buxton, William, Reeves, William, Baecker, Ronald, & Mezei, Leslie. (1978). The Use of Hierarchy and Instance in a Data Structure for Computer Music. *Computer Music Journal, 2*, pp. 10-20.

Cage, John. (1968). *Silence: Lectures and Writings by John Cage*: Wesleyan University Press.

Caires, Carlos. (2006). *Algorithmes de composition: Exemples d'outils informatiques de génération et manipulation du matériau musical.* Universite Paris VIII Paris.

Cardoso, Miguel. (2013). Mediation with Computers in Music Practice. In M. Carvalhais & P. Tudela (Eds.), *Mono #2 Cochlear Poetics: Writings on Music and Sound Arts*. Porto: i2ADS / FBAUP, pp. 121-135.

Carvalhais, Miguel. (2010). *Towards a Model for Artificial Aesthetics: Contributions to the Study of Creative Practices in Procedural and Computational Systems.* (PhD), Faculdade de Belas Artes da Universidade do Porto.

Cascone, Kim. (2000). The Aesthetics of Failure: 'Post-digital' Tendencies in Computer Music. *Computer Music Journal, 24*(4), pp. 12-18.

Casserley, Lawrence. (1998). A digital signal processing instrument for improvised music. *Journal of Electroacoustic Music, 11*, pp. 1-10.

Chadabe, Joel. (1984). Interactive Composing: An Overview. *Computer Music Journal, 8*(1).

Chadabe, Joel. (1997). *Electric Sound: The Past and Promise of Electronic Music.* New Jersey: Prentice-Hall.

Chadabe, Joel. (2001). A Statement...
Retrieved 27/07/2013, from http://www.chadabe.com/statement.html

Chadabe, Joel. (2009). The performer is us. *Contemporary Music Review*, pp. 37-41.

Chayka, Kyle. (2012). The New Aesthetic: Going Native. Retrieved 10-09-2013, from http://thecreatorsproject.vice.com/blog/in-response-to-bruce-sterlings-essay-on-the-new-aesthetic

Chion, Michel. (1983). *Guide des Objets Sonores: Pierre Schaeffer et la recherche musicale*. Paris: Éditions Buchet/Chastel.

Coghlan, David, & Brannick, Teresa. (2010). *Doing Action Research in Your Own Organization*. London: Sage Publications, Ltd.

Collins, Nick. (2006). *Towards Autonomous Agents for Live Computer Music: Realtime Machine Listening and Interactive Music Systems.* (PhD), University of Cambridge.

Collins, Nick. (2011). Live Coding of Consequence. *Leonardo, 44*(3).

Cox, Geoff, & Ward, Adrian. (2008). Perl. In M. Fuller (Ed.), *Software Studies*. Cambridge, Massachusetts: The MIT Press.

Cramer, Florian. (2005). *Words made Flesh: Code, Culture, Imagination.* (PhD), Piet Zwart Institute Rotterdam, Netherlands.

Dannenberg, Roger. (1993). A Brief Survey of Music Representation Issues, Techniques, and Systems. *Computer Music Journal, 17*(3), pp. 20-30.

Davis, Tom. (2010). Complexity as Process: Complexity-inspired approaches to composition. *Organised Sound, 15*, pp. 137-146.

de Campo, Alberto. (2011). Microsound. In S. Wilson, D. Cottle & N. Collins (Eds.), *The SuperCollider Book*. Cambridge, Massachusetts: The MIT Press, pp. 463-504.

Dean, Roger. (2009). Envisaging Improvisation in Future Computer Music. In R. Dean (Ed.), *The Oxford Handbook of Computer Music*. New York: Oxford University Press, pp. 133-147.

Di Scipio, Agostino. (1994). *Formal Processes of Timbre Composition Challenging the Dualistic Paradigm of Computer Music.* Paper presented at the ICMC Inte'l Computer Music Conference, Helsinki.

Di Scipio, Agostino. (1998). Compositional Models in Xenakis's Electroacoustic Music. *Perspectives of New Music, 36*(2), pp. 201-243.

Di Scipio, Agostino. (2001). Iterated Nonlinear Functions as a Sound-Generating Engine. *Leonardo, 34*(3), pp. 249-254.

Di Scipio, Agostino. (2002). Systems of Embers, Dust and Clouds: Observations after Xenakis and Brün. *Computer Music Journal, 26*, pp. 22-32.

Di Scipio, Agostino. (2003). 'Sound is the interface': from interactive to ecosystemic signal processing. *Organised Sound, 8*, pp. 269-277.

Di Scipio, Agostino. (2010). The Synthesis of Environmental Sound Textures by Iterated Nonlinear Functions, and its Ecological Relevance to Perceptual Modeling. *Journal of New Music Research*, pp. 37-41.

Dias, António Sousa. (2005). *L'Object Sonore: Situation, Évaluation et Potentialités.* (PhD), Université Paris 8

Döbereiner, Luc. (2010). *Model and Material, Composing sound and the Construction of Compositional Models.* (Master's), Institute of Sonology, The Hague.

Doornbusch, Paul. (2004). Computer Sound Synthesis in 1951: The Music of CSIRAC. *Computer Music Journal, 28*(1).

Drummond, Jon. (2009). Understanding Interactive Systems. *Organised Sound, 14*.

Dunn, David (Ed.). (1992). *Pioneers of Electronic Art*. Linz: Ars Electronica.

Eco, Umberto. (1989). *The Open Work*: Harvard University Press.

Edmonds, Ernest. (2009). *On New Constructs in Art: Structure, Time, Correspondences and Interaction*. Paper presented at the Electronic Visualisation and the Arts, London.

Eigenfeldt, Arne. (2007). Real-time Composition or Computer Improvisation? A composer's search for intelligent tools in interactive computer music. *Proceedings of the Electronic Music Studies*.

Essl, Karlheinz. (2007). Algorithmic Composition. In N. Collins & J. d'Escriván (Eds.), *Cambridge Companion to Electronic Music*. Cambridge, New York: Cambridge University Press.

Feenberg, Andrew. (1991). *Critical Theory of Technology*. New York: Oxford University Press.

Fishwick, Paul A (Ed.). (2006). *Aesthetic Computing*. Cambridge, Massachusetts: The MIT Press.

Flusser, Vilém. (2011). *Does Writting Have a Future?* Minneapolis: University of Minnesota Press.

Friedman, Alan, & Donley, Carol. (1985). *Einstein as Myth and Muse*: Cambridge University Press.

Gabor, D. (1947). Theory of Communication. *The Journal of the Institution Of Electrical Engineers, 93*, pp. 429-457.

Galanter, Philip. (2003). What is Generative Art ? Complexity Theory as a Context for Art Theory. *6th Generative Art Conference*.

Goldin, Dina, & Wegner, Peter. (2007). The Interactive Nature of Computing: Refuting the Strong Church-Turing Thesis. *Minds and Machines, 18*, pp. 1-26.

Goldin, Dina, Wegner, Peter, & Smolka, Scott. (2006). *Interactive Computation: The New Paradigm*. Berlin: Springer-Verlag.

Green, Owen. (2006). More than 'Just a Hammer': Critical Techniques in Electroacoustic Practice. *SoundAsArt*. Aberdeen.

Guerreiro, Ricardo. (2015). *Redes Mediadas por Computador na Composição e Performação de Situações Musicais Interactivas.* (PhD), Universidade Católica do Porto.

Hamman, Michael. (2002). From technical to technological: The imperative of technology in experimental music composition. *Perspectives of New Music, 40*(1).

Harkins, James. (2011). High-Level Structures for Live Performance: dewdrop_lib and chucklib. In S. Wilson, D. Cottle & N. Collins (Eds.), *The SuperCollider Book*. Cambridge, Massachusetts: The MIT Press.

Harley, James. (2004). *Xenakis: His Life in Music*: Routledge Press.

Harley, James. (2009). Computational Approaches to Composition of Notated Instrumental Music: Xenakis and the Other Pioneers. In R. Dean (Ed.), *The Oxford Handbook of Computer Music*. New York: Oxford University Press, pp. 109-147.

Hayles, Katherine. (2005). *My Mother was a Computer: Digital Subjects and Literary Texts*: University of Chicago Press.

Hayles, Katherine. (2008). *How we became posthuman: virtual bodies in cybernetics, literature, and informatics*. Chicago: The University of Chicago Press.

Heylighen, Francis, & Joslyn, Cliff. (2001). Cybernetics and Second-Order Cybernetics. In R. Meyers (Ed.), *Encyclopedia of Physical Science & Technology* (3rd ed.). New York: Academic Press.

Hoffmann, Peter. (2010). Towards an "Automated Art ": Algorithmic Processes in Xenakis' Compositions. *Contemporary Music Review*, pp. 37-41.

Hofstadter, Douglas. (1979). *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Basic Books.

Hofstadter, Douglas. (1995). *Fluid Concepts & Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. New York: Basic Books.

Holmes, Thom. (2008). *Electronic and Experimental Music: Technology, Music, and Culture* (3rd ed.): Routledge.

Holtzman, Steve. (1979). An Automated Digital Sound Synthesis Instrument. *Computer Music Journal, 3*(2), pp. 53-61.

Howe, Hubert. (2009). My Experiences with Max Mathews in the Early Days of Computer Music. *Computer Music Journal, 33*(3), pp. 41–44.

Hugill, Andrew. (2007). The origins of electronic music. In N. Collins & J. d'Escriván (Eds.), *Cambridge Companion to Electronic Music*. Cambridge, New York: Cambridge University Press.

Impett, Jonathan. (1994). A Meta-trumpet(er). *ICMC Proceedings*.

Impett, Jonathan. (2001). Interaction, simulation and invention: a model for interactive music. *Proceedings of the workshop on artificial life models for musical applications*. Cosenza.

James, Richard Schmidt. (1981). *Expansion of sound resources in France, 1913-1940, and its relationship to electronic music.* University of Michigan.

Jordá, Sergi. (2005). *Digital Lutherie, Crafting musical computers for new musics' performance and improvisation.* (PhD), Universitat Pompeu Fabra, Barcelona.

Kay, Alan. (1993). The Early History of Smalltalk. *ACM SIGPLAN Notices, 28*(3).

Kernighan, Brian, & Ritchie, Dennis. (1988). *The C Programming Language* (2nd ed.): Prentice Hall, Inc.

Kersten, Stefan, Baalman, Marije, & Bovermann, Till. (2011). Ins and Outs: SuperCollider and External Devices. In S. Wilson, D. Cottle & N. Collins (Eds.), *The SuperCollider Book*. Cambridge, Massachusetts: The MIT Press.

Kittler, Friedrich. (1995). There is No Software.
Retrieved 10-09-2013, from http://www.ctheory.net/articles.aspx?id=74

Koenig, Gottfried Michael. (1975). My Experiences with Programmed Music.
Retrieved 10-09-2013, from http://www.koenigproject.nl

Koenig, Gottfried Michael. (1991). Working With " Project 1 ". My Experiences with Computer Composition. *Journal of New Music Research*, pp. 1-4.

Leiserson, Charles, Rivest, Ronald, Stein, Clifford, & Cormen, Thomas. (2009). *Introduction to Algorithms* (3rd ed.). Cambridge, Massachusetts: The MIT Press.

Lewis, George. (2000). Too Many Notes: Computers, Complexity and Culture in Voyager. *Leonardo Music Journal, 10*.

Lieberman, Henry. (1982). Machine Tongues IX: Object-Oriented Programming. *Computer Music Journal, 6*(3).

Loy, Gareth. (1989). Composing with Computers – A Survey of Some Compositional Formalisms and Music Programming Languages. In M. Mathews & J. Pierce (Eds.), *Current Directions in Computer Music Research*. Cambridge, Massachusetts: The MIT Press.

Loy, Gareth, & Abbott, Curtis. (1985). Programming languages for computer music synthesis, performance, and composition. *ACM Computing Surveys, 17*, pp. 235-265.

Magnusson, Thor. (2009). Of Epistemic Tools: musical instruments as cognitive extensions. *Organised Sound, 14*.

Manning, Peter. (2003). The influence of recording technologies on the early development of electroacoustic music. *Leonardo Music Journal, 13*.

Manning, Peter. (2009). Sound Synthesis using Computers. In R. Dean (Ed.), *The Oxford Handbook of Computer Music*. New York, pp. 85-105.

Manousakis, Stelios. (2009). Non-Standard Sound Synthesis with L-Systems. *Leonardo, 19*.

Marino, Mark C. (2006). Critical Code Studies. Retrieved 10-09-2013,
from http://www.electronicbookreview.com/thread/electropoetics/codology

Mathews, Max. (1963). The Digital Composer as a Musical Instrument. *Science, 142*(11).

Mathews, Max, Miller, Joan, Moore, Roger, Pierce, John, & Risset, Jean Claude. (1969). *The Technology of Computer Nusic*. Cambridge, Massachusetts: The MIT Press.

Mathews, Max, & Moore, Roger. (1970). GROOVE A Program to Compose, Store, and Edit Functions of Time. *Communications of the ACM, 13*(12), pp. 715-721.

Mathews, Max, & Pierce, John (Eds.). (1989). *Current Directions in Computer Music Research*. Cambridge, Massachusetts: The MIT Press.

Maturana, Humberto, & Varela, Francisco. (1980). *Autopoiesis and Cognition: The Realization of the Living*. Dordrecht, Holland: Reidel Publishing.

McCartney, James. (1996). SuperCollider: a new real time synthesis language. *Proceedings of the 1996 International Computer Music Conference*: International Computer Music Association.

McCartney, James. (2002). Rethinking the computer music language: Supercollider. *Computer Music Journal, 26*(4), pp. 61-68.

McLean, Alex. (2004). Hacking Perl in Nightclubs. Retrieved 10-09-2013, from http://www.perl.com/pub/2004/08/31/livecode.html

McLean, Alex. (2011). *Artist-Programmers and Programming Languages for the Arts.* (PhD), University of London.

Mcwilliams, Chandler B. (2009). *The Other Software.* Paper presented at the Digital Arts and Culture Conference, California.

Minsky, Marvin. (1988). *Society of Mind*. New York: Simon & Schuster, Inc.

Miranda, Eduardo. (2002). *Composing Music with Computers*: Focal Press.

Montfort, Nick. (2008). Obfuscated Code. In M. Fuller (Ed.), *Software Studies*. Cambridge, Massachusetts: The MIT Press.

Mumma, Gordon. (1967). Creative Aspects of Live-Performance Electronic Music Technology. pp. 12-13.

Paine, Garth. (2009). Gesture and Morphology in Laptop Music Performance. In R. Dean (Ed.), *The Oxford Handbook of Computer Music*. New York: Oxford University Press, pp. 214-232.

Palisca, Claude. (2001). Guido of Arezzo [Aretinus]. The New Grove Dictionary of Music and Musicians: Oxford University Press.

Panken, Ted, & Lewis, George. (2009). In Conversation with George Lewis. Retrieved 27-07-2013, from http://www.jazz.com/features-and-interviews/2009/12/11/in-conversation-with-george-lewis

Penny, Simon (Ed.). (2005). *Critical Issues in Electronic Media*. New York: State University of New York Press.

Perkins, Tim. (2002). *Complexity and Emergence in the American Experimental Music Tradition*. Santa Fe Institute.

Perkins, Tim. (2009). Some Notes on my Electronic Improvisation Practice. In R. Dean (Ed.), *The Oxford Handbook of Computer Music*. New York: Oxford University Press, pp. 161-165.

Petri, Carl, & Reisig, Wolfgang. (2008). Petri net. *Scholarpedia.* 3, Retrieved 10-09-2013, from http://www.scholarpedia.org/article/Petri_net

Pope, Stephen. (1993). Machine Tongues XV: Three Packages for Software Sound Synthesis. *Computer Music Journal, 17*(2).

Pressing, Jeff. (1987). Improvisation: Methods and Models. In J. Sloboda (Ed.), *Generative processes in Music*: Oxford University Press.

Puckette, Miller. (1988). *"The Patcher"*. Paper presented at the International Computer Music Conference, San Francisco.

Puckette, Miller. (2002). Max at Seventeen. *Computer Music Journal, 26*, pp. 31-43.

Ribas, Luísa. (2011). *The Nature of Sound-image Relations in Digital Interactive Systems.* (PhD), Faculdade de Belas Artes do Porto.

Risset, Jean Claude. (2003). The Perception of Musical Sound. *Computer music: Why*, pp. 1-12.

Risset, Jean-Claude. (1976). Computer Music, Why?

Risset, Jean-Claude. (2005). Horacio Vaggione: Toward a Syntax of Sound. *Contemporary Music Review, 24*(4/5).

Risset, Jean–Claude. (1985). Digital techniques and sound structure in music. In C. Roads (Ed.), *Composers and the Computer*. Madison: A-R Editions.

Roads, Curtis. (1999). *The Computer Music Tutorial*. Cambridge, Massachusetts: The MIT Press.

Roads, Curtis. (2004). *Microsound*. Cambridge, Massachusetts: The MIT Press.

Roads, Curtis. (2005). The Art of Articulation: The Electroacoustic Music of Horacio Vaggione. *Contemporary Music Review, 24*(4/5).

Roads, Curtis, & Mathews, Max. (1980). Interview with Max Mathews. *Computer Music Journal, 4*(4), pp. 15-22.

Robinson, Julia (Ed.). (2009). *The Anarchy of Silence. John Cage and Experimental Art.* Barcelona: Museu d'Art Contemporani de Barcelona.

Robinson, Julia, Bois, Yve-Alain, Kotz, Liz, & Joseph, Branden. (2009). *The Anarchy of Silence. John Cage and Experimental Art*. Barcelona: Museu d'Art Contemporani de Barcelona.

Rohrhuber, Julian, & Campo, Alberto de. (2009). Improvising Formalisation - Conversational Programming and Live Coding. In G. Assayag & A. Gerzso (Eds.), *New Computational Paradigms for Computer Music*: Delatour France / Ircam-Centre Pompidou.

Rohrhuber, Julian, & Campo, Alberto de. (2011). Just-in-Time Programming. In S. Wilson, D. Cottle & N. Collins (Eds.), *The SuperCollider Book*. Cambridge, Massachusetts: The MIT Press.

Rohrhuber, Julian, Collins, Nick, McLean, Alex, & Ward, Adrian. (2003). Live coding in laptop performance. *Organised Sound, 8*(3), pp. 321-330.

Rohrhuber, Julian, de Campo, Alberto , Wieser, Renate, van Kampen, Jan-Kees , Ho, Echo , & Hölzl, Hannes (2007). *Purloined Letters and Distributed Persons.* Paper presented at the Music in the Global Village Conference, Budapest.

Rowe, Robert. (1993). *Interactive Music Systems: Machine Listening and Composing*. Cambridge, Massachusetts: The MIT Press.

Rowe, Robert. (2001). *Machine Musicianship*. Cambridge, Massachusetts: The MIT Press.

Ryan, Joel. (n.d.). As If ByMagic. Retrieved 20-11-2011, from http://jr.home.xs4all.nl/MusicInstDesign.htm

Schaeffer, Pierre. (1966). Traité des Objets Musicaux. *Editions du Seuil. Paris. France.*

Schafer, Raymond Murray. (1979). *Le paysage sonore. Toute l'histoire de notre environnement sonore a travers les âges*.

Schnell, Norbert, & Battier, Marc. (2002). Introducing Composed Instruments , Technical and Musicological Implications. *Proceedings of the 2002 Conference on New Instruments for Musical Expression (NIME-02)*. Dublin.

Schoenberg, Arnold. (1975). *Style and Idea: Selected Writings of Arnold Schoenberg* (L. Stein Ed.): University of California Press.

Serafin, Stefania. (2007). Computer generation and manipulation of sounds. In N. Collins & J. d'Escriván (Eds.), *The Cambridge Companion to Electronic Music*. Cambridge: Cambridge University Press.

Shannon, Claude. (1948). A mathematical theory of communication.

Simon, Herbert. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge, Massachusetts: The MIT Press.

Smalley, Denis. (1994). Defining timbre — Refining timbre. *Contemporary Music Review, 10*(2), pp. 35-48.

Smalley, Denis. (1997). Spectromorphology: explaining sound-shapes. *Organised Sound, 2*, pp. 107-126.

Smith, Julius O. (1991). *Viewpoints on the History of Digital Synthesis.* Paper presented at the International Computer Music Conference, Montreal.

Smith, Wesley, & Wakefield, Graham. (2009a). Augmenting Computer Music With Just-in-Time Compilation. *Computer*.

Smith, Wesley, & Wakefield, Graham. (2009b). *Computational Composition and Creativity.* Paper presented at the Media Arts Science and Technology Conference, Santa Barbara.

Solomos, Makis. (2005). An Introduction to Horacio Vaggione's Musical and Theoretical Thought. *Contemporary Music Review, 24*(4/5).

Solomos, Makis. (2006). The granular connection (Xenakis , Vaggione , Di Scipio …).

Solomos, Makis. (2007). Espaces Composables, Essais sur la Musique et la Pensée Musicale d' Horacio Vaggione

Spiegel, Laurie. (1992). An Alternative to a Standard Taxonomy for Electronic and Computer Instruments. *Computer Music Journal, 16*.

Stockburger, Axel. (2012). *Post-Media Conditioning*. Paper presented at the Emoção Art.ficial 6.0 Symposion, São Paulo, Brasil.

Stroustrup, Bjarne. (2008). *The C++ Programming Language* (3rd ed.): Addison-Wesley.

Thoresen, Lasse. (2007). Spectromorphological analysis of sound objects: an adaptation of Pierre Schaeffer's typomorphology. *Organised Sound, 12*(2), p. 129.

Truax, Barry. (1976). For Otto Laske: A Communicational Approach to Computer Sound Programs. *Journal of Music Theory, 20*(2), pp. 227-300.

Turing, Alan. (1936). On Computable Numbers, with an Application to the Entscheidungs Problem. *Proceedings of the London mathematical society*.

Turing, Alan. (1950). Computing Machinery and Intelligence. In N. Wardrip-Fruin & N. Montfort (Eds.), *The New Media Reader*. Cambridge, Massachusetts: The MIT Press.

Turkle, Sherry, & Papert, Seymour. (1990). Epistemological Pluralism: Styles and Voices within the Computer Culture. *Signs: Journal of Women in Culture and Society, 16*(1).

Vaggione, Horacio. (1991). A Note on Object-based Composition. *Interface, 20*.

Vaggione, Horacio. (2001). Some Ontological Remarks about Music Composition Processes. *Computer Music Journal, 25*(1), pp. 54-61.

Vaggione, Horacio, & Soulez, Antonia. (2005). Composing, Listening. *Contemporary Music Review, 24*(4/5).

Varela, Francisco. (1991). *The Embodied Mind*. Cambridge, Massachusetts: The MIT Press.

Varèse, Edgard, & Wen-chung, C. (1966). The Liberation of Sound. *Perspectives of new music, 5*(1), pp. 11-19.

Veak, Tyler. (2000). Whose Technology? Whose Modernity? Questioning Feenberg's Questioning Technology. *Science, Technology & Human Values, 25*(2), pp. 226-237.

Wang, Ge. (2008). *The ChucK Audio Programming Language " A Strongly-timed and On-the-fly Environ / mentality ".* (PhD), University of Princeton.

Ward, Adrian, Rohrhuber, Julian, Olofsson, Frederik, & McLean, Alex. (2004). Live algorithm programming and a temporary organisation for its promotion. Retrieved 27-07-2013, from http://www.toplap-org

Wardrip-fruin, Noah. (2006). *Expressive Processing: On Process-Intensive Literature and Digital Media.* (PhD), Brown University.

Wessel, David. (2005). An Enactive Approach to Computer Music Performance. pp. 93-98.

Wiener, Norbert. (1948). *Cybernetics; or control and communication in the animal and the machine.* Cambridge, Massachusetts: The MIT Press.

Winkler, Todd. (2001). *Composing Interactive Music: Techniques and Ideas using Max*. Cambridge, Massachusetts: The MIT Press.

Winograd, Terry. (1979). Beyond programming languages. *Communications of the ACM, 22*(7), pp. 391-401.

Winograd, Terry, & Flores, Fernando. (1986). Understanding computers and cognition. *Norwood, Nova Jersey: Ablex Publishing Corporation*.

Wishart, Trevor. (1996). *On Sonic Art* (S. Emmerson Ed.). Amsterdam, Netherlands: Harwood Academic Publishers.

Wishart, Trevor. (2009). Computer Music: Some Reflections. In R. Dean (Ed.), *The Oxford Handbook of Computer Music*. New York: Oxford University Press, pp. 151-160.

Wittgenstein, Ludwig. (1958). *Philosophical Investigations* (2nd ed.). Oxford: Basil Blackwell Ltd.

Wright, Mathew. (2005). Open Sound Control: an enabling technology for musical networking. *Organised Sound, 10*(3).

Xenakis, Iannis. (1992). *Formalized Music, Thought and Mathematics in Composition*: Pendragon Press.

Zicarelli, David. (1987). M and Jam Factory. *Computer Music Journal, 11*(4).

Zielinski, Siegfried. (2006). *Deep Time of the Media*. Cambridge, Massachusetts: MIT Press.

CITED WORKS

Bach, Carl. (1758). Einfall einin doppelten Contrapunct in der Octave von sechs Tacten zu machen ohne die Regeln davon zu wissen.

Bach, Johann S. (1742-1746). Art of the Fugue. BWV 1080.

Beethoven, Ludvig. (1806). String Quartet No.7, Op.59 nº1.

Behrman, David. (1978). On the Other Ocean. Lovely Music. LCD 1041

Boulez, Pierre. (1958). Third Sonata for Piano. Pierre Boulez and the Piano. Paris: Cybele Records CYBELE 004

Brün, Herbert. (1976-1981). Sawdust.

Buxton, William. (1978). SSSP. Structured Sound Synthesis Project.

Cage, John. (1950-51). Music of Changes. Hat Hut Records ART 133

Cage, John. (1951). Imaginary Landscape nº4.

Cage, John. (1952). 4'33.

Cage, John. (1958-1967). Variations II.

Cage, John. (1958-1967). Variations IV. Los Angeles: Everest 3230

Cage, John. (1960). Cartridge Music.

Cage, John. (1969). HPSCHD.

Chadabe, Joel. (1978). Solo. From the Kitchen Archives. New York: Orange Mountain Music OMM-0015.

Chadabe, Joel. (1987). M. Cycling'74. Accessed 12-10-2014, from https://cycling74.com/products/m/

Gresham-Lancaster, Scot. (1987). Vague Notions of Lost Textures.

Guttman, Newman. (1957). The Silver Scale.

Hiller, Lejaren; Isaacson, Leonard. (1957). "Illiac Suite" Quartet nº 4 for strings.

Impett, Jonathan. (2001). Meta-Trumpet.

Ives, Charles. (1916). Symphony nº4.

Koenig, Gottfried. (1964). Project 1. Accessed 12-10-2014, from http://www.koenigproject.nl/

Koenig, Gottfried.(1966-1967). Terminus II. Amsterdam: Haast Records BVHaast 9001/2.

Koenig, Gottfried. (1967-1969). Funktion Grau. Amsterdam: Haast Records BVHaast 9001/2.

Koenig, Gottfried. (1978). Sound Synthesis Program. Accessed 12-10-2014, from http://www.koenigproject.nl/

Lewis, George. (1986-1995). Voyager.

Lucier, Alvin. (1969). I Am Sitting in a Room.

Magnusson, Thor; David Bausola and Enrike Hurtado. (2005-). IXI audio software. Accessed 18-02-2015, from http://www.ixi-software.net/content/software.html

Mark Trayle. (1987). Simple Degradation. The Hub. Boundary Layer. New York: Tzadik TZ 8050-3.

Mathews, Max, & Moore, Roger. (1970). GROOVE.

McLean, Alex. (2004). Feedback.pl. Accessed 18-02-2015, from http://www.perl.com/pub/2004/08/31/livecode.html

McLean, Alex. (2004). Texture. Accessed 18-02-2015, from https://github.com/yaxu/

McLean, Alex. (2004). Tidal. Accessed 18-02-2015, from http://tidal.lurk.org/

Mozart, Wolfgang. (1787). Musikalisches Würfelspiel.

Mumma, George. (1967). Hornpipe. Live-Electronic Music. New York: Tzadik TZ 7074.

Perkins , Tim. (1991).Waxlips. The Hub. Boundary Layer. New York: Tzadik TZ 8050-3.

Pousseur, Henri. (1957). Scambi.

Puckette , Miller. (1988). Max. Cycling'74. Accessed 12-10-2014, from https://cycling74.com/

Risset, Jean-Claude. (1969). Mutations. Paris: INA_GRM. INA 1003.

Risset, Jean-Claude. (1977). Inharmonique. Paris: INA_GRM. INA 1003.

Roads, Curtis. *(2001-2003). Volt Air. Point Line Cloude. San Francisco: Asphodel. ASP 3000.*

Rowe, Robert. (1993) Cypher. *Interactive Music Systems: Machine Listening and Composing*. Cambridge, Massachusetts: The MIT Press.

Scalleti, Carla. (1987). Kyma. Accessed 20-02-2015, from http://kyma.symbolicsound.com/

Schumann, Robert. (1829). Abegg Variations.

Spiegel, Laurie. (1986). Music Mouse. Accessed 20-02-2015, from http://retiary.org/ls/programs.html

Stockhausen, Karlheinz. (1956). KlavierStück XI.

Stone, Phil. (1989). Borrowing and Stealing. The Hub. Boundary Layer. New York: Tzadik TZ 8050-3.

Truax, Barry. (1972). POD. POisson Distribution.

Vaggione , Horacio. (2001). *24 Variations. Contemporary Music Review, 24*(4/5).

Varèse, Edgard (1958). Poéme electronique. OHM+: The Early Gurus of Electronic Music: 1948-1980. Ellipsis Arts CD3690.

Weizenbaum, Joseph. (1966). Eliza.

Xenakis, Iannis. (1954). Pithoprakta. Paris: Le Chant Du Monde LDC 278368.

Xenakis, Iannis. (1955). Metastasis. Paris: Le Chant Du Monde LDC 278368.

Xenakis, Iannis. (1957). Achorripsis. Orchestral Works Vol. 5. Timpani 1C1113.

Xenakis, Iannis. (1957-1958). Diamorphoses. Electronic Music. EMF CD 003.

Xenakis, Iannis. (1958). Duel.

Xenakis, Iannis. (1958). ConcretPH. Electronic Music. EMF CD 003.

Xenakis, Iannis. (1960). Orient-Occident. Electronic Music. EMF CD 003.

Xenakis, Iannis. (1962). Bohor. Electronic Music. EMF CD 003.

Xenakis, Iannis. (1962). Stochastic Music Program.

Xenakis, Iannis. (1977-1994). UPIC. Unité Polyagogique Informatique du CEMAMu.

APPENDIX

Appendix 1: Research Results

Software

SCPad!, iOS Application available at http://itunes.apple.com/tr/app/scpad!/id521113570

Thr44 Framework, available at https://github.com/Thr44

Works

Variable Laptop Orchestra from CITAR (Cardoso, M. et al.), Zíngaro, C., (music) in Festival Black & White, Universidade Católica Portuguesa - Escola das Artes, Porto, April 2009

Variable Laptop Orchestra from CITAR (Cardoso, M. et al.) (music) (De) Criando... à sombra de Oliveira in Festival Black & White, Universidade Católica Portuguesa - Escola das Artes, Porto, April 2009.

Variable Laptop Orchestra from CITAR (Cardoso, M. et al.), Orquestrutópica, (music for) Luis Ferreira, J., "Limits, Capacities", Isto não é um Concerto, Centro Cultural de Belém, Lisbon, January 2009.

Variable Laptop Orchestra from CITAR (Cardoso, M. et al.), Orquestrutópica, (music for) Júlio Lopes, J., Ferreira Lopes, et al., Transfronteiras, Culturgest, Lisbon, September 2009.

Reinhold Friedl Ensemble (Cardoso, M. et al), Friedl, Reinhold (música) in Festival Metasonic, Goëthe Instituit, Lisbon, April 2010.

2+n (Cardoso, M., Guerreiro, R.) (música), Gonçalves, A., (imagem) in Granular na Arthobler - Arthobler, Ler Devagar, Lisbon, September 2010

Cardoso, M., Guerreiro, R., Lopes, P. (music for) Ruttmann W. Berlin, Die Sinfonie der Großstad - Göethe Institut, Lisbon, September 2010

2+n (Cardoso, M., Guerreiro, R.) (música), Gomes, J., (imagem) in Ciclo Vinte e Sete Sentidos, Granular – Culturgest, Lisbon, June 2011

2+n (Cardoso, M., Guerreiro, R.); Fernandes, H.; Costa, G. (música), in MicroVolumes 2.2, Sonoscopia, Porto, December 2009

2+n (Cardoso, M., Guerreiro, R.) (música), in RadiaLx 2012, Festival de Arte Radio, Flausina, Lisbon, June 2012

Activities

Cardoso, M., "Visualização Dinâmica de Informação", Ciclo de Conferências: Design de Informação, Faculdade de Belas Artes, Lisbon, May 2011

Cardoso, M., "Interaction and Complexity in Information Visualization", Colóquio Internacional: Image in Science and Art, Fundação Calouste Gulbenkian, Lisbon, February 2011

Cardoso, M., Guerreiro, R., Silva, A., "PetriNet.sc, a PetriNet based class for SuperCollider", SuperCollider Symposium 2012, Queen Mary University, London, April 2012

Appendix 2: Media Content

audio

/audio/Live at Back & White Festival, Porto 2009.mp3

Recording of performance by Variable Laptop Orchestra and Carlos Zíngaro at Festival *Black & White*, Universidade Católica Portuguesa - Escola das Artes, Porto, 2009

/audio/MicroVolumes 2.2.wav

Recording of performance by Miguel Cardoso and Ricardo Guerreiro (2+n) with Gustavo Costa and Henrique Fernandes at Sonoscopia, Porto, December 2009

/audio/n=0.aif

Demo session by Miguel Cardoso and Ricardo Guerreiro (2+n), Lisbon, March 2012

software

SCPad! 1.0

/source/SCPad/SCPad/

The source code of the published version of SCPad!

Thr44 C Library

/source/SCPad/Thr44-C-Lib/

Sources of a resource library, developed in C++.

/documentation/Thr44-C-Lib-doc/index.html

Generated documentation of the Thr44 C Library.

Thr44 Framework Supercollider classes

/sources/Thr44-SC-quark/

Sources of the *Thr44* class library for *SuperCollider.*

/documentation/Thr44SCquark/Thr44.html

Documentation of the *Thr44* class library for *SuperCollider.*

video samples

/software/video samples/scpadDemo.mov

An example of *SCPad!* dynamically generating GUIs, as *Odef* snippets are executed in *SuperCollider*.

/software/video samples/petriExample.mov

A demonstration video of the use of PetriNets.

/software/video samples/boidsSpacializationExample.mov

A basic example of mapping between *Thr44Node* and *Odef* instances.

publications

/*MiguelCardoso_phD.pdf*

A full version of this dissertation.

(blank page)