Ministero dell'Università e della Ricerca
Alta Formazione Artistica, Musicale e Coreutica

Conservatorio di Musica "G. B. Martini" Bologna
Diploma accademico di I° Livello in
Musica Elettronica

Lo sviluppo delle tecnologie digitali legate alla composizione ed esecuzione della musica elettronica

Utilizzo di SuperCollider in una performance live con composizione assistita

Candidato: Giovanni Onorato
Matricola n° 9333

Relatore: Damiano Meacci

Sessione Invernale a.a. 2018/2019

1. Premesse

L’obiettivo di questo lavoro è indagare circa l’utilizzo degli strumenti digitali in contesti musicali.

Vengono presi in esame alcuni esempi dalla storia della computer music. Questi sono stati scelti in base alla pertinenza con la parte progettuale e alla loro rilevanza nello sviluppo di questo ambito.

La parte progettuale prevede una performance eseguita dal musicista e dal calcolatore.

Il computer si occupa di generare aleatoriamente una struttura in base a dei parametri forniti dal programmatore, per poi eseguire gli eventi sonori definiti anch’essi in maniera randomica e sempre entro limiti prestabiliti.

Il musicista quindi, sulla stessa macchina ma su un altro supporto software, va ad interagire con la struttura generata attraverso l’elaborazione digitale del segnale proveniente da un basso elettrico.

I parametri forniti per la gestione della struttura forniscono delle possibili direzioni che sono state scelte a priori dal compositore. I movimenti possibili sono stati pensati per lasciare spazio al musicista e quindi considerando una possibilità di interazione: possono occupare un preciso spazio spettrale, possono essere più o meno densi in certe porzioni temporali, o possono avere un'identità ben precisa che li contraddistingue dagli strumenti utilizzati invece in live electronics.

In questo documento, vengono affrontati i passi più rilevanti per la realizzazione della performance.

2. Breve panoramica sui primi utilizzi del computer in contesti musicali

Nel primo capitolo di storia preso in considerazione, è necessario considerare il contesto economico e sociale in atto ed il risvolto nell'ambito delle tecnologie: dagli anni dell'immediato dopoguerra, in cui le nazioni vincitrici hanno vissuto tendenzialmente una supremazia sulla ricerca e l'innovazione in campo informatico; fino agli anni ottanta, in cui il processo di sviluppo sia sociale che tecnologico favorì, in maniera relativamente più paritaria, gran parte di quello che possiamo definire mondo occidentale.

Nei primi anni della storia delle tecnologie digitali, soltanto le più prestigiose istituzioni e le più facoltose aziende avevano l'accesso ai *mainframes*[1]. Questi potevano occupare più stanze, e la loro gestione richiedeva diverso personale. Nella composizione musicale, ciò si traduceva in tempi di attesa estremamente lunghi prima di poter ottenere un risultato sonoro delle istruzioni date al calcolatore: la capacità di calcolo delle prime macchine era di un solo bit alla volta (gli attuali computer sono in grado di processare 32 o 64 bit in parallelo).

Soltanto verso la metà degli anni settanta si arrivò alla comparsa delle prime *workstations*[2], il linguaggio C e quindi la diffusione dei sistemi UNIX, che rendevano possibile la portabilità del software attraverso diversi sistemi hardware.

2.1. Il CSIR Mark 1 in Australia ed il Ferranti Mark 1 in Inghilterra

Durante gli anni quaranta, vi erano diversi calcolatori elettronici ed elettromeccanici, come ad esempio *Colossus* o il primo *ENIAC*.

Nella metà del 1948 entra in funzione il *Manchester Small Scale Experimental Machine* quello che viene considerato essere il primo elaboratore elettronico *a programma*

[1] Secondo il dizionario Cambridge, la definizione di mainframe è "a very large, powerful computer with a lot of memory that many people can use at the same time".

[2] Internet Engineering Task Force nella Request For Comment 782[9], definisce *workstation environment* come hardware e software dedicati ad un singolo utente, al contrario dei mainframes i quali erano progettati principalmente per più utenti; segnano quindi un passaggio importante nel percorso della human-computer interaction.

memorizzato (stored-program machine), ovvero un calcolatore elettronico provvisto di memoria per dati e programmi.

Dall'invenzione di questo particolare tipo di macchina al primo utilizzo di un simile dispositivo a fini musicali, intercorsero circa due anni. Va sottolineato comunque, che le principali finalità di tutti i computers costruiti in questo periodo erano fortemente legate al calcolo: il Colossus sopracitato, era stato inventato per decriptare i messaggi dell'Asse, svolgendo sostanzialmente operazioni Booleane; lo scopo dell'ENIAC era invece inerente al calcolo balistico.

Nella fine degli anni quaranta il Consiglio Australiano per la Ricerca Scientifica e Industriale (CSIR) decide di costruire un computer interamente elettronico-digitale*:* il CSIR Mark 1 (ribattezzato CSIRAC nel 1955).

Un ricercatore di questo progetto Geoff Hill, matematico che aveva anche l'orecchio assoluto, si trovò a programmare il CSIR Mark 1 per suonare melodie verso la fine dell'anno 1950. La prima sequenza suonata fu una scala maggiore, mentre il primo brano ad essere riprodotto fu la melodia della *Colonel Bogey March*.

Il CSIRAC era dotato di un diffusore che serviva per ricevere le *hoot instructions,* ovvero dei suoni-segnale che permettevano al computer di comunicare agli operatori informazioni come la fine di una *routine* o il suo arresto improvviso.

Il suono veniva quindi generato mandando degli impulsi ad un intervallo di tempo regolare per generare una frequenza udibile. Di qui la possibilità di controllare l'altezza musicale di questo suono "ricco in armoniche" ad intervalli prestabiliti, non senza difficoltà.

Quasi parallelamente in Inghilterra, il fisico e ricercatore Christopher Stacey scrisse un programma per suonare melodie nell'Ottobre del 1951 per il Ferranti Mark 1. Anche qui il suono veniva generato sostanzialmente nella stessa maniera del CSIRAC, attraverso l'interfaccia per le hoot instructions.

La notevole differenza fra i due sistemi era nei componenti delle stesse macchine. Tra questi, il sistema di memoria del calcolatore australiano era una *Mercury delay line*, tipo di dispositivo utilizzato nei sistemi radar durante la Seconda Guerra Mondiale, attraverso la quale era possibile accedere ai dati in maniera sequenziale. Nel Ferranti Mark 1 troviamo invece il cosiddetto *Williams Tube* che utilizzava il fosforo del tubo a raggi catodici per immagazzinare i dati. Attraverso questo dispositivo era possibile accedere ai dati in *maniera*

randomica (Randomic Access Memory), con cui venivano facilitate le operazioni inerenti al *timing*.

In entrambi i sistemi, il controllo delle altezze ad intervalli di tempo prestabiliti era sicuramente una grande sfida ed un faticoso esercizio di programmazione: non era ancora presente un vero e proprio modulo di conversione digitale-analogica (DAC), per cui la gestione della sintesi si trovava ancora ad uno stato primordiale.

Ad ogni modo, le finalità musicali erano senza dubbio superficiali, la riproduzione di melodie, anche se in *real-time*, era considerata come un gioco; non erano ancora chiare le possibilità compositive che l'elaboratore numerico avrebbe poi apportato alla ricerca musicale con l'arrivo dei musicisti nelle sale dei computers (Doornbusch, 2017).

2.2. MUSIC-N - I Bell Labs e il pioniere della computer music: Max Mathews

I Bell Laboratories sono stati e sono tutt'ora uno dei centri ricerca e sviluppo (Research & Development) più importanti nel panorama mondiale. Nati come Bell Telephone Laboratories, Inc., dipartimento dell'azienda AT&T, co-fondata proprio da Alexander Graham Bell.

L'azienda, soprattutto nei primi anni, ha dato un contributo enorme circa lo sviluppo tecnologico legato al suono. È qui nel 1957 infatti, che nasce il primo ambiente di programmazione (programming environment) per la sintesi del suono chiamato MUSIC e più tardi ribattezzato MUSIC-I, sistema funzionante sul computer IBM 740[3].

Max Mathews fu il padre di questo progetto ed il pioniere della computer music.

Ciò che ha reso possibile lo sviluppo delle tecnologie musicali digitali fu proprio l'utilizzo del convertitore digitale-analogico di cui sopra, componente che si occupa di trasformare blocchi di dati digitali (*samples*) in segnali analogici.

Alla realizzazione di MUSIC-I seguì velocemente il suo diretto erede MUSIC-II, e nel '59 MUSIC-III, progettato per il mainframe IBM 7094.

[3] Il primo computer ad implementare la *magnetic-core memory* al posto del *Williams tube*; il tipo di memoria *core-memory*, in uso fino agli anni '70

In quest'ultima versione si trovò per la prima volta il termine *unit generator*, in breve UGen. Le UGens erano sostanzialmente dei blocchi per generare o elaborare il suono. (Dodge, 1997) Nelle UGens potevano essere definiti input e output. Un esempio basilare di UGen potrebbe essere un oscillatore periodico a cui vengono fornite attraverso i suoi input, variazioni di frequenza e fase. Alcuni di questi segnali di input inoltre, viaggiavano ad una velocità diversa dei segnali udibili: la cosiddetta control-rate.

Le UGens erano quindi modulari: se a quest'ultima UGen veniva moltiplicata ad un'altra con funzione di envelope generator, il segnale periodico sarebbe stato inviluppato, e quindi definito nel tempo.

Nel connettere questi elementi assieme, si creava ciò che veniva definita una patch, come nel mondo dei sintetizzatori analogici, o uno strumento (instrument).

L'insieme degli strumenti formava un'orchestra, a questa vengono date informazioni circa la loro durata nel tempo attraverso uno spartito (score).

Seguirono MUSIC-IV e MUSIC IV-B, quest'ultimo creato nel 1963 fuori dai Bell Labs: i laboratori condivisero il codice della quarta versione con la Princeton University, in cui era presente un IBM 7094. Mathews inoltre, decise di donare una scatola con 3000 carte perforate, ovvero il programma MUSIC-IV, ed un biglietto con scritto "Good Luck!" ad un giovane appena laureato alla Stanford University: John Chowning.

Più avanti, questi riscrisse il programma insieme ai propri colleghi, e chiamarono il software MUSIC-10, dove ten stava per PDP-10, il computer per cui il programma era stato scritto. A corredo del nuovo software, venne implementato anche un programma chiamato SCORE, che permetteva di gestire gli eventi musicali in maniera più fluida rispetto ai precedenti.

Nel '68, era disponibile il primo vero e proprio linguaggio di programmazione general-purpose high-level: FORTRAN. Fu attraverso questo che venne scritto MUSIC-V, e che quindi diventa il primo software musicale portabile in ogni sistema che implementasse il linguaggio.

Alcuni diretti eredi di MUSIC-N al di fuori dei Bell Labs furono MUSIC-360, per il computer IBM 360, MUSIC-11, per il PDP-11, di Barry Vercoe al Massachusetts Institute of Technology (MIT) e più in là Cmusic e Csound, come vedremo più avanti. Ad ogni modo, la struttura alla base di questi ed altri programmi rimarrà invariata rispetto anche alle prime versioni di MUSIC-N.

I concetti chiave di questi software giocheranno un ruolo fondamentale nel design dei successivi programmi per la sintesi audio tutt'ora esistenti.

2.3. Pietro Grossi, il CNUCE, la musica algoritmica ed il sistema TAU2/TAUMUS

La storia di Pietro Grossi è decisamente una delle più affascinanti ed emblematiche circa lo sviluppo delle tecnologie musicali nello scorso secolo.

Nel 1936, appena diciannovenne, vinse il concorso per diventare primo violoncello nell'orchestra del Maggio Musicale Fiorentino. Divenne poi Maestro al Conservatorio di Firenze, dove continuò ad insegnare questo strumento per lungo tempo.

Durante la mezza età, egli intravide le possibilità musicali che le nuove tecnologie avrebbero potuto favorire nella composizione. La prima esperienza in tal senso fu la visita allo Studio di Fonologia RAI di Milano nel 1961.

Rimase talmente affascinato da questo che decise di creare in casa un proprio studio, denominato Studio di Fonologia Musicale di Firenze (S2FM) e operativo nel 1963. Successivamente, lo studio venne accolto e ampliato al Conservatorio di Firenze, in cui si tenne il primo corso di Musica Elettronica già nel 1965.

L'interesse di Grossi non si fermò nel dominio analogico e poco dopo egli comprese che la sua ricerca avrebbe dovuto migrare verso il mondo della *computer music*.

Fu così che il compositore riuscì a lavorare per la Olivetti General Electric dove incontrò Ferruccio Zulian, un giovane fisico dello staff che aveva impiegato un sistema simile al metodo delle hoot instructions per generare altezze determinate. Il sistema era stato sviluppato sul GE-115, computer della Olivetti che venne utilizzato persino in concerto alla Biennale di Venezia nel 1964. (Parolini, 2014)

La collaborazione tra il fisico ed il compositore cessò perché la compagnia decise di porre fine al progetto sulla computer music.

Grossi aveva però intanto allargato le sue conoscenze riguardo la computer music nel territorio nazionale e venne invitato a collaborare con il Centro Nazionale Universitario di Calcolo Elettronico (CNUCE) a Pisa. Il Centro venne spostato sotto la direzione del CNR, e,

tra le diverse collaborazioni, fu molto importante quella con il dipartimento R&D italiano dell'azienda IBM, il quale veniva ospitato proprio al secondo piano dello stesso edificio.

Il primo sistema generato da questa fertile collaborazione fu il Digital Computer Music Program (DCMP) tra il 1969 ed il 1970 basato su un sistema IBM 360/67. I software designati *Algor* e *Create* permettevano di creare strutture musicali aleatorie in tempo reale.

L'azienda successivamente, mise a disposizione una nuova macchina, l'IBM 1800, dotata di notevoli DAC. Fu così che la squadra di Grossi cominciò a lavorare su un altro software, PLAY 1800 tra il 1970 e il 1971; il programma era stato sviluppato per poter suonare musica in tempo reale potendo controllare timbro e volume in maniera più precisa rispetto ai sistemi precedenti.

Un'altra importante collaborazione del CNUCE era con l'Istituto dell'Elaborazione dell'Informazione (IEI), un ente pressoché analogo ma in cui venivano prodotti anche componenti hardware.

Così venne sviluppato nel 1975 il progetto TAU2, un sintetizzatore polifonico a 12 voci di maggiore qualità sonora rispetto anche ai convertitori dell'IBM 1800. In questo sistema il suono veniva prodotto nel dominio analogico, e questo era di notevole risparmio computazionale.

Negli anni successivi, Grossi ed i suoi colleghi lavorarono intensamente sul software TAUMUS, la cui applicazione era quella di poter controllare la sintesi del TAU2, creando quindi un potente sistema ibrido analogico-digitale.

La squadra riuscì nell'intento di rendere possibile la performance in real-time, l'interattività e l'automazione attraverso la combinazione di questi sistemi. Riuscirono inoltre nell'obiettivo di poter controllare la sintesi da remoto attraverso il software TELETAU.

L'operato di Pietro Grossi ha portato a notevoli passi avanti nel percorso dell'informatica musicale in Italia e nel mondo: il concerto sopracitato al teatro *La Fenice* rappresenta il primo caso in assoluto di musica algoritmica nel territorio nazionale (Giomi, 1996).

2.4. I sistemi 4N di Giuseppe Di Giugno e le successive ricerche all'IRCAM di Parigi

Giuseppe Di Giugno è un fisico nisseno laureatosi all'Università di Roma.
Lavorò a diversi progetti di ricerca sul campo energetico arrivando a collaborare persino col CERN.
L'incontro con Robert Moog e successivamente, quello con Luciano Berio, favorirono il suo interesse verso le applicazioni degli strumenti digitali in musica.
Dopo diversi contatti oltreoceano, tra cui con i Bell Labs, nel 1975 concluse il primo progetto del 4A, il capostipite della serie dei processori definiti 4N. Il sistema gli fece guadagnare un certo interesse da diverse istituzioni: presentò il progetto anche a Stanford e al MIT. Nell'anno successivo il fisico venne invitato a lavorare all'IRCAM sotto la direzione di Pierre Boulez.[8]
A Parigi, dopo un anno, porta a termine il processore 4B, in collaborazione con Hall Alles dei Bell Labs. Questo sistema disponeva già di notevoli capacità: 64 oscillatori FM; 128 envelope generators, e la possibilità di utilizzare 4 forme d'onda contemporaneamente, dato che gli oscillatori erano controllati dalla sintesi wavetable nel microcomputer LSI-11 (Alles, Di Giugno, 1977).
Il progetto del modello 4C partì già nel gennaio del sessantotto, e ultimato nel maggio dello stesso anno. Il processore 4C era stato scritto per il DPD-11, si trattava in sostanza, di una versione più efficiente e flessibile del suo predecessore.
Di Giugno dunque, era intenzionato a progettare un sistema "che non fosse soggetto al processo di rinnovamento della tecnologia digitale e nemmeno all'evoluzione del pensiero musicale". [8]
Il 4X venne realizzato nel 1981 ed utilizzato per diversi anni di lì in avanti. Attraverso questo processore era possibile la sintesi e l'elaborazione del suono in tempo reale attraverso una tecnologia modulare.
Il sistema subì diversi aggiornamenti ed ampliamenti ad opera dello staff dell'istituto. Tra questi vi era un giovane Miller Puckette.

Nel 1986 l'impianto 4X si interfacciava con un apparato di controllo basato sul sistema operativo UNIX su un computer Sun 2/13, collegato a un monitor e una unità di memoria di 160Mb; su questo insieme giravano diversi programmi.
4xy era uno di questi, e consisteva in un insieme di implementazioni del linguaggio in C designato per gestire funzioni chiamate *Actions*. L'idea era quella di "semplificare il codice necessario alla generazione di algoritmi compositivi e migliorare il controllo in real-time di questi" (Favreau, 1986).
MAX era un altro componente software dell'intero impianto. Il nome era dedicato proprio a Max Mathews, il programma era una prima versione del celebre Max/MSP, come vedremo più avanti.
Le applicazioni di questo sistema potevano spaziare dalla sintesi FM al controllo del Live-Electronics; era definito il sintetizzatore/processore audio più potente al mondo nell'anno 1984 (Puckette, 2002).

2.5. John Chowning e la sintesi FM: la diffusione di massa dei dispositivi digitali

L'utilizzo di *Amplitude Modulation* e *Ring Modulation* ebbe un ampio riscontro fin dagli esordi dell'era della sintesi additiva: nel dominio analogico inizialmente ed in quello digitale poi.
La modulazione di frequenza (FM) invece, era già stata affrontata ampiamente all'inizio del secolo scorso ed era applicata alla trasmissione dei segnali radio nell'ordine dei MHz (C. Roads, 1996). Il primo ad occuparsi di questo fenomeno nel dominio musicale, e quindi udibile, fu proprio Chowning, altro pioniere della computer music: i suoi studi cominciarono tra il 1967 ed il 1968 a Stanford.
Egli stava ricercando circa la possibilità di ottenere un suono ricco nello spettro che evolvesse nel tempo, come avviene nei fenomeni naturali. Nel dominio della sintesi additiva però, sarebbero stati necessari più oscillatori con più modulazioni in ampiezza, richiedendo quindi troppo sforzo computazionale.
Quando Chowning si trovò a sperimentare con la modulazione di frequenza però, si rese conto che con soli due oscillatori, avrebbe potuto raggiungere un risultato abbastanza simile a

quello che avrebbe ottenuto impiegando la somma di una cinquantina di oscillatori sinusoidali. (Chowning, 1973)

È interessante notare che Chowning utilizzò il software MUSIC-V per la sua ricerca e la relativa divulgazione.

Brevettata la sua scoperta, nel 1973 la Yamaha riesce ad ottenere i permessi per applicare la sintesi nei suoi prodotti. Di qui il primo sintetizzatore digitale GSI, e nel 1983 il celebre DX7 il quale, per via del suo prezzo accessibile, ha segnato un passo fondamentale nella diffusione di massa dei sistemi musicali digitali.

Le applicazioni di lì in avanti furono ampiamente esplorate, alcuni dei tipi di sintesi derivati dalla FM saranno: *exponential FM*, *multiple-carrier FM*, *multiple-modulator FM*, *feedback FM* (C. Roads, 1996).

3. Dagli istituti di ricerca alla diffusione di massa

Nel 1969, Ken Thompson, giovane programmatore e ricercatore ai Bell Labs, inizia a programmare un gioco chiamato Space Travel, il cui fine era quello di mostrare all'utente una simulazione del sistema solare e osservare il paesaggio di vari pianeti e i loro satelliti. Questo gioco diverrà il primo ad essere portabile in diversi computer, una delle prime applicazioni UNIX. [7]

Thompson infatti, scrisse il programma per Multics, un sistema abbastanza macchinoso e complesso che veniva eseguito su una macchina altrettanto obsoleta a quei tempi: il PDP-7.

Quando i Bell Labs si ritirarono dal progetto Multics nel 1969, Thompson ed il suo gruppo (composto da D. M. Ritchie, M. D. McIlroy, J. F. Ossanna) decisero di creare un proprio sistema operativo chiamato UNIX, sistema scritto ancora per il PDP-7 [6]

Il sistema era stato scritto in linguaggio assembler (assembly language), e questo rendeva molto pedante e macchinoso il lavoro sullo stesso. Thompson decise allora di provare a scrivere il sistema su FORTRAN. Il tentativo fu paradossalmente vano: egli riuscì invece a scrivere un linguaggio, e il suo relativo compilatore, chiamato B perché fortemente influenzato da un linguaggio preesistente, il BCPL - Basic Combined Programming Language -. (Ritchie, 1984)

A causa di diverse inefficienze del linguaggio B, Thompson, Ritchie e il loro gruppo di ricerca cominciarono a sviluppare un altro linguaggio: C.
Dalle possibilità raggiunte attraverso il nuovo linguaggio venne riscritto UNIX: fu così che il primo sistema operativo portabile in altri sistemi hardware prese vita.
UNIX venne sviluppato fino alla fine degli anni ottanta e utilizzato ampiamente sia in ambito di mainframes sia nelle workstation. è possibile affermare che quasi tutti i concetti chiave di questo sistema operativo sono tutt'oggi alla base dei sistemi moderni.
Con lo sviluppo del sistema UNIX, del linguaggio C e la circolazione delle prime workstations assistiamo al primo passo verso diffusione di massa dei dispositivi digitali ed il radicale cambio di paradigmi nel design dei software e nei computer stessi.
L'utilizzo del calcolatore infatti, diventa sempre di più legato all'utente singolo, in contrapposizione alla gestione multi-user, ovvero il controllo simultaneo del mainframe da parte di più operatori.
Ad oggi siamo testimoni di un periodo storico in cui la maggior parte delle persone nel mondo ha accesso ad un device digitale. Esulano dall'obiettivo di questa tesi maggiori considerazioni circa i risvolti informatici, filosofici e sociali di questa transizione.
Vengono presi in esame adesso alcuni dei softwares che hanno più influito nello sviluppo degli ambienti di programmazione per l'audio.

3.1. Csound

Lo stesso Barry Vercoe che aveva già realizzato MUSIC-360 nel 1969 e contribuito alla realizzazione dell'antecedente 4-B, nel 1971 ottenne un impiego al MIT. Qui si trovò davanti ad un nuovo device: il DPD-11 e per questo computer, nel 1973 sviluppò Music-11 sulla base della precedente versione.
Tra le innovazioni apportate con questa riscrittura, una fu la più rivoluzionaria: il *k-rate*. Se già dalle prime versioni alcuni parametri (*filter motion*, *amplitude motion*, etc.) potevano essere controllati con la sopracitata control-rate, in questa versione, ogni UGen poté diventare un segnale di controllo atto a modulare altri parametri. Ottenendo un gran risparmio computazionale, dato che la k-rate viaggia ad una minore frequenza di campionamento.

Nel 1985, Vercoe tornò al MIT. Avendo a disposizione macchine più prestanti e dato che MUSIC-11 e Synthetic Performer[4] erano già scritti nel linguaggio C, nell'autunno dello stesso anno, realizza la prima versione di Csound con l'intento di creare un software per piattaforme UNIX, migliorare le prestazioni ed ampliare le possibilità dei precedenti software. (Vercoe, 2000)

Ad oggi, Csound è un programma utilizzato in diversi contesti, maggiormente per la computer music ed il sound design. In Csound, i blocchi chiamati UGens diventano *opcodes* (o talvolta *functions*), nella versione attuale 6.13.0, gli opcodes sono quasi 1700. Uno dei pregi di Csound è il suo range di applicazione: oltre ai sistemi operativi Linux, Windows e Mac, è utilizzabile persino su Android.[2]

3.2. "The MAX paradigm"

"The MAX paradigm" è il termine a cui si riferisce Miller Puckette nel suo articolo (Puckette, 2002). Attraverso questa definizione vengono messi insieme diversi programmi che rispondono allo stesso paradigma, tra cui *Max*, software commerciale distribuito dall'azienda Cycling '74, e *puredata*, gratis e open-source.

Secondo lo stesso autore, le idee alla base di Max confluirono da diversi contesti informatico-musicali tra cui: il programma RTSKED di Max Mathews; l'influenza di B. Vercoe ed il software MUSIC500; l'ambiente del MIT Experimental Music Studio nei primi anni ottanta; il software prese forma proprio all'IRCAM, trovando impiego nel sistema 4X, come menzionato sopra. (Puckette, 2002)

> The Max paradigm can be described as a way of combining pre-designed building blocks into configurations useful for real-time computer music performance. This includes a protocol for scheduling control and audio sample computations, an approach to modularization and component intercommunication, and a graphical representation and editor for patches. (M. Puckette, 2002).

[4] *Synthetic Performer* scritto da B. Vercoe con l'aiuto di M. Puckette, fu un programma pensato per rendere possibile l'esecuzione live di un brano strumentale assieme ad uno più esecutori umani. L'algoritmo inquadrava tre momenti: *listen*, ovvero l'analisi delle informazioni in arrivo dall'esecutore; *perform*, e quindi sincronizzare congrui eventi musicali; *learn*, cioè elaborare ed applicare, in base statistica, i dati delle precedenti esecuzioni. Il programma era stato scritto per funzionare con un processore 4X all'IRCAM (B. Vercoe 1984; B. Vercoe, M. Puckette, 1985). Un esempio video è mostrato in sitografia[10]

Come descritto dalle parole dell'autore, quando parliamo del paradigma Max, ci riferiamo ad un approccio alla programmazione unico nel suo genere, e che ne ha sicuramente facilitato l'accesso a tanti musicisti.

Un approccio che prevede dunque dei blocchi prestabiliti, questi però non sono le stesse entità degli oggetti nella Object Oriented Programming, come vedremo nel paragrafo successivo, sarebbe forse corretto quindi parlare di *Object-Based Programming*[5]. Gli oggetti di Max infatti non prevedono i concetti di *ereditarietà* e *polimorfismo* che verranno esplicitati nella sezione seguente.

Una delle peculiarità di Max è quella di rendere ancora più alto il livello di astrazione, e facilitare l'utilizzo del software attraverso una interfaccia grafica. In questo ambiente grafico, gli oggetti sono dei moduli rettangolari provvisti di determinati *inlets* e *outlets* a cui collegare dei cavi (*patch cords*), in maniera analoga al dominio dei sintetizzatori analogici.

Nel design di Max l'intento è stato quello di favorire la semplicità e l'esplicitazione, al fine di rendere il workflow più trasparente possibile, sebbene "trasparenza opposta all'intelligenza".(Puckette 2002).

> [...]Further, Max lacks any notion of linear "control flow" such as is fundamental in any real-world programming environment. The whole notion of control flow, with loops, conditionals, and subroutines, is easy to express in text languages, but thus far graphical programming languages haven't found the same fluency or economy of expression as text languages have. (Puckette, 2002)

Come riportato qui, ci sono degli aspetti critici riguardo l'utilizzo di Max, di fatto il sistema è orientato ai processi piuttosto che alla gestione dei dati.

Dunque, se da un lato questo ambiente può far avvicinare il musicista al dominio digitale, è pur vero che l'utente che abbia delle particolari necessità sarà costretto ad ovviare ad alcuni dei limiti che esistono dentro questo paradigma.

[5] Una definizione di object-based è riportata Chin e Chanson: "A programming language is defined as being object based if it supports objects as a language feature and object oriented if it also supports the concept of inheritance."(R.Chin, S..Chanson, 1991)

3.3. Object Oriented Programming

La *programmazione orientata all'oggetto*, d'ora in poi OOP, rappresenta uno dei passi fondamentali del processo legato alla diffusione di massa dei dispositivi digitali: la necessità di rendere un software di facile utilizzo per l'utente finale.

> Most ideas come from previous ideas. The sixties, particularly in the ARPA community, gave rise to a host of notions about "human-computer symbiosis" through interactive time-shared computers, graphics screens and pointing devices. [...] The soon-to-follow paradigm shift of modern personal computing, overlapping window interfaces, and object-oriented design came from seeing the work of the sixties as something more than a "better old thing." This is, more than a better way: to do mainframe computing; for end-users to invoke functionality; to make data structures more abstract. [13]

È necessario specificare che il linguaggio di programmazione rappresenti di per sé un modello astratto di computazione: sarebbe completamente inefficiente descrivere le operazioni computazionali in codice binario, ovvero come il computer gestisce le informazioni.

L'innovazione principale del paradigma OOP è quella di portare il linguaggio di programmazione ad un grado più alto di astrazione (in gergo: alto livello), e sebbene ciò accada in parte nei linguaggi strutturati come C, attraverso l'ambiente OOP l'utente è ulteriormente sollevato da alcuni dei compiti più tediosi e pedanti, necessari prima di allora nei linguaggi a basso livello.

Le principali caratteristiche dell'OOP sono i concetti di *incapsulamento, modularità, ereditarietà* e *polimorfismo.*

L'*oggetto* in sé, è una entità astratta definita a priori, la quale può assolvere diverse funzioni. L'oggetto risponde a dei messaggi, anche questi predefiniti, attraverso determinati *metodi*. In un ambiente di programmazione OOP, l'utente si trova davanti a una moltitudine di oggetti che possono essere manipolati e messi insieme in maniera diversa.

Tutti gli oggetti sono catalogati in delle classi, a seconda delle loro funzioni. Questa categorizzazione è in qualche modo analoga alle tassonomie naturalistiche: "Sarebbe inutile definire come proprietà di ogni mammifero l'allattamento dei pargoli. È invece utile definirla

in alto nell'albero tassonomico così che essa venga automaticamente ereditata da tutti i nodi inferiori."(Valle, 2015)
Gli oggetti quindi, rispondono a diversi metodi nell'ambito della classe di discendenza. Il concetto di polimorfismo è dato dal fatto che molti metodi abbiano lo stesso nome per diversi oggetti, ma rispondano a funzioni diverse, a seconda della classe di appartenenza.
I concetti alla base di questo paradigma sono stati molto influenti per lo sviluppo dei linguaggi di programmazione: esistono di linguaggi ad oggetto puri, come Smalltalk o Eiffel, e linguaggi ibridi come i più noti C++ e Java.

3.4. SuperCollider

Dopo aver abbandonato nel novanta il progetto di Synth-O-Matic, un software *C-like* per la sintesi audio in tempo differito, ed aver ultimato l'oggetto Max *Pyrite*, McCartney viene incoraggiato da C. Roads a riprendere il progetto del primo software, il quale lo condusse successivamente allo sviluppo di SuperCollider.
Il motivo che porta McCartney a sviluppare un altro linguaggio di programmazione, nonostante Csound e Max fossero già affermati nella computer music, è stato proprio quello di oltrepassare certe limitazioni presenti sia nella struttura di Csound, sia nella maniera in cui in Max e derivati gestiscono e trattano i propri oggetti. [1]
Ad oggi, riportando la definizione presente sul sito ufficiale, Supercollider è una piattaforma per la sintesi audio e la composizione algoritmica.
Ciò che si intende per piattaforma è il fatto che SuperCollider non sia un solo software, ma un agglomerato: *scsynth*, *sclang* e *scide*. Questa separazione sta alla base di uno dei punti di forza di SuperCollider: l'architettura *client*/*server*.
Il software scsynth è il server in tempo reale ed il nucleo di SuperCollider, contiene più di 400 UGens (lo stesso termine di Csound). Essendo il software open-source, ognuno può creare e aggiungere altre unit generators.
La componente sclang è invece proprio il linguaggio di programmazione OOP, e nell'architettura globale, rappresenta il client.
In SuperCollider ogni entità è un oggetto, il software è stato infatti sviluppato partendo proprio da Smalltalk. Ad eccezione di alcuni oggetti detti *primitives*, i quali sono scritti in

C++ per ragioni di efficienza, la quasi totalità degli oggetti di SuperCollider è scritta nel suo stesso linguaggio. (McCartney, 2002)

4. Il progetto

4.1. Alcuni aspetti tecnici

Nei prossimi paragrafi viene presa in considerazione la parte relativa al linguaggio di programmazione SuperCollider, e quindi la parte che genera in maniera aleatoria il suono con cui il musicista deve porsi in relazione. Per quanto concerne questa sezione, invece, verranno presi in esame alcuni dei procedimenti non legati a SuperCollider.

Il software che ho scelto per suonare in tempo reale con la struttura generata è Ableton Live 10, per via del fatto che ad oggi, è il programma più stabile e agile per l'esecuzione dal vivo.

L'unico problema che si interpone in questa fase è quello di far comunicare questi due programmi: in caso di avvio del server in SuperCollider, o dell'*Audio Engine* di Ableton, soltanto uno tra questi potrà avere accesso al driver audio scelto.

L'unica possibilità di ovviare a questa problematica rimanendo sulla stessa macchina è stata quella di impiegare il software *Jack*. Nativo di Linux ma portato anche in tutte le piattaforme più importanti, è una risorsa molto preziosa e utile anche se purtroppo, ad oggi non ha ancora raggiunto una piena stabilità, soprattutto sul sistema operativo Windows, ovvero quello utilizzato da me.

Attraverso Jack è possibile fare un *routing* del segnale tra diversi software se in questi è utilizzato come driver audio.

In tal modo, i due canali provenienti da SuperCollider vengono pilotati verso due ingressi di Ableton Live, e qui gestiti soltanto attraverso una *Group Track*.

In Ableton un ingresso è riservato al segnale proveniente dal basso elettrico. Questo ingresso viene prelevato da due tracce: una per la presa diretta vera e propria, e una per l'eventuale campionamento.

Entrambe le tracce vengono mandate interamente ad altre 3 tracce di ritorno: è qui che viene effettuata l'elaborazione del segnale proveniente dal basso elettrico attraverso diverse operazioni rese possibili dal programma stesso e alcuni device Max for Live.

4.2. Cenni su alcune scelte estetiche

Come accennato nelle premesse, il codice è stato progettato considerando che un musicista andrà ad interfacciarsi con questa struttura generata attraverso l'improvvisazione.
Questo si traduce in diverse strategie: anzitutto la disposizione dei suoni nel tempo. Il totale degli eventi possibili in una esecuzione va da un minimo di otto ad un massimo di dodici, considerando che la durata totale dell'esecuzione sia tra gli otto e i dieci minuti.
Questi eventi possono avere una durata compresa tra trenta secondi e due minuti, in realtà soltanto i materiali legati al paesaggio sonoro hanno una durata tendenzialmente maggiore per via della loro stessa natura. Inoltre, il fatto di estendere la durata di questo tipo di eventi permette di identificare il tipo e la presenza del materiale in questione, anche qualora questo dovesse presentarsi una sola volta in tutta la durata del brano.
La maniera in cui questi eventi si dispongono tra loro nel tempo ha a che fare con i *processi di accumulazione* di Salvatore Sciarrino: come verrà riportato dopo, c'è un array di tempi possibili che fa da perno, o come centro gravitazionale, attorno a cui vengono attratti tutti gli altri array contenenti le informazioni di partenza e fermata di ogni singolo evento. Risultando quindi in una aggregazione di elementi che viene dettata da regole temporali non circoscrivibili in misure come i BPM.
Un'altra fonte di ispirazione per quanto riguarda l'architettura degli eventi è stata la pratica del *Back to Silence* con Francesco Giomi, ed in particolare le sezioni da lui definite *Sintagmi*[6] all'interno delle partiture. La relazione con il concetto di Giomi è tuttavia presente soltanto prendendo in considerazione esclusivamente la parte auto-generativa: in relazione all'esecuzione infatti, il silenzio non è un aspetto artistico di rilevanza.
La scelta dei materiali tessiturali, ritmici e sintetici, ha a che fare con il mio orientamento artistico recente. Durante la mia permanenza a Stoccolma per un periodo Erasmus+ ai fini di studio, sotto la supervisione del Prof. William Brunson, ho cominciato un percorso compositivo partendo dalla registrazione dei suoni della città e dei suoi dintorni, il che è stato anche un piacevole pretesto per entrare in contatto con il luogo in cui mi trovavo.
L'idea di contrapporre in maniera così netta paesaggio reale e materiali sintetici viene da una registrazione di una piccola banchina nel centro di Stoccolma sull'isola di Skeppsholmen.

[6] Per entrambi i riferimenti è inserita una breve discografia alla fine di questo documento.

Nella registrazione è possibile udire la risonanza causata da un anello metallico ad un capo della corda ed il suo sfregamento con il perno a cui era attaccato. Ogniqualvolta le onde muovevano la barca dalla sua posizione originale veniva fuori questo suono quasi sinusoidale molto acuto.

Questo particolare suono ha attratto fortemente il mio interesse, sia perché in relazione al contesto arricchiva un altrimenti spoglio paesaggio di una banchina in centro città; sia perché se estraniato appena dal proprio contesto può realmente suonare come un glitch.

Partendo da questa registrazione ho composto il primo brano *microDocks*, e sempre su questa idea, un altro lavoro chiamato *tra*. Il primo brano è stato eseguito per ventinove altoparlanti nella *Lillasalen* della Royal Academy of Music di Stoccolma, diffuso con tecniche miste tra cui *Ambisonics*.

5. Utilizzo di SuperCollider nella performance

Circa l'utilizzo di SuperCollider come mezzo di espressione musicale, uno dei possibili ostacoli che può dover affrontare l'utente finale, ossia il programmatore-musicista, è proprio il fatto di dover leggere e scrivere, piuttosto che vedere e toccare. (Valle, 2015)

Nonostante ciò, le ragioni che protendono verso questa scelta sono molteplici, anche mettendo da parte le sue enormi capacità di cui sopra.

Una delle motivazioni che ha spinto me ad usare questo linguaggio è legata al concetto di *Affordances* di J. Gibson. Esula dal fine di questo documento un'analisi più dettagliata del lavoro dello psicologo, per cui verranno presi in esame soltanto alcuni dei concetti chiave.

La definizione che l'autore suggerisce circa il termine è: "the affordance of anything is a specific combination of the properties of its substance and its surfaces taken with reference to an animal" (Gibson, 1966).

Una delle traduzioni del termine affordance proposte sarebbe quindi una sorta di invito all'uso che l'oggetto propone al soggetto, attraversando quindi la separazione soggetto-oggetto, ed apportando una notevole prospettiva circa la definizione di percezione in relazione dell'ambiente circostante (environment) ed il suo rapporto percettivo-propriocettivo, sia nel dominio umano che in quello animale.

La ragione per cui viene presa in considerazione questa idea è che, a mio avviso, il mezzo espressivo gioca un ruolo chiave in termini compositivi ed esecutivi; dunque lo strumento permette al musicista un qualcosa di ben preciso, mentre ne scoraggia altri usi.
La questione non è sicuramente oscura al musicista: un violino permette di ottenere un suono sostenuto, un insieme di voci crea sempre un effetto *chorus*, e come questi, possono essere fatti molti altri esempi.
Il focus in questo caso è la ragione dell'utilizzo di SuperCollider piuttosto che software del tutto *user friendly* come Ableton, o persino ambienti di programmazione visuali, quindi di certo meno ostativi, come Max o puredata.
Rispetto all'utilizzo di programmi user-friendly, tra le diverse implicazioni, una è di notevole rilievo: se da un lato avere una griglia che scandisce il tempo in battiti per minuto invita il musicista all'uso del tempo in senso tradizionale, dall'altro può essere limitante circa l'utilizzo di tempistiche non convenzionali o non misurate in BPM. In SuperCollider invece, facendo click sull'icona l'utente si interfaccia con una pagina vuota.
Appare quindi ovvio che le affordances di un ambiente come SuperCollider diventano sicuramente meno prominenti ma molto più vaste.
Per quanto concerne l'utilizzo di piattaforme visuali, sono diverse le caratteristiche che facilitano l'approccio a diversi tipi di funzioni in SuperCollider rispetto a Max o puredata, tra queste ad esempio la gestione degli *array* e di conseguenza la *multichannel expansion*, come vedremo più avanti.

5.1. La struttura del progetto

Tra le caratteristiche che rendono SuperCollider una piattaforma particolarmente efficiente e flessibile, è la possibilità di organizzare il proprio flusso di lavoro in una maniera estremamente personale e definibile ad-hoc per ogni progetto.
La strutturazione del codice è senza dubbio uno dei compiti più importanti del programmatore/musicista. Per il mio progetto ho scelto di seguire una struttura suggerita in un tutorial di Eli Fieldsteel, professore di teoria e composizione e direttore degli Experimental Music Studios all'Università dell'Illinois. [11]

La prima azione da programmare è legata alla configurazione del server: questo non verrà avviato subito per motivazioni che vedremo a breve.

In questa fase è importante definire tutte le opzioni che dovremo assegnare al server in maniera tale che il sistema funzioni correttamente una volta che sarà avviato, tra le opzioni da definire vi sono: la pulizia di precedenti assegnazioni alle funzioni *Server Boot*, *Server Tree* e *Server Quit*; la scelta dei corretti driver audio per il sistema con cui il computer si interfaccerà ad ogni diversa performance; la scelta dell'adeguato valore di *block size,* ovvero la porzione di samples - e quindi temporale - che il sistema audio prevede prima della riproduzione della sintesi in tempo reale.

Il secondo passo da percorrere è quello di definire e inizializzare tutte le variabili globali, ovvero le variabili necessarie e richiamabili in tutte le sezioni del codice, nel mio caso queste sono: ~tempo, l'oggetto TempoClock stabilisce lo scheduling di certe azioni in base all'unità di tempo BPS, in questa fase viene soltanto stanziato per essere utilizzato più tardi come vedremo dopo; ~bufPath e ~scapePath, ovvero le stringhe che permettono di richiamare il percorso (in termini di cartelle) per arrivare ai campioni audio da caricare in degli appositi buffers; ~times, i diversi punti temporali da cui dipendono i diversi tipi di eventi; e ~scale, un array contenente diversi tipi di scale che verranno scelte in maniera casuale ogni volta che ogni diverso synth sarà istanziato.

Subito dopo la definizione delle variabili globali, diventa logico istanziare le diverse funzioni globali, ovvero funzioni che verranno richiamate più avanti nel codice. Tra queste abbiamo ~chrono, una funzione presa in prestito dal libro Introduzione a SuperCollider (Valle, 2015) e leggermente ritoccata che permette di mostrare una finestra esterna all'IDE contenente un cronometro. Questo è notevolmente utile ai fini dell esecuzione della performance, in aggiunta alla *post window* di SuperCollider in cui vengono restituiti con un anticipo di trenta secondi il tipo di eventi che seguiranno. Abbiamo poi ~arrFunc, una funzione che, restituisce un array contenente *n* valori; fornendole l'array dei valori di tempo ~times, un array contenente le probabilità di quali di questi debbano essere scelti (*weights*), e un valore che indica la quantità di valori che sarà presente nell'array restituito (*num*)[7].

[7] Grazie alla vasta e molto attiva comunità attorno a SuperCollider, questo codice mi è stato suggerito da James Harkins, compositore ed insegnante presso il Conservatorio di Guangzhou in Cina.

```
~arrFunc = { |array, weights, num|
    var result = Array(num), i;
    weights = weights.copy;
    min(weights.size, num).do {
       i = weights.windex;
       result = result.add(array[i]);
       weights = weights.put(i, 0).normalizeSum;
    };
    result
};
```

Grazie a questa funzione vengono dunque stabiliti i diversi punti di partenza di ogni singolo evento, in relazione al primo array ~times.

Le altre funzioni dichiarate in questa sezione sono ~makeBusses e ~makeNodes in cui vengono stanziati Bus, Gruppi e i Synth utilizzati come elaboratori di segnale.

Le funzioni ~makeRhytmBuffers, ~makeScapes sono quasi identiche, sono delle funzioni ricorsive che iterano nelle cartelle, grazie alla variabile ~path precedentemente dichiarata, per caricare in degli appositi buffers i campioni scelti ed inserirli in un proprio oggetto *Dictionary,* classe discendente da *Collection* (come l'oggetto Array) che si occupa di assegnare dei simboli a dei dati.

Abbiamo poi ~makeBuffers che carica in appositi buffers delle tabelle (nel formato wavetable specifico di SuperCollider) da utilizzare per la sintesi *wavetable*.

In questa terza sezione vengono anche determinate le funzioni da ~texture_1 a ~scapes che si occuperanno di instanziare i synth a tempo debito, come vedremo più avanti.

La quarta sezione del nostro codice è quella che si occupa di registrare quasi tutte queste funzioni nel programma di avvio e spegnimento del server, questa è una delle caratteristiche principali che rende l'utilizzo dei linguaggi OOP davvero pratico ed efficiente. Come vedremo, non tutte le funzioni globali sono ancora state stanziate nelle loro apposita destinazione.

A questo punto, è possibile definire i processi che necessitano il server avviato, ovvero la definizione dei synth: i SynthDef.

Come vedremo a breve, gli oggetti SynthDef sono il nucleo delle operazioni inerenti al server, e quindi della sintesi audio e la sua elaborazione.

Nel mio progetto faccio uso di dieci diversi SynthDef, tre dei quali assolvono il compito di generazione del suono, tre strettamente legati alla elaborazione del segnale, e quattro per la gestione dei canali, in maniera simile a come avverrebbe su un mixer. Stanziati questi, si arriva finalmente all'ultima funzione globale da istanziare.

Prima di fare ciò, bisogna sottolineare la differenza tra comandi sincroni ed asincroni in SuperCollider: alcuni ordini dell'interprete richiedono più tempo per essere completati rispetto ad altri, e sono perciò denominati asincroni. Alcuni esempi più emblematici sono il caricamento di un buffer o di un Synthdef: per poter utilizzare quel buffer o quel synth, il server richiede del tempo prima di poter soddisfare la richiesta.

Fortunatamente, nel linguaggio viene implementato il metodo sync dell'oggetto server, il quale si occupa proprio di creare un tempo di ritardo affinché il compito appena assegnato sia già completo. Questo tipo di comando, può esistere soltanto in alcuni tipi di *bundles*, ovvero dei pacchetti di comandi spediti al server, come il metodo waitForBoot, o bind che abbiamo appena visto.

Per far fronte a questo limite, l'istanziazione dei SynthDef è inserita in una routine waitForBoot, al termine della quale si trova un sync, cui segue il comando *ServerTree.add(~makeNodes)*.

È proprio qui che viene richiamata la funzione mancante: quella di creare sul server i nostri vari gruppi e synth, necessari al *routing* del segnale, e che non potrebbero esistere se il server non fosse già avviato. In più, il comando permette di assicurare che ogni volta che verranno liberati tutti i synth (ovvero quando viene pressata la combinazione di tasti Cmd + . o Ctrl + ., una sorta di *panic button*) allora anche l'*albero di nodi* venga richiamato.

A questo punto avvengono tutte le funzioni di scheduling che richiamano i synth e le funzioni modellati in base alla scelta artistica e che vengono trattate nel prossimo paragrafo.

5.2. Scheduling in SuperCollider

Se fino a ora si è parlato di organizzazione del codice, anche in relazione al tempo di inizializzazione, non abbiamo ancora affrontato la questione di come esattamente il suono venga organizzato nel tempo.

La traduzione più appropriata del termine scheduling in italiano sarebbe quella di programmazione (inerente agli eventi), per via però del possibile fraintendimento, il termine non verrà tradotto.

Per gestire la sequenziazione degli eventi in SuperCollider esistono diverse possibilità. Tra queste vi è un gruppo di oggetti simili e discendenti dallo stesso oggetto Clock: SystemClock, AppClock e TempoClock, quest'ultimo introdotto precedentemente.

Dato che l'organizzazione della struttura globale del mio progetto non ha molto a che fare con l'utilizzo di questa unità di misura, ho preferito utilizzare SystemClock che permette invece di utilizzare l'orologio più preciso a disposizione, ovvero quello interno alla macchina e nel dominio dei secondi.

Per ogni evento sonoro viene dunque definita una funzione che prevede due argomenti: *start* e *stop*. Subito dopo la dichiarazione o la scelta di alcune variabili, ci troviamo di fronte all'oggetto SystemClock.sched(*delta*, *item*). Il metodo sched richiede proprio come primo argomento il momento in cui far partire l'evento, e come secondo l'evento stesso.

Tutte le varie funzioni come ~rhythm funzionano pressoché in maniera identica per quanto riguarda lo scheduling.

Queste funzioni contengono principalmente altri oggetti SystemClock.sched, è interessante notare come lo stesso oggetto venga utilizzato sia per la routine di stanziamento dei synth, con le loro varie caratteristiche, sia per lo stop, semplicemente iterando il messaggio "\gate, 0" tra i vari array contenenti i diversi synth.

Subito dopo aver interpretato queste funzioni, il linguaggio si trova davanti un array contenente due variabili globali, ovvero altri due array, e questi vengono accoppiati attraverso il metodo flop. Ogni coppia di questi valori viene quindi utilizzata come argomento della funzione richiamata dal metodo do.

Con questo metodo dunque, le azioni da intraprendere vengono decise subito dopo che il server sia avviato, ma verranno eseguite soltanto al momento deciso a priori in maniera casuale dal sistema stesso, entro alcuni parametri prestabiliti

5.3. Le funzioni per la gestione degli eventi sonori

Finora sono state affrontate le funzioni attraverso le quali la macro-struttura viene organizzata.

Ogni singolo evento sonoro invece, viene gestito da delle funzioni ad-hoc precedentemente citate, e sono divise in base al loro tipo di materiale o alla loro maniera di evolversi nel tempo, queste sono: ~texutre_1; ~texutre_2; ~texutre_3; ~angryTexture; ~rhythm e ~scapes.

La prima, la terza e la quarta funzione sono abbastanza simili: viene scelta dal sistema una scala tra un array di scale possibili e viene formato un accordo in base ai gradi della scala, in maniera tale da ottenere un insieme di frequenze fondamentali più o meno coerenti.

Dopo ciò, viene riempito un array di sedici diversi synths, e per ognuno di questi, oltre ad una propria fondamentale, verrà scelto: un proprio inviluppo (dadsr, ovvero un tradizionale adsr con l'aggiunta di un tempo iniziale di delay); la posizione all'interno dei quattro buffer disponibili; una frequenza di taglio di un filtro passa-basso; la quantità di effetto detuning; la quantità di *jitter*, ovvero il movimento casuale della posizione all'interno dei buffer; la quantità di segnale da inviare ai bus degli effetti riverbero e delay.

```
synthi = Array.fill(16, {
    Synth.new(\vOsc, [
        \freq, chords.choose,
        \amp, rrand(0.01, 0.4),
        \dly, rrand(0.001, 15),
        \atk, if(rrand(0, 0.0005).coin, 0, rrand(15.0, 19)),
        \dcy, rrand(4.5, 12),
        \sus, rrand(0.2, 0.6),
        \rel, if(rrand(0, 0.1).coin, 0, rrand(10., 35)),
        \crv, rrand(-9, 6.5),
        \pos, rrand(0.5, 2.7),
        \cutoff, rrand(6000, 15000),
        \detuneAmount, rrand(0.001, 0.55),
        \jitAmount, rrand(0.001, 0.2),
        \buf, ~bufSin.bufnum,
        \out, ~textBus
        \rev, rrand(0, 0.99),
        \revOut, ~revBus,
```

```
        \del, rrand(0, 0.2),
        \delOut, ~delBus
    ], ~srcGrp
)});
```

L'unica differenza fra ~texture_1 e ~texture_3 sta nel fatto che in quest'ultima funzione, vengono prima stanziati 4 filtri passa-banda ad una frequenza di taglio scelta aleatoriamente, e quindi i quattro synths che generano la tessitura vengono pilotati sul bus su cui i passa-banda elaborano il suono.
Per quanto riguarda invece la funzione ~texture_2, ho deciso di utilizzare un tipo di oggetti molto interessante del linguaggio SuperCollider: i Patterns.
Questi sono uno dei punti di forza di questo linguaggio, permettono una articolata e dettagliata programmazione temporale degli eventi e, come nel mio caso, possono permettere di scoprire altre affordances dello stesso SynthDef.
La funzione ~rhythm è basata anche su quattro patterns che gestiscono quattro tipi di campioni diversi: *kick*, *sines*, *snares* e *hihat*.
Per questa funzione ho scelto di utilizzare un TempoClock, impostato arbitrariamente a 149, il quale facilita la gestione degli eventi ritmici. Anche qui un certo grado di imprevedibilità è presente, il che rende ogni evento di questo tipo più stabile o instabile a seconda della scelta pseudo-randomica del sistema.
L'ultima di queste funzioni, ~scapes, è quella che gestisce la riproduzione di alcuni campioni di ambienti reali. Sostanzialmente, il funzionamento non è tanto differente rispetto ai primi eventi tessiturali.

5.4. SynthDefs

Come accennato sopra, il nucleo dell'elaborazione digitale del segnale (DSP) in SuperCollider, è proprio il SynthDef, in cui vengono esplicitate le funzioni, e definite le UGens, di ogni futuro synth stanziato.
I SynthDefs nel mio codice sono: vOsc; rhythmBuf; fieldBuf; rev; delay; BPF; textMix; bufMix; scapeMix, master.

Il primo, vOsc, è quello che si occupa della sintesi wavetable, attraverso l'oggetto VOsc: l'oscillatore che legge un buffer (in formato wavetable[8]), e sottoclasse di PureUgen.

```
SynthDef.new(\vOsc, {
    arg buf=0, numBufs=4, pos= 1.0,
    freq = 300, amp = 0.9, detuneAmount = 0.001,
    jitAmount=0.001, cutoff = 12000,
    atk=0.001, dcy=0.2, sus = 1.0, rel = 0.4, dly=0.0, crv=1,
    bpf= 0, bpfout = 1, gate = 1, out=0;
    var sig, env, bufpos, jitter, detuneSig;
    jitter = LFNoise0.kr(jitAmount, jitAmount, jitAmount);
    bufpos = buf + pos + jitter;
    detuneSig =
    LFNoise1.kr(0.02!8).bipolar(detuneAmount).midiratio.lag(0.5);
    env = EnvGen.kr(Env.dadsr(dly, atk, dcy, sus, rel, curve:crv), gate,
    doneAction:2);
    sig = VOsc.ar(bufpos.clip(buf+0.1, numBufs-0.1),
    freq*(detuneSig).lag(0.3));
    sig = Splay.ar(sig) * env * amp;
    sig = LPF.ar(sig, cutoff);
    Out.ar(out, sig);
    Out.ar(bpfout, sig * bpf);
    Out.ar(delout, sig * del);
}).add;
```

Nel codice, oltre la dichiarazione degli argomenti e della variabili, troviamo subito il segnale jitter: un LFNoise0 (un tipo di segnale di controllo che genera valori randomici) che ripete il suo argomento jitAmount di cui sopra, nei suoi ingressi di frequenza, mul e add (banalmente, *jitAmount, +jitAmount). Questo segnale verrà moltiplicato alla posizione del buffer per fare in maniera tale che, su richiesta, il punto di lettura cambi continuamente, risultando in uno spettro continuamente variabile e più organico.
Troviamo poi la variabile detuneSig, si tratta sempre di un generatore di valori randomici, ma al suo argomento troviamo un tipo di scrittura particolare: "!8".

[8]Un tipo di formato che rende più efficiente la lettura della tabella; nella documentazione in SuperCollider: "Signal: [a0, a1, a2...]; Wavetable: [2*a0-a1, a1-a0, 2*a1-a2, a2-a1, 2*a2-a3, a3-a2...]. This strange format is not a standard linear interpolation (integer + frac), but for (integer part -1) and (1+frac)) due to some efficient maths for integer to float conversion in the underlying C code."

Attraverso il punto esclamativo, stiamo dicendo al server di creare 8 istanze dello stesso segnale, ognuna con i propri valori randomici. Il punto esclamativo è una delle abbreviazioni in SuperCollider che prende il nome di *sugar syntax*.
Questo esempio di moltiplicazione di segnali concerne proprio la multichannel expansion citata ad inizio del paragrafo. Sono davvero tante le applicazioni e tanto affascinanti i risvolti di questo metodo.
Le regole di questa espansione possono però creare confusione ad un primo sguardo: se moltiplico la stessa variabile detuneSig, per una istanza contenente soltanto una UGen VOsc, otterrò otto istanze dello stesso VOsc che a sua volta, moltiplica 8 istanze con ognuna diversi valori. Ottenendo quindi, in uscita 8 oscillatori con la stessa posizione di buffer, ma con differente frequenza (dato che l'argomento freq moltiplica la variabile detuneSig).
Troviamo poi due SynthDef pressoché identici, rhythmBuf, fieldBuf:

```
SynthDef.new(\rhythmBuf, {
    arg atk=0.00001, sus=1, rel=0.2,
    buf=0, rate=1,
    amp=1, out=0, pan=0;
    var sig, env;
    env = EnvGen.kr(Env([0,1,1,0],[atk,sus,rel],), doneAction:2);
    sig = PlayBuf.ar(1, buf, rate*BufRateScale.ir(buf));
    sig = sig*env*amp;
    sig = Pan2.ar(sig, pan);
    Out.ar(out,sig);
    Out.ar(delout, sig * del);
}).add;
```

Il nucleo di questo blocco è l'oggetto PlayBuf che, come è facile immaginare si occupa di riprodurre un buffer caricato precedentemente. L'efficienza di questo blocco sta nel fatto che con un solo SynthDef, possiamo gestire contemporaneamente tutti i tipi di buffer ritmici predefiniti nel dizionario (un oggetto *Dictionary* nella funzione ~makeBuffers).
Vengono poi definiti i SynthDef relativi all'elaborazione del segnale. Il primo di questi è rev, in cui il nucleo di base è l'oggetto FreeVerb di SC.

Il secondo è il tipo di delay che verrà miscelato in fase di mix del segnale, ogni SynthDef di generazione del suono prevede più uscite in maniera tale che il segnale venga prelevato randomicamente per ogni synth, sempre entro certi parametri stabiliti.

```
SynthDef(\delay, {
    arg in, out;
    var sig, del;
    sig = In.ar(in, 2);
    sig = sig - OnePole.ar(sig, exp(-2pi * (120 * SampleDur.ir)));
    sig = OnePole.ar(sig, exp(-2pi * (14000 * SampleDur.ir)));
    del = CombN.ar(
       sig,
       4.0,
       LFNoise1.kr(0.1, 0.5, 1.5) * 2.5, //deltime between 1 and 2.5
       LFNoise1.kr(0.1, 0.5, 0.5) * 8//max 8 sec decayTime
    )!8;
    del = Splay.ar(del);
    Out.ar(out, del)
}).add;
```

Il nucleo di questo SynthDef è l'oggetto CombN[9], ovvero un tipo di comb filtering che prevede la reiterazione nel processo, cioè il feedback. Essendo di per sé il comb filtering una operazione che ha a che fare col ritardo del segnale e per via della grandezza delle linee di ritardo, questo oggetto viene utilizzato più come un tradizionale effetto eco con feedback piuttosto che un vero e proprio comb filtering.
È da sottolineare che anche in questo blocco viene utilizzato in maniera creativa la multichannel expansion e la randomicità di ogni istanza dei synth creati: il tempo di ritardo e di decay è infatti gestito da due oggetti LFNoise1, ovvero un generatore di valori randomici che abbiamo visto precedentemente.
Gli otto segnali vengono poi miscelati in un fronte stereo con Splay, in maniera del tutto analoga alle precedenti sezioni.

[9] Esistono diversi oggetti per il comb filtering: CombC, CombL ed altri ancora. La differenza fra questi, come avviene in diversi tipi di oggetti, sta nell'interpolazione dei segnali. CombN è il tipo di oggetto senza interpolazione e quindi di minore carico computazionale.

Il blocco successivo è quello denominato BPF che sta per BandPass Filter. L'oggetto chiave qui, prende lo stesso nome. Anche per questo è prevista una componente di jitter, ovvero alterazione della sua frequenza di taglio.

Nel SynthDef è specificato un inviluppo al fine di utilizzare la propria funzione *doneAction*, di qui la possibilità di eliminare dal server i synth stanziati attraverso il messaggio "0" all'argomento "gate".

I tre SynthDefs definiti successivamente sono anch'essi analoghi, e servono soltanto per un corretto routing del segnale e per bilanciarne i volumi. L'unica differenza tra questi occorre nel canale che gestisce i diversi eventi ritmici, in questo viene utilizzato un compressore.

```
SynthDef.new(\bufMix, {
    arg in=0, out=0, vol;
    var sig, comp;
    sig = In.ar(in, 2) * vol;
    comp = Compander.ar(sig, sig,
       thresh: 0.4,
       slopeBelow: 1,
       slopeAbove: 1/6,
       clampTime:  0.01,
       relaxTime:  0.03
    );
    sig = XFade2.ar(sig, comp, -0.5, 1);
    Out.ar(out, sig);
}).add;
```

Nonostante l'insolita maniera di descrivere i parametri della compressione, Compander è un oggetto particolarmente utile in quanto è possibile rendere lo stesso oggetto in un compressore, un limiter, un expander o un gate, essendo queste delle operazioni analoghe fra loro.

L'ultimo SynthDef definito come master, è simile ai tre precedenti, con l'aggiunta di due oggetti: l'oggetto Limiter, per prevenire il clipping del segnale oltre i 0 dBFS; e l'oggetto LeakDC, il quale previene il problema del DC Offset.

Questo potrebbe intercorrere per via del fatto che il quarto buffer tra quelli utilizzati per la wavetable, è generato randomicamente, per cui potrebbe risultarne una forma d'onda unipolare che potrebbe creare problemi in fase di riproduzione negli altoparlanti.

6. Conclusioni

Anche se il focus delle esperienze storiche riportate è stato inevitabilmente condensato e centrato sugli ambienti di programmazione, la panoramica generale sull'evoluzione della computer music ha messo in luce la pluralità degli approcci ad un campo relativamente ristretto come i linguaggi inerenti alla sintesi audio nel dominio digitale.

Questa pluralità permette al musicista di oggi una vasta scelta tra i mezzi a sua disposizione affinché ogni parte progettuale possa assolvere determinati compiti: nel mio progetto di fatto, interagiscono tre componenti diverse: il linguaggio ed il software SuperCollider, il software Ableton Live, ed il paradigma Max di cui sopra, anche se in maniera indiretta.

Dunque, se il primo software è capace di permettere un flusso di lavoro del tutto personalizzabile e adattabile ad-hoc, è anche vero che per una esecuzione live è molto agevole utilizzare l'impianto di Ableton Live. Inoltre, il paradigma Max, come abbiamo visto, è stata una grande rivoluzione che permette al musicista - e quindi non un soggetto che ha solitamente a che fare con la programmazione - di poter creare i propri devices specifici per la sua espressione anche all'interno di Ableton.

Considero questo progetto come un inizio di un percorso artistico che continuerà nel mio percorso di studi successivo: ho proposto a diverse istituzioni in Europa la mia ricerca intitolata "Exploring the boundaries between Improvisation and Composition in the context of Electroacoustic Music".

L'obiettivo è quello di raggiungere una strategia compositiva che sia particolarmente elastica, in maniera tale da rendere l'esecuzione flessibile e adattabile in ogni contesto; rendendo comunque possibile definire una composizione come tale e quindi apprezzarne le peculiarità che ogni spazio può apportare all'esecuzione.

Credo fermamente che ogni differenza possa essere vista come un vantaggio, e non come un impedimento. Questa è una lezione che ho imparato attraverso la pratica

dell'improvvisazione, in contrapposizione all'idea legata ai lavori tradizionali per supporto fissato in cui lo spazio ostile può diventare un ostacolo alla corretta fruizione della composizione musicale.

L'impianto logico della mia futura ricerca è vicino all'idea di questo progetto ma soltanto in maniera parziale: nel mio progetto infatti è possibile udire delle somiglianze, talvolta anche strette, attraverso le varie esecuzioni sebbene ogni esecuzione concernerà diversi assemblaggi differenti del materiale prestabilito, e quindi della parte improvvisata.

Le somiglianze tra le varie esecuzioni hanno a che fare con alcuni aspetti compositivi. Tra i più evidenti vi è l'impianto concettuale: il fatto che all'interno della parte auto-generativa ci sia già una contrapposizione netta tra le identità dei materiali sintetici e quelli provenienti dal mondo reale. Questo contrasto viene ulteriormente accentuato attraverso l'esecuzione dal vivo del basso elettrico, di qui la sua dimensione propriamente strumentale, e la sua elaborazione nel dominio digitale.

Altri aspetti compositivi di rilevanza sono legati alla porzione di spettro occupata e l'evolversi dei suoni nel tempo: è possibile talvolta che il field recording della metro di Stoccolma occupi le stesse bande della massa generata dalla funzione della texture sintetica, o che talvolta la parte improvvisativa si contrapponga alla parte auto-generativa attraverso l'occupazione di bande spettrali opposte, e come questi molti altri aspetti.

Un ulteriore aspetto importante che lega il mio progetto futuro a questo progetto è il legame tra la composizione e l'esecuzione tra cui intercorre necessariamente lo strumento.

Dall'inizio della pratica esecutiva in Conservatorio ho sempre dato una certa importanza alla possibilità di interfacciarsi con uno strumento che di per sé sia capace di dare una risposta sonora immediata al musicista. Per questa ragione, all'inizio del mio lavoro sulla tesi ho sempre avuto in mente l'idea di includere uno strumento nella parte progettuale ed esecutiva, un obiettivo che ha a che fare anche col mio prossimo progetto.

Per cui, al percorso verso la strategia compositiva di cui sopra, seguirebbe parallelamente lo sviluppo di uno strumento designato ad-hoc per le mie necessità compositive ed esecutive.

L'elaborazione del proprio strumento nel contesto della musica elettroacustica, dal mio punto di vista, è inevitabilmente legato all'approfondimento dello sviluppo delle tecnologie, e spero che il mio interesse in quest'ambito sia emerso da questo testo.

L'approfondimento delle tecnologie digitali nel dominio sonoro è l'obiettivo del mio prossimo tirocinio post-laurea al Notam ad Oslo che dovrebbe cominciare in Maggio 2020.

7. Ringraziamenti

Anzitutto, vorrei ringraziare la mia famiglia che mi ha sempre supportato nelle mie decisioni, anche quelle più terribili, e che mi permette tutt'ora di inseguire quelli che sono i miei progetti e le mie ambizioni: sentitamente grazie Gino, Marinella, Francesca e le nonne, io non so come siate riusciti a sopportarmi finora. Vorrei inoltre ringraziare tutti i miei amici di Palermo. Anzitutto grazie Giulio, senza cui il mio percorso non sarebbe mai cominciato, grazie Vincenzo, Carlo, Alessandro, Roberto, Fazio e gli amici del box, i Bellanca e tutti quelli che ogni tanto mi chiedono "come stai?".

Un doveroso e sentito ringraziamento va al nostro corpo docenti, e in particolare al mio relatore Damiano Meacci e ai nostri maestri Francesco Giomi e Lelio Camilleri: è stata (e spero sarà) una risorsa indescrivibilmente positiva quella di avere un insieme coeso come guida per noi studenti. Un aspetto che credo di poter affermare si sia riflettuto su di noi colleghi: di qui la nascita del Collettivo della Scuola di Musica Elettronica.

Per cui il prossimo ringraziamento va proprio a loro: tutti i miei colleghi e colleghe che ogni giorno prendono parte alle assurde follie di queste brave persone ma con ambizioni un po' bislacche: grazie Agnese, Pipia e il collettivo Minus tutto, Nicola Maria Nik Vez Venturo, Giada, Mittino, Vogli... Grazie a voi tutti picciotti e picciotte (non me ne abbiate se non ho scritto tutti i nomi, ma mi toccava scrivere una pagina intera se no!). Un altro ringraziamento importante è quello per i ragazzi del Senza Distinzione Di Genere lab che sostanzialmente molti sono gli stessi, per cui, praticamente in aggiunta al periodo precedente: grazie ancora a voi, ma soprattutto grazie Nickies, senza di te tutto ciò non sarebbe mai esistito.

Altro doveroso ringraziamento va all'istituto Kungliga Musikhögskolan i Stockholm e il corpo docenti del dipartimento di composizione elettroacustica: *thanks to Bill Brunson, hope you are getting well: it is possible we will see each other again soon! Thanks to Mattias Petersson who inspired me to use SuperCollider which eventually ended up as being one of my main focus on my Bachelor Thesis. A great thanks to Giuseppe Pisano i met there, it is more than a treasure to meet someone like you. And thanks to all the colleagues the people I met in Stockholm, it was a really great experience! Thanks Emile, Luca, Diego, Mikael, and thanks to Alice and her family.*

Another important thanks goes to the SuperCollider Community which helped me a lot for the developing of this project, a particular thanks goes to Eli Fieldsteel for his precious tutorials, and James H. Harkins, teacher in Guangzhou, China and one of the most active user in the SC community: really thanks James, in one of the darkest times of my project, you do not really know how much you helped me with your advice!

Un ultimo ringraziamento necessario e particolarmente sentito va a tutte le persone che hanno gravitato attorno all'interno numero uno di Piazza Aldrovandi 19, ovviamente il ringraziamento più importante è quello da rivolgere ai coinquilini passati e presenti, per aver subìto e sopportato tutti i miei deliri (più e meno alcolici). Grazie Galvagno, senza cui questa casa non sarebbe stata quello che è stata (e spero sarà), grazie Michele, Maccario, Coscione, Sam, Robert, Emil, Sibo, Bea, Cecilia e Pierluigi!

Per cui, tirando le somme, non resta che ringraziare colui che forse è stato più che un compagno di vita e di suonate: senza di te non so quanti incontri avrei perso, quante idee avrei lasciato sfumare, quante volte mi sarei lasciato condizionare dai miei pregiudizi o dalla mia stupidità. Ma soprattutto quanto il mio fegato starebbe meglio senza di te, grazie Daniele Abo Carcassi!

8. Referenze

8.1. Bibliografia

CHIN, Roger S., CHANSON, Samuel T., *Distributed Object-Based Programming Systems* (1991), in "ACM Computing Surveys", Vol. 23 No.1: 91-124.

CHOWNING, John M., *The Synthesis of Complex Audio Spectra by Means of Frequency Modulation* (1973), in "Journal of Audio Engineering Society", Volume 21 Issue 7: 526-534.

DIGIUGNO, Giuseppe, KOTT, Jean, GERZO, Andrew, *Progress Report on the 4X Machine and Its Use* (1981), in "Proceedings of International Computer Music Conference 1981", North Texas State University.

DODGE, Charles, JERSE Thomas A., (1997), *Computer Music: Synthesis, Composition and Performance,* Schirmer, New York, USA

DOORNBUSCH, Paul, *Early Computer Music Experiments in Australia and England* (2017), in "Organised Sound", Cambridge University Press, Volume 22, Issue 2: 297-307

DOORNBUSCH, Paul, *Computer Sound Synthesis in 1951: The Music of CSIRAC* (2004), in "Computer Music Journal", MIT Press, Volume 28 Issue 1: 10-25

FAVREAU, Emmanuel, FINGERHUT M., Koechlin, O., POTACSEK, P., PUCKETTE, M. ROWE, R., *Software Developments for the 4x Real-time System* (1986), in "Proceedings of International Computer Music Conference 1996", North Texas State University.

GIBSON, James J. (1977), *The Theory of Affordances,* in R. Shaw e J. Bransford (edited by), Perceiving, Acting and Knowing, Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, New Jersey: 67-82.

GIOMI, Francesco, *An early case of algorithmic composition in Italy* (1996), in "Organized Sound", Cambridge University Press, Volume 1, Issue 3: 179-182.

MATHEWS, Max V., *The Digital Computer as a Musical instrument* (1963), in "Science, New Series", Cambridge University Press, Vol. 142, No. 3592: 553-557.

MCCARTNEY, James, *Rethinking the Computer Music Language: SuperCollider* (2002), in "Computer Music Journal", MIT Press, Volume 26 Issue 4: 61-68.

PAROLINI, Giuditta, *Music without Musicians ... but with Scientists, Technicians and Computer Companies* (2017), in "Organized Sound", Cambridge University Press, Volume 22 Special Issue 2: 286-296.

PUCKETTE, Miller, *Something Digital* (1991), in "Computer Music Journal", MIT Press, Volume 15 Issue 4: 31-43.

PUCKETTE, Miller, *Max at Seventeen* (2002), in "Computer Music Journal", MIT Press, Volume 26 Issue 4: 65-69.

ROADS, Curtis e STRAWN, John(1985), *The Computer Music Tutorial,* MIT Press, Cambridge, Massachusetts.

RITCHIE, Dennis M. (1980), *The evolution of the Unix Time-Sharing System* in J. M. Tobias (edited by), *Lecture Notes in Computer Science vol 79, Language Design and Programming Methodology*, Springer, Berlin, Heidelberg, Germany

ROADS, Curtis (1996), *Foundation of Computer Music,* MIT Press, Cambridge, Massachusetts.

VALLE, Andrea (2015), *Introduzione a Supercollider,* Apogeo Education - Maggioli Editore, Milano.

VERCOE, Barry (1984), *The Synthetic Performer in the Context of Live Performance,* in "Proceedings of International Computer Music Conference 1984", North Texas State University.

VERCOE, Barry e PUCKETTE, Miller (1984), *Synthetic Rehearsal: Training the Synthetic Performer,* in "Proceedings of International Computer Music Conference 1984", North Texas State University.

VERCOE, Barry (2000), *Foreword* in R. Boulanger (edited by), *The Csound Book*, MIT Press, Cambridge, Massachusetts: 26, 30.

WANG, Ge (2008), *A History of Programming and Music*, in N. Collins, J. D'Escrivan (edited by), Cambridge Companion to Electronic Music, Cambridge University Press, Cambridge, England.

8.2. Sitografia

[1]http://www.audiosynth.com/icmc96paper.html - ICMC ‘96 Proceedings, J. McCartney (Gennaio 2020)

[2]https://csound.com/download.html - Csound official site

[3]https://issuu.com/adpware/docs/mc101/272 - 101 MCmicrocomputer, C. Giustozzi, S. Polini (Febbraio 2020)

[4]https://issuu.com/adpware/docs/mc102/241 - 102 MCmicrocomputer, S. Polini (Febbraio 2020)

[5]https://issuu.com/adpware/docs/mc105/292 - 102 MCmicrocomputer, T. Masi (Febbraio 2020)

[6]http://www.linfo.org/pdp-7.html - Linux Information Project (Febbraio 2020)

[7]http://www.linfo.org/thompson.html - Linux Information Project (Febbraio 2020)

[8]https://www.musicainformatica.it/argomenti/4x.php - 4X, A. Di Nunzio (Gennaio 2020)

[9]https://supercollider.github.io/ - SuperCollider, sito ufficiale (Gennaio 2020)

[10]https://tools.ietf.org/html/rfc782 - Workstation Environment, Request For Comments 782 (Gennaio 2020)

[11]https://youtu.be/P85X1Ut3Hfc - Composing a Piece Part III, E. Fieldsteel (Dicembre 2019)

[12]https://youtu.be/vOYky8MmrEU - The Synthethic Performer, B. Vercoe (Gennaio 2020)

[13]https://web.archive.org/web/20080710144930/http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html# - The Early History of Smalltalk, abstract, A. Kay (Febbraio 2020)

8.3. Discografia

Stockhausen Karlheinz,
Gruppen Für 3 Orchester in *Stockhausen Complete Edition no. 5*
Stockhausen-Verlag,
1992

Carcassi Daniele, Giomi Francesco, Minus - Collettivo d'Improvvisazione, Onorato Giovanni,
Sintagmi,
Switch Music Recordings,
2019