UNIVERSITÀ DI PADOVA **FACOLTÀ DI INGEGNERIA**

TESI DI LAUREA

SINTESI AUDIO IN TEMPO-REALE PER UNA INSTALLAZIONE MUSEALE INTERATTIVA

Laureando: Michael Lovisa

Relatore: Dott. Federico Avanzini

Correlatore: Dott. Enrico Marchetto

Corso di Laurea Triennale in Ingegneria Informatica

Anno Accademico 2011/2012

Alle esplosioni nel cielo di Londra

Sommario

In questa tesi verrá sviluppato un progetto software per la simulazione e la ricostruzione virtuale di una parte della strumentazione elettronica dello Studio di Fonologia di Milano della RAI [3], uno studio fondato negli anni ’50 che ha avuto importanza fondamentale nella nascita della musica elettronica ed elettroacustica in Europa consentendo ai compositori di creare nuova musica con nuovi mezzi e un nuovo approccio. Il modello verrá sviluppato in linguaggio *Pure Data*, un linguaggio di programmazione grafico in real-time. Il lavoro fa parte del progetto europeo D.R.E.A.M. [3] che punta alla costruzione di un’installazione interattiva da esibire al Museo degli strumenti musicali nel Castello Sforzesco a Milano. Nella visione complessiva del progetto, questo lavoro di tesi ha fornito un codice che potesse essere il punto di partenza per modifiche e sviluppi, presenti e futuri.

Di seguito una descrizione del contenuto dei capitoli.

Primo capitolo É un capitolo introduttivo che presenta le conoscenze base sul suono. Si parte dai parametri che lo descrivono, passando per le tipologie di forme d’onda prodotte dagli oscillatori e i vari tipi di filtri, e concludendo sulle tecniche di sintesi piú comuni.

Secondo capitolo Presenta una panoramica sulle tecnologie utilizzate, ossia il linguaggio *Pure Data*, il protocollo *Open Sound Control* e il concetto di ”virtual analog”. Vengono presentati brevemente il linguaggio e gli utilizzi possibili; ci si sposta poi all’analisi del software e degli oggetti a disposizione. Si passa alla presentazione del protocollo *OSC*, ad alcuni semplici esempi, per concludere sui concetti introduttivi al ”virtual analog”.

Terzo capitolo Viene ripercorsa la storia dello Studio di Fonologia RAI di Milano, ne vengono presentati i componenti ed infine si espongono gli obiettivi complessivi del progetto D.R.E.A.M.

Quarto capitolo Presenta in dettaglio il lavoro svolto e le *patch* implementate seguendo passo passo il processo di sviluppo del modello delle interfacce, del modello di alcuni componenti e della comunicazione tramite *OSC*.

Quinto capitolo Espone brevemente i risultati finali del progetto D.R.E.A.M. mostrando gli sviluppi del codice sviluppato precedentemente.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Sintesi del suono

1.1 Concetti base del suono

Il suono é un'oscillazione, prodotta da un corpo vibrante detto sorgente, che si propaga attraverso un mezzo trasmissivo (solitamente l'aria) e raggiungendo l'orecchio provoca una sensazione uditiva. Piú in particolare la sorgente, nel momento in cui viene fatta vibrare, trasmette il proprio movimento alle particelle a lei adiacenti facendo in modo che questo moto di particelle da lei generato continui nel mezzo fino ad arrivare all'apparato uditivo. Il movimento delle particelle, ossia il suono, puó quindi essere visto come un'onda che é possibile rappresentare in un grafico cartesiano, ponendo il tempo nell'asse x e lo spostamento delle particelle nell'asse y, per descriverne i vari parametri.

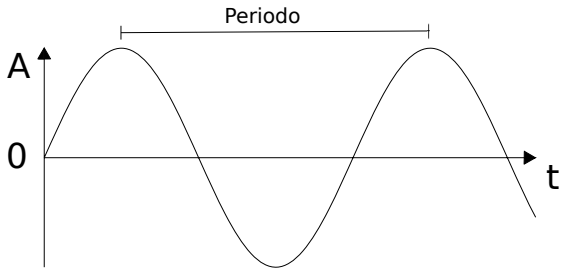


Figura 1.1: *Onda sonora.*

Periodo É il parametro che indica la durata dell'oscillazione ed é definito come il lasso di tempo impiegato da una particella per compiere l'intero movimento ondulatorio che le permette di ritornare nella posizione di partenza; é misurato in secondi o in sottomultipli del secondo. Dal numero di periodi al secondo si determina la frequenza di un'onda, parametro misurato in Herz (Hz) che determina l'altezza di un suono (da grave ad acuto).

Ampiezza Indica di quanto le particelle si spostano dalla posizione di quiete durante la loro oscillazione ed é l'intensitá oggettiva intesa come livello di pressione che queste esercitano con-

tro un ostacolo. Data la risposta non lineare dell'orecchio umano all'intensitá sonora, questa viene misurata secondo la scala logaritmica dei decibel (Db); l'ampiezza assieme alla frequenza determina l'intensitá soggettiva.

Fase É il punto fisso da cui si inizia a misurare il periodo; non é possibile definirla in un suo aspetto ben definito, ma sempre rispetto ad un punto di riferimento. Questo parametro si utilizza nel confronto fra due onde: queste possono essere in fase quando hanno la stessa fase, in controfase quando sono sfasate di 180 gradi, in quadratura di fase quando sono sfasate di 90 gradi, oppure sfasate di un angolo diverso.

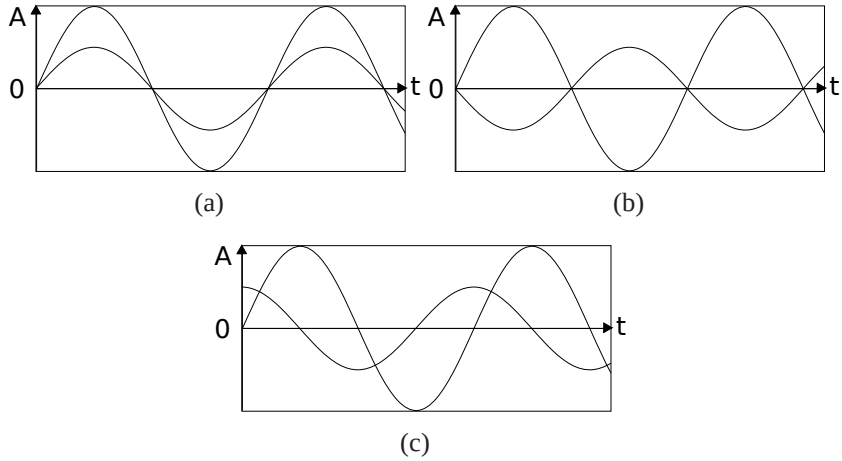
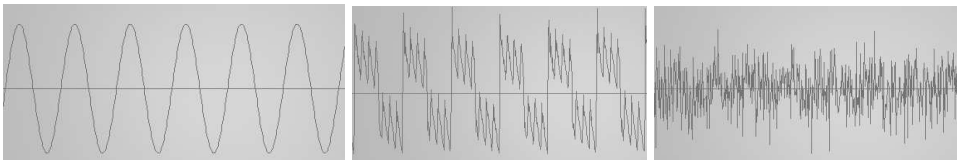


Figura 1.2: *Sfasamenti tra onde: in fase (a), controfase (b), quadratura di fase (c).*

Finora si é parlato di onda periodica senza specificarne la forma, ed assumendola sinusoidale. Le onde possono essere raggruppate in tre macro categorie: semplici, complesse, aperiodiche. Le onde semplici sono quelle onde periodiche descritte da funzioni seno, vengono anche dette pure. Le onde complesse sono quelle onde periodiche non sinusoidali che presentano anomalie nelle curve. Le onde aperiodiche sono onde che non possono essere scomposte in periodi, sono irregolari e tipicamente descrivono rumori. La quasi totalitá dei suoni naturali e dei suoni prodotti da strumenti reali ha caratteristiche intermedie tra le onde complesse e quelle aperiodiche.



(a) (b) (c)

Figura 1.3: *Onda semplice (a), complessa (b), aperiodica (c).*

Per descrivere le onde semplici bastano i parametri descritti di periodo, frequenza e ampiezza; invece per descrivere onde complesse é preferibile scomporle in una serie di onde semplici che sono descrivibili tramite i parametri giá descritti. Questo procedimento si rifá alla teoria sviluppata da Jean Baptiste Joseph Fourier la quale afferma che una qualsiasi funzione limitata e continua o con discontinuitá di prima specie si puó descrivere come somma di N funzioni dello stesso tipo; al tendere di N all'infinito, la somma converge alla funzione. Semplificando e applicando questa affermazione ad un'onda complessa possiamo quindi affermare che questa puó essere descritta come somma di N onde sinusoidali: maggiore é N migliore sará l'approssimazione.

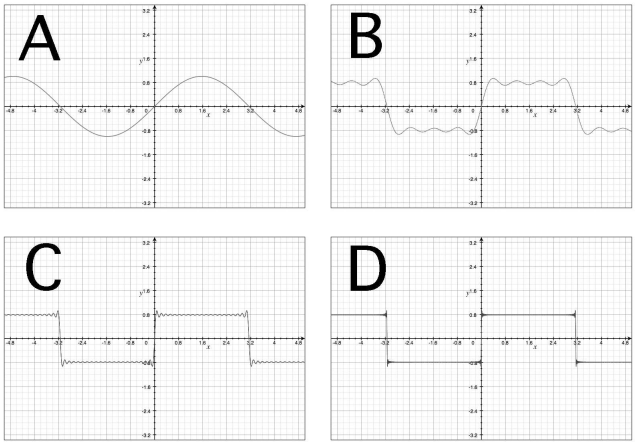


Figura 1.4: *Onda complessa e approssimazioni con numeri diversi di parziali: N=1 (A), N=6 (B), N=40 (C), N=200 (D).*

Ogni sinusoide che compone l'onda viene detta parziale; le parziali costituiscono lo spettro dell'onda. A seconda della frequenza le parziali vengono suddivise in armoniche e inarmoniche. Le parziali armoniche sono quelle parziali che hanno frequenza uguale ad un multiplo intero della frequenza dell'onda complessa; vengono indicate da numeri progressivi interi (1, 2, 3, ecc) che indicano il rapporto con la fondamentale (1 = stessa frequenza, 2 = frequenza doppia, 3 = frequenza tripla, ecc). Le parziali inarmoniche invece sono quelle con frequenza uguale a multipli non interi della frequenza dell'onda complessa. Se lo spettro di un suono é costituito anche da parziali inarmoniche, questo viene detto spettro inarmonico.

Inviluppo L'inviluppo di un suono descrive come questo si modifica nell'arco della sua durata [6]; per semplicitá lo si tratta relativo all'ampiezza del suono, anche se puó essere relativo al comportamento di un filtro o ad altri parametri. Viene definito inviluppo ADSR e si puó scomporre in quattro fasi:

- attack (A), indica il tempo che impiega il suono a raggiungere la sua massima ampiezza;
- decay (D), indica il tempo che impiega il suono a raggiungere il livello di sustain dopo aver raggiunto la massima ampiezza;
- sustain (S), é l'unico parametro che non indica un tempo, si misura in percentuale e indica il livello mantenuto fino alla fase successiva;
- release (R), indica il tempo che impiega il suono a raggiungere ampiezza nulla.

É un parametro che vedremo essere indispensabile nelle varie tecniche di sintesi per scolpire il suono.

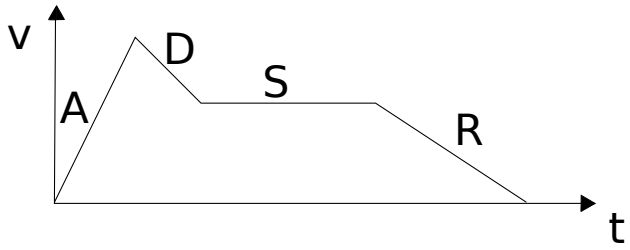


Figura 1.5: *Inviluppo ADSR.*

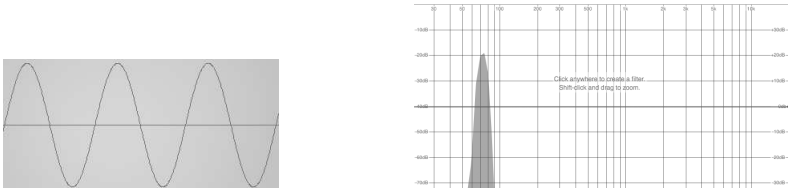
1.2 Strumenti per la sintesi audio

Dopo aver introdotto i concetti base del suono, possiamo passare a definire la sintesi audio e a vedere gli strumenti utilizzati a questo scopo.

Sintesi audio La sintesi audio indica il processo di creazione artificiale del suono tramite circuiti elettronici. Puó avere la finalitá di ricreare il suono di uno strumento tradizionale oppure di creare un timbro non esistente in natura. Il termine sintetizzatore deriva dal nome di uno strumento prodotto nel 1951 dalla RCA Comporation, l'Electronic Music Synthesizer, anche se questo non generava effettivamente il suono in tempo reale. La metodologia di generazione e modifica del suono si basa su due componenti: l'oscillatore e il filtro.

Oscillatori Un oscillatore é un componente che produce un suono periodico mediante la continua ripetizione di una forma d'onda scelta fra diversi tipi:

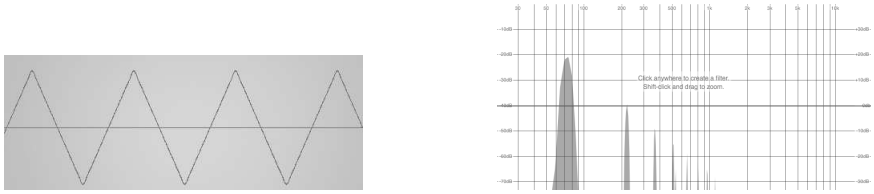
- sinusoidale (sine), produce un suono puro, privo di armoniche;



(a) (b)

Figura 1.6: *Onda seno: oscilloscopio (a) e spettro (b).*

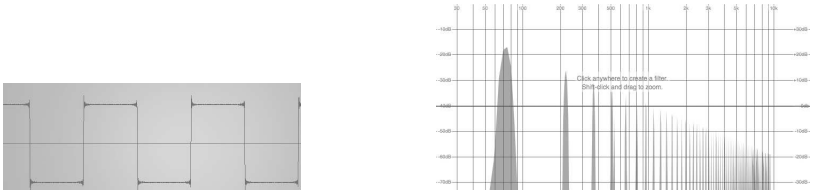
- triangolare (triangle), produce un suono simile all'onda sinusoidale ma presenta anche delle armoniche dispari con ampiezze che decrescono esponenzialmente rendendo il suono cupo o sordo;



(a) (b)

Figura 1.7: *Onda triangle: oscilloscopio (a) e spettro (b).*

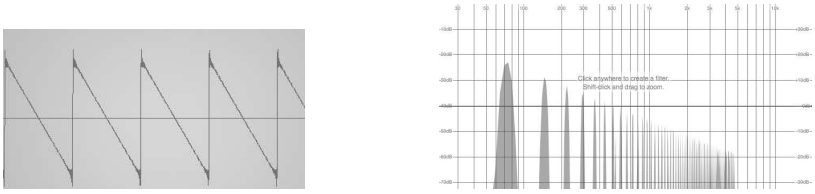
- quadra (square), produce un suono piú ricco di alte frequenze rispetto all'onda triangolare presentando armoniche dispari con ampiezze che decrescono costantemente;



(a) (b)

Figura 1.8: *Onda square: oscilloscopio (a) e spettro (b).*

- a dente di sega (sawtooth), produce un suono molto ricco di armoniche pari e dispari con ampiezze che decrescono costantemente;



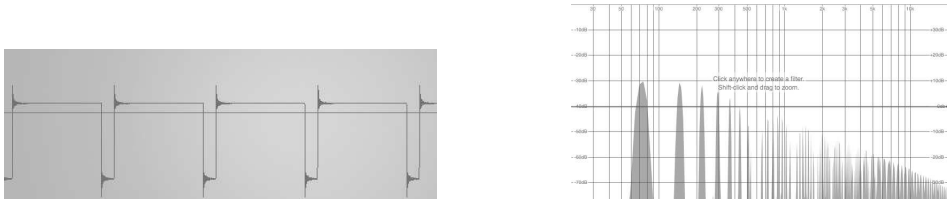
(a) (b)

Figura 1.9: *Onda sawtooth: oscilloscopio (a) e spettro (b).*

- rettangolare (pulse), produce un suono simile all'onda quadra costituito peró da tutte le armoniche dove le ampiezze delle armoniche pari sono differenti da quelle delle armoniche dispari. Quest'onda non ha una struttura fissa ma puó essere modificata regolandone la larghezza (pulse width) raggiungendo ai suoi estremi un suono brillante oppure un suono nasale;

(a) (b)

Figura 1.10: *Onda rect: oscilloscopio (a) e spettro (b).*

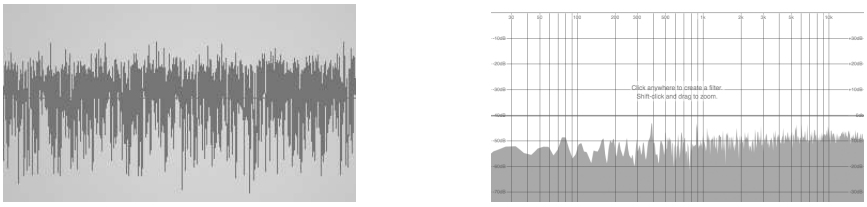


(a) (b)

Figura 1.11: *Onda pulse: oscilloscopio (a) e spettro (b).*

Altri tipi di oscillatori che non producono onde periodiche vengono detti generatori di rumore e posso produrre due tipi diversi di rumore:

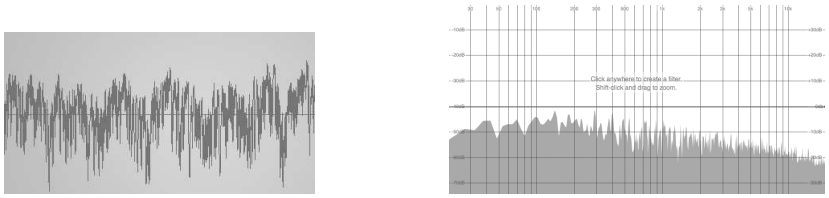
- rumore bianco (white noise), produce un suono aperiodico con ampiezza costante su tutto lo spettro delle frequenze;



(a) (b)

Figura 1.12: *White noise: oscilloscopio (a) e spettro (b).*

- rumore rosa (pink noise), produce un suono aperiodico dove le componenti a bassa frequenza hanno ampiezza maggiore rispetto alle componenti ad alta frequenza;



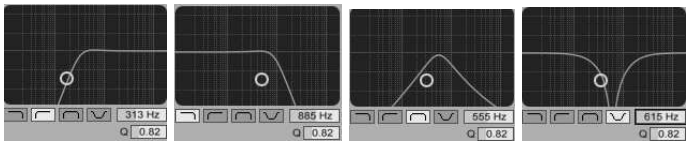
(a) (b)

Figura 1.13: *Pink noise: oscilloscopio (a) e spettro (b).*

Filtri I filtri sono i componenti in grado di alterare lo spettro armonico e quindi il timbro di un suono enfatizzando oppure eliminando determinate bande di frequenze. Ne esistono di diversi tipi e si differenziano per il loro comportamento:

- filtro passa alto, permette il passaggio delle frequenze al di sopra di una frequenza fissata detta frequenza di taglio;
- filtro passa basso, permette il passaggio delle frequenze al di sotto di una frequenza fissata detta frequenza di taglio;
- filtro passa banda, deriva dall'accoppiamento di un filtro passa alto e di un filtro passa alto. Permette il passaggio delle frequenza comprese fra le frequenze minima e massima fissate;
- filtro a soppressione di banda (notch), l'opposto di un passa banda ed elimina una ristretta banda di frequenze lasciando passare le altre inalterate;

Ogni filtro ha due caratteristiche che descrivono ulteriormente il suo comportamento: la risposta del filtro e la risonanza. La filter response indica la pendenza con cui il filtro elimina le frequenze al di fuori della propria banda passante (12dB, 24dB,ecc); la risonanza indica invece quanto vengono enfatizzate le frequenze prossime alla frequenza di taglio.



(a) (b) (c) (d)

Figura 1.14: *Tipi di filtri: passa-alto (a), passa-basso (b), passa-banda (c), soppressione di banda (d).*

1.3 Tecniche di sintesi

Sintesi additiva É la prima tecnica che é stata sviluppata ed utilizzata; si basa sulla somma di forme d'onda elementari per la creazione di una forma d'onda piú complessa [2]. L'idea alla base di questa tecnica deriva dall'analisi di Fourier, infatti se una qualsiasi onda é scomponibile in piú onde sinusoidali, allora tramite il procedimento inverso sará possibile ottenere una qualsiasi onda sommando piú onde sinusoidali pure. Nel caso in cui le onde semplici siano multiple tra loro, la sintesi viene detta per armoniche e il suono risultante verrá percepito come un unico suono avente un'altezza definita e riconoscibile.

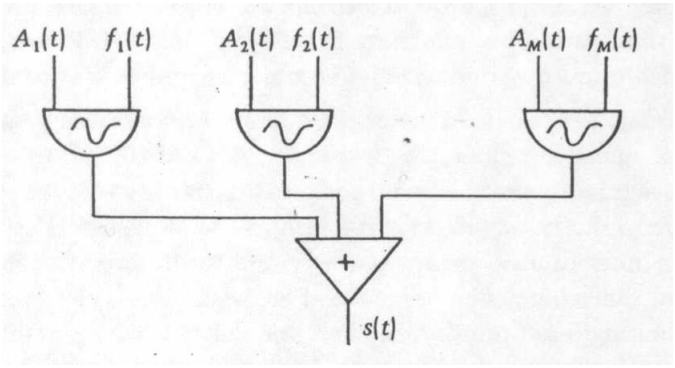


Figura 1.15: *Schema sintesi additiva.*

Un classico esempio di strumento che utilizza questa tecnica di sintesi del suono é l'organo a canne; qui infatti le canne producono un suono semplice, e per ottenere un suono piú ricco vengono utilizzate piú canne che suonano contemporaneamente a differenti altezze multiple della fondamentale.

Analizzando il suono di uno strumento reale si nota che l'energia spettrale si concentra attorno a poche bande di frequenza; queste bande corrispondono alle onde sinusoidali da cui é costituito il suono. Si nota che ogni parziale ha una propria evoluzione nel tempo, ad esempio nell'attacco saranno maggiormente presenti alcune parziali che poi diventeranno trascurabili nelle fasi successive. In generale le parziali possono essere o non essere armoniche della fondamentale.

Figura 1.16: *Organo a canne.*

La sintesi additiva sinusoidale puó essere descritta dalla seguente formula:

$$s(n) = \sum_k A_k(n) * \sin(2\pi \frac{f_k(n)}{F_s} n + \Phi_k) \tag{1.1}$$

dove le onde sinusoidali coinvolte nella sintesi hanno ampiezza $A_k(n)$ e frequenza $f_k(n)$ tempo varianti.

Questa tecnica é teoricamente in grado di riprodurre qualsiasi timbro, ma ha bisogno di una grande quantitá di dati in quanto il numero di onde semplici dovrebbe tendere all'infinito per ricreare fedelmente un suono; il carico di informazioni mette a dura prova la velocitá di calcolo. Per questo motivo la sintesi additiva é poco utilizzata nei prodotti commerciali e rimane esclusivamente una tecnica di studio.

Sintesi sottrattiva La sintesi sottrattiva é l'opposto della sintesi additiva, ossia parte da una forma d'onda complessa per modificarla eliminando tramite uno o piú filtri una parte delle componenti armoniche dello spettro.

Figura 1.17: *Schema sintesi sottrattiva.*

Negli anni ’50 questa tecnica prese il posto della sintesi additiva, in quanto consentiva di ottenere un migliore risultato in minor tempo. Infatti, avendo a disposizione pochi oscillatori, per creare un suono complesso tramite sintesi additiva serviva registrare e sovrapporre ogni singola forma d’onda che ne costituiva lo spettro. Invece utilizzando la sintesi sottrattiva bastano un oscillatore che produca un’onda ricca di armoniche, un filtro per eliminare una parte dello spettro armonico del suono generato e un amplificatore per disegnarne l’inviluppo di ampiezza. I filtri lineari utilizzati nella sintesi sottrattiva sono descritti dall’equazione alle differenze:

$$y(n) = \sum_i b_i x(n-i) - \sum_k a_k y(n-k) \tag{1.2}$$

dove a_k e b_i sono i coefficienti del filtro e $x(n)$ e $y(n)$ sono rispettivamente i segnali di ingresso e di uscita. Il filtro ha una risposta in frequenza definita da

$$H(f) = \frac{Y(f)}{X(f)} \tag{1.3}$$

dove $X(f)$ e $Y(f)$ sono gli spettri del segnale di ingresso e uscita. A seconda dell’andamento della risposta in frequenza si puó variare l’andamento dello spettro del segnale d’ingresso. Se il filtro é statico, quindi se i parametri del filtro non cambiano, rimane costante anche il suo effetto; se invece i parametri sono tempo varianti, cambia anche la risposta in frequenza del filtro.

Per questo motivo nei sintetizzatori che operano la sintesi sottrattiva é presente un controllo di inviluppo anche per il filtro. Anche questa tecnica stata utilizzata in strumenti tradizionali, come per esempio il violino e la chitarra acustica, dove la cassa risonante svolge la funzione di filtro che modella il timbro dello strumento. Un altro classico esempio é il suono prodotto da una tromba con sordina: quest’ultima si comporta come un filtro passa-alto che blocca le armoniche piú gravi e lascia passare le armoniche piú acute. Per quanto riguarda invece i sintetizzatori, i piú famosi Moog e Roland sono stati i primi basati su sintesi sottrattiva.

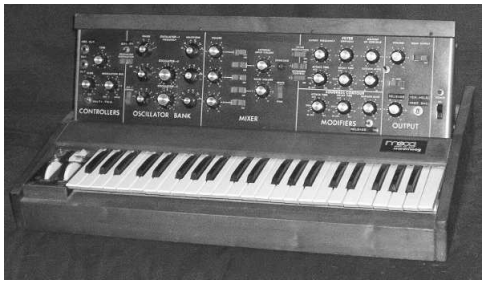


Figura 1.18: *Sintetizzatore Moog a sintesi sottrattiva.*

Sintesi FM La sintesi a modulazione di frequenza (frequency modulation) [7] é stata sviluppata nel 1979 da John Chowning al "Center computer research in music and acustic" dell'universitá di Stanford. Deriva dalla tecnica del vibrato che modula la frequenza di un oscillatore con un low frequency oscillator (LFO), ossia con un oscillatore a bassa frequenza (esempio 20Hz). Nel momento in cui la frequenza dell'oscillatore viene invece modulata utilizzando un oscillatore a frequenza audio, si parla di modulazione di frequenza. Ricapitolando, si ha modulazione di frequenza quando un oscillatore é utilizzato per variare la frequenza di un secondo oscillatore; questi prendono rispettivamente il nome di modulatore e di portante. Questa tecnica é divenuta molto popolare da quando é stata implementata nei sintetizzatori Yamaha serie DX, in quanto permette di ottenere timbri molto ricchi e differenti utilizzando pochi oscillatori. Il suo funzionamento si basa sulle proprietá matematiche di una formula:

$$s(t) = \sin(2\pi f_c n + \Phi(t)) \tag{1.4}$$

dove $\Phi(t)$ é il segnale modulante in ingresso e f_c é la frequenza della portante. Tralasciando per ora l'indice di modulazione, si puó affermare che ogni armonica della portante sará quindi moltiplicata per un numero infinito di volte all'interno dello spettro risultante. L'indice di modulazione quindi determina la profonditá della modulazione, ossia piú si incrementa questo indice e piú armoniche appariranno nel suono risultante. Molte di queste armoniche si estenderanno oltre la banda di udibilitá dell'orecchio umano, ma quelle percepite saranno comunque un numero molto elevato. Gli oscillatori utilizzati nella modulazione di frequenza prendono il nome di operatori. Il primo sintetizzatore ad utilizzare questa tecnica di sintesi é stato il Yamaha DX7 che disponeva di 6 oscillatori sinusoidali configurabili in diversi algoritmi per modificarne il ruolo di modulatori e portanti.

Figura 1.19: *Yamaha DX7.*

Sintesi granulare La sintesi granulare si fonda sull'idea di creare dei suoni complessi tramite suoni piú semplici, come la sintesi additiva; ma mentre quest'ultima si basa sulla sovrapposizione temporale di sinusoidi, la sintesi granulare si basa sulla successione di forme d'onda di brevissima durata dette grani [4]. I grani hanno ognuno una propria altezza, un'ampiezza, una durata (tipicamente di alcuni millisecondi, al limite della percezione uditiva umana), un proprio timbro ed una propria collocazione spaziale. Per creare un suono di elevata complessitá e dinamicitá é necessario l'utilizzo di un insieme elevato di grani. Queste tecnica nasce nella mente del fisico inglese Dennis Gabor che pubblicó due articoli nel 1946 e nel 1947 nei quali sosteneva la possibilitá di ricreare qualsiasi tipo di suono tramite la combinazione di migliaia di grani.

Negli anni ’60 il compositore Iannis Xenakis riprese questa idea cercando di applicarla, ma la sua visione non poté attuarsi per la mancanza della tecnologia necessaria. Solo negli anni ’70 Curtis Roads poté comporre le prime composizioni basate su questa tecnica. Tutt’ora la sintesi é utilizzata da numerosi compositori e scienziati per la sua eleganza formale e per la filosofia operativa che costituisce.

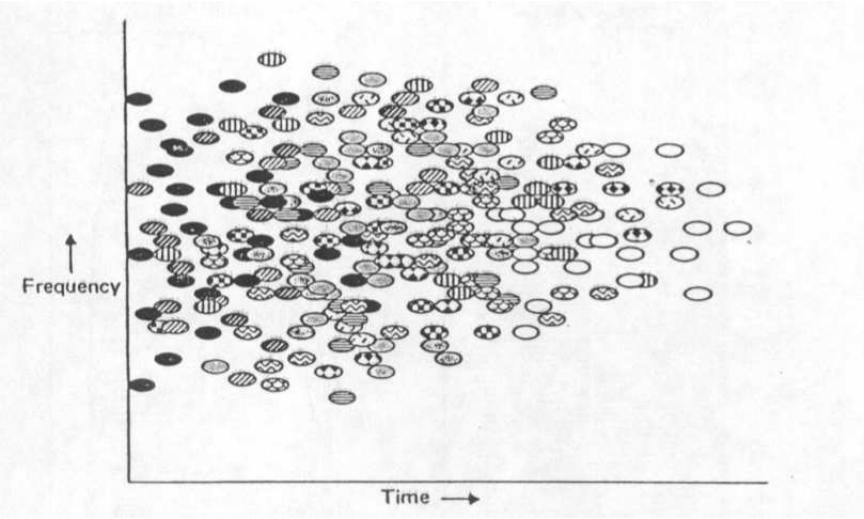


Figura 1.20: *Rappresentazione grafica della sintesi granulare.*

Capitolo 2

Tecnologie utilizzate

2.1 Pure Data

2.1.1 Panoramica

Pure Data[1] é un programma open source sviluppato negli anni ’90 da Miller Puckette e utilizzato negli ambiti audio, video and grafico. Il suo utilizzo non si limita al solo processo di composizione, Pd infatti viene utilizzato anche per live performance musicali, veejaying, analisi audio, interfacciamento con sensori, controllo di robot e interazioni con siti web. É un software multipiattaforma che puó essere installato ed utilizzato su Windows, Mac OS X, GNU/Linux e su piattaforme mobili come iPhoneOS, Android e Maemo.

É il diretto successore, o meglio l’alternativa free, di Max/MSP, programma molto popolare ideato dallo stesso creatore una decina d’anni prima. Entrambi sono dei linguaggi di programmazione veri e propri, in quanto permettono di sviluppare qualsiasi tipo di funzione, da semplice a complessa. Sono basati sulla programmazione grafica, ossia sul creare programmi, detti *patch*, collegando tra loro vari oggetti utilizzando appositi ”cavi”; é un approccio che ricorda molto gli strumenti elettronici del ventesimo secolo in cui i suoni venivano creati e trasformati da oggetti elettronici che erano collegati fra loro tramite patch cables. Non é l’unico aspetto che li differenzia dai linguaggi di programmazione piú tradizionali come C o Java: Pd e Max/MSP sono linguaggi real-time. In C per esempio, un programmatore deve scrivere codice in forma di testo in un file che deve poi compilare per poterlo successivamente utilizzare; in Pure Data invece non esiste un vero e proprio stacco fra il processo di programmazione e il processo di esecuzione del programma in quanto ogni azione ha effetto nel momento stesso in cui viene completata. Per questo motivo molti artisti utilizzano Pd (o Max/MSP) nelle loro performance.

L’unitá su cui si basa lo sviluppo di nuovi programmi in Pd é l’*oggetto* [5]; le patch, come si é visto, sono formate da piú oggetti collegati tra loro con ”cavi” che connettono gli ingressi degli oggetti detti *inlets* e le uscite degli oggetti dette *outlets*. I cavi possono essere di due spessori a seconda del tipo di dati che trasportano: cavi sottili trasportano dati di controllo, cavi spessi trasportano segnale audio. Allo stesso modo esistono due diversi tipi di oggetti: gli oggetti che

[1] Definito in breve utilizzando le iniziali Pd

elaborano dati di controllo e gli oggetti che elaborano segnali audio; questi ultimi si distinguono dai primi per la presenza di una ∼ nel nome.

Le altre unitá base del linguaggio sono i messaggi, le GUI box e i commenti. I messaggi si utilizzano per modificare il modo in cui si comportano gli oggetti, vengono collegati a questi ed inviano il proprio messaggio quando vengono cliccati oppure quando ricevono un comando apposito detto *bang*. Le GUI box sono, come suggerisce il nome, delle ”graphical user interface box”, ossia degli oggetti con cui l’utente puó interagire variandone l’aspetto mentre la patch é in esecuzione; possono presentarsi in forma di *number box*, fader (*slider*), interruttori (*toggle*), manopole (*knob*) o altro.

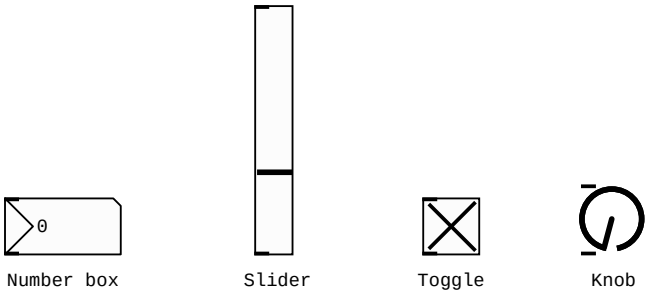


Figura 2.1: *Vari tipi di GUI box.*

Il programma fa anche uso di altri tipi di oggetti non nativi, definiti *externals*, che vengono scritti in linguaggio C dagli utenti e che vengono messi a disposizione online nelle ”Pd user community” e raccolti in librerie. La versione ”Extended” di Pd contiene alcune di queste librerie mettendo a disposizione del programmatore molti oggetti dedicati alle funzioni piú disparate. Per conoscere la funzione di un oggetto, dopo averlo creato, basta visualizzare l’*help* che descrive a parole e con esempi come si comporta l’oggetto in questione. Il linguaggio mette anche a disposizione il costrutto *send-receive* per eseguire connessioni ”wireless” che trasportano dati di controllo, oppure la versione *send∼-receive∼* per segnali audio.

2.1.2 Interfaccia

Una volta avviato, il programma mostra la sua finestra principale.

Questa ha diverse funzioni, tra cui:

- mostra nel terminale le librerie caricate all’avvio e la loro posizione nell’hard disk
- visualizza i messaggi che é possibile stampare dall’interno di una patch con funzione di controllo tramite l’oggetto *print*
- mostra e conteggia gli errori di clip per l’audio in ingresso e in uscita
- consente la configurazione delle impostazioni audio e MIDI

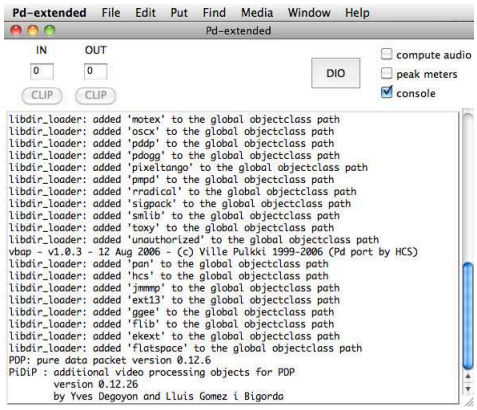


Figura 2.2: *PD window.*

- mostra se avvengono errori di sincronizzazione nell'audio tramite il "DIO errors"
- consente di creare,aprire, salvare le patch
- permette di impostare il percorso delle librerie esterne
- consente di avviare il motore DSP, ossia il "Digital Signal Processor", per cominciare l'elaborazione dei dati audio[2]

Aprendo una nuova patch, verrá aperta un'altra finestra diversa da quella principale. Questa é la "patch window" e rappresenta l'ambiente di lavoro vero e proprio. Puó essere in due diversi stadi: *run mode* in cui la patch é in funzione e non possono essere modificati oggetti al di fuori di quelli interattivi (GUI box), oppure *edit mode* in cui é possibile modificare la patch.

Figura 2.3: *Patch window.*

[2]Se una patch elabora esclusivamente dati, Puredata elabora attivamente fin dal suo avvio; se invece una patch elabora anche segnali audio serve l'attivazione del DSP

2.1.3 Oggetti piú comuni

Verranno qui presentati in tabella alcuni oggetti, che verranno utilizzati in seguito, con il loro simbolo e la rispettiva funzione.

oggetto	funzione
+	somma fra numeri
*	prodotto fra numeri
+~	somma fra due segnali audio
*~	prodotto fra un segnale audio e un numero oppure fra due segnali audio
bang	manda un impulso che attiva messaggi o oggetti quando viene cliccato o quando riceve un dato
int	convertitore da float a int
select	direziona l'output in una delle uscite a seconda del valore di input
adc~	convertitore da segnale analogico a digitale,ingresso
dac~	convertitore da segnale digitale ad analogica,uscita
print	mostra l'output nella console
metro	metronomo
osc~	oscillatore sinusoidale
hp~	filtro passa-alto
lp~	filtro passa-basso
bp~	filtro passa-banda
line~	generatore di rampe lineari

2.1.4 Temporizzazioni

Un aspetto del linguaggio a cui serve dedicare molta attenzione é il corretto ordine delle operazioni. Nel momento in cui in una patch si collegano messaggi a due o piú *inlet* di un oggetto, bisogna tener presente che il risultato dell'operazione puó dipendere dall'ordine in cui sono state eseguite le connessioni. Per questo motivo, due patch identiche possono comportarsi in maniera differente a seconda dell'ordine delle connessioni [5]. Questo perché nella maggior parte degli oggetti, l'*inlet* di sinistra viene definito "hot", ossia i messaggi quando arrivano danno il via all'operazione e alla generazione di un output. Al contrario, se un messaggio raggiunge l'*inlet* destro, questo non fará produrre output all'oggetto in questione. Per esempio, in operazioni come la somma (oggetto +) é necessario che il messaggio contenente il numero da sommare arrivi all'*inlet* destro prima del messaggio all'*inlet* sinistro, altrimenti la somma non sará eseguita. Il problema del corretto ordine delle connessioni viene risolto utilizzando un *trigger*, ossia un oggetto che forza l'ordine di attivazione dei suoi *outlet* da destra verso sinistra.

In figura il *bang* quando viene cliccato manda un impulso al *trigger* per attivarlo.

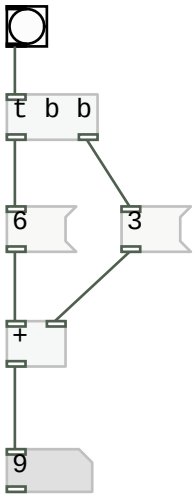


Figura 2.4: *Esempio di utilizzo del* trigger.

2.1.5 Segnali audio

In Pure Data i segnali audio vengono rappresentati da numeri a 32-bit in virgola mobile, tuttavia sono solitamente limitati a 16 o 24 bit a seconda dell'hardware utilizzato. Sia i valori di input che quelli di output assumono valori compresi fra -1 e 1 e la frequenza di campionamento di default é impostata a 44.100 Hz. Pure Data puó leggere o scrivere file audio a 16 o 24 bit a virgola fissa oppure a 32 bit in virgola mobile, nei formati WAV, AIFF o AU tramite gli oggetti *soundfiler*, *readsf*$\sim$ e *writesf*$\sim$. L'elaborazione dei segnali audio viene eseguita dagli oggetti che riportano nel nome il simbolo $\sim$; questi oggetti comunicano tra loro tramite connessioni audio. Gli *inlet* e gli *outlet* sono in generale predisposti sia per segnali audio che per dati di controllo, ma dipende dalle specifiche dell'oggetto. Non é possibile connettere un *outlet* audio ad un *inlet* che non possa ricevere audio, e viceversa. Il percorso complessivo del segnale audio in una patch deve essere aciclico; se sono presenti loop, verrá segnalato un errore "DSP loop". Vedremo in seguito come risolvere questo errore.

2.1.6 Esempio patch

Viene ora presentata una patch di esempio per mostrare le capacitá di Pure Data e la semplicitá della programmazione. É stato implementato un semplice sintetizzatore che produce onde a dente di sega su frequenza pre-impostate tramite numeri di note MIDI, che comprende anche:

- accensione/spegnimento
- controllo di inviluppo ADSR

- effetto di riverbero con controllo di segnale wet,dry e room size
- effetto di delay con controllo del tempo di delay, di volume e del filtro passa-basso per il segnale ritardato
- controllo del numero di note eseguite in loop, decidendo fra quelle pre-impostate
- controllo del tempo
- oscilloscopio che mostra l'onda generata

A sinistra si vede la parte di patch riservata all'interfaccia dello strumento, alla destra la parte relativa alla generazione del suono. É possibile chiudere questa parte di finestra per rimanere con i soli controlli dell'interfaccia.

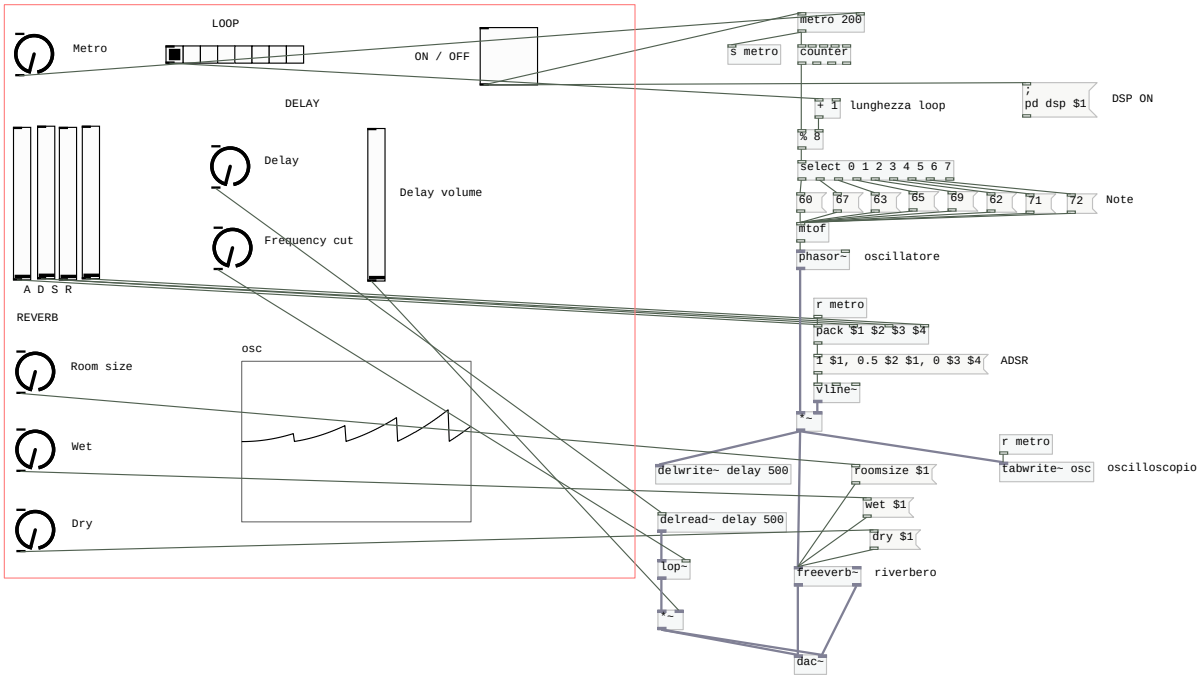


Figura 2.5: *Patch di esempio.*

2.2 Protocollo Open Sound Control

L'Open Sound Control, OSC in breve, é un protocollo di comunicazione che fa parte del linguaggio Pd come di altri linguaggi come Max/MSP, Java, C++ e SuperCollider. É nato per condividere dati di controllo musicale fra diversi strumenti musicali elettronici, computer e altri

device multimediali. É una valida alternativa al protocollo MIDI in quanto permette un piú ampio range di valori possibili, una migliore risoluzione dati, una maggiore flessibilitá, una minore latenza e toglie l'obbligo di possedere un hardware dedicato (interfaccia MIDI). I messaggi OSC infatti viaggiano su reti che possono essere di differenti tipologie come Ethernet, usb, bluetooth, wireless e firewire.

In Pure Data viene utilizzato l'oggetto *sendOSC* che invia i messaggi ricevuti al suo *inlet* utilizzando il protocollo UDP. La destinazione sará un altro computer o un dispositivo, oppure é possibile inviare messaggi ad un'altra applicazione sullo stesso computer. In questo caso, e sará il caso che verrá utilizzato successivamente, il computer in questione si collegherá a sé stesso ossia al "localhost" e i messaggi saranno inviati ad una porta[3] stabilita.

Per instaurare una connessione utilizzando l'oggetto *sendOSC*, serve mandare un messaggio contenente l'indirizzo della destinazione e il numero di porta oppure "localhost" seguito dal numero di porta se la comunicazione é fra applicazioni dello stesso computer. I messaggi utilizzati nella comunicazione sono composti da due parti: la prima indica la destinazione, la seconda contiene i dati. Le destinazioni sono organizzate gerarchicamente come URL[4], vediamo un messaggio di esempio: \mixer\fader1 127. Questo potrebbe presumibilmente essere un messaggio inviato da un mixer (\mixer), relativo alla posizione del primo dei suoi fader (\fader1) e il dato sulla posizione (127).

L'oggetto utilizzato per la ricezione dei messaggi é *dumpOSC*; questo rimane in ascolto nella porta indicata e manda in output i messaggi che riceve. Per indirizzare i messaggi ricevuti ai corretti destinatari, viene utilizzato l'oggetto *OSCroute* che analizza gli indirizzi URL e inoltra i dati su una delle sue uscite.

Passiamo a vedere degli esempi concreti che mostrino come si instaura la connessione ad una porta del "localhost" e come si preparino i messaggi da inviare. Nella figura seguente si puó vedere una patch che estende l'esempio considerato precedentemente, mostrando un mixer con 8 controlli di tipo fader (rappresentati da 8 *slider*) che invia tramite protocollo OSC dei messaggi di controllo al "localhost" sulla porta 9000.

[3]Una porta, indicata da un numero, indica un punto dove é possibile stabilire una connessione TCP o UDP

[4]Uniform Resource Locator, sono gli indirizzi utilizzati in Internet

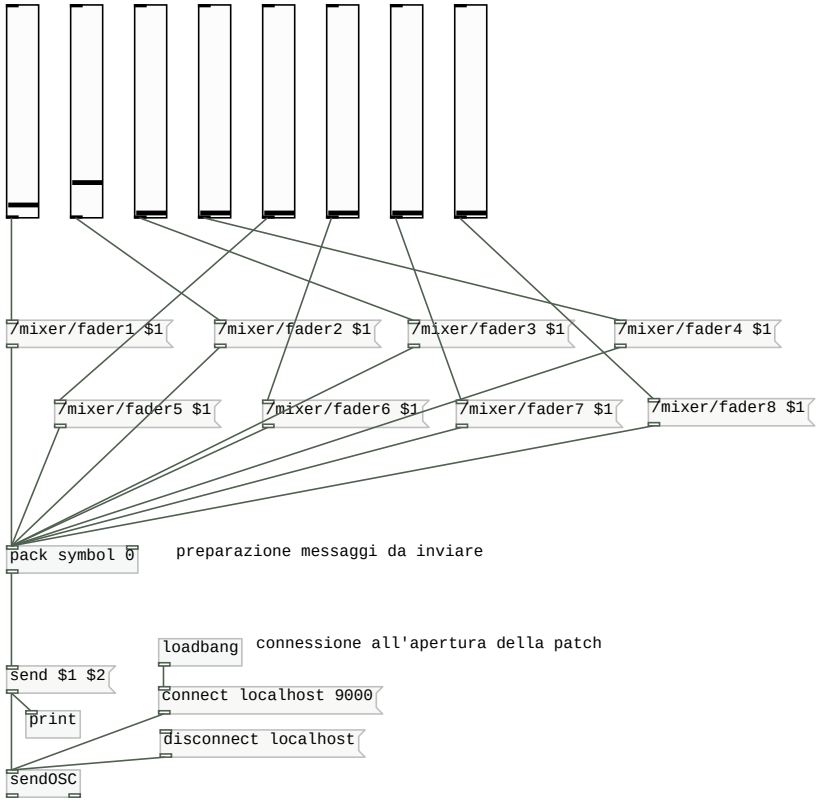


Figura 2.6: *Connessione e preparazione messaggi.*

Nella figura successiva viene riportata la Pd window che mostra l'avvenuta connessione UDP e i messaggi provenienti dall'oggetto *print* dopo aver modificato le posizioni dei primi tre fader.

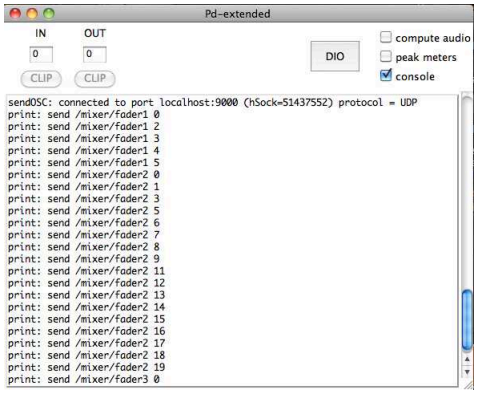


Figura 2.7: *Print dei messaggi inviati.*

Passiamo ora a vedere la patch che riceverá i messaggi inviati dalla prima. Questa contiene due oggetti *OSCroute*: il primo analizza il primo livello dell'indirizzo individuando il compo-

nente che dovrá ricevere il messaggio ossia il mixer, il secondo invece individua il numero di fader.

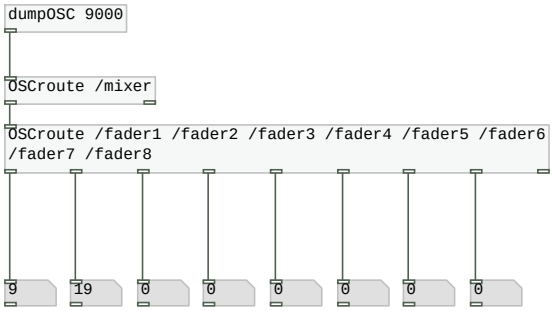


Figura 2.8: *Ricezione e indirizzamento dei messaggi.*

2.3 Virtual Analog

Il *Virtual Analog*, in breve *VA*, é una tecnica di modellazione che si applica a strumenti analogici con il fine di ricrearne il suono nella maniera piú fedele possibile. Anche se questa tecnica ha preso piede in questi ultimi anni, le prime pubblicazioni sperimentali a riguardo sono nate negli anni ’70. Uno dei primi motivi che ha spinto la ricerca in questa direzione é stata la necessitá da parte dei musicisti di utilizzare i suoni dei vecchi sintetizzatori analogici degli anni ’60/’70 cercando di superare i limiti imposti da queste macchine, come la difficoltá nel trasporto, i problemi d’intonazione, la limitata polifonia, la manutenzione, l’assenza del MIDI e della possibilitá di salvataggio di preset. Un altro motivo, sicuramente piú recente, é invece legato alla conservazione degli strumenti elettronici analogici e quindi alla virtualizzazione di questi[5].

Il vero primo passo é stato compiuto negli anni ’90 quando l’azienda svedese Clavia ha introdotto nel mercato il primo sintetizzatore virtual analog, il NordLead[6]. Da quel primo passo, molte aziende si sono focalizzate su questo tipo di prodotti, sia nell’ambiente hardware come Access Virus, Roland, Korg, Alesis, Yamaha, sia nell’ambiente software come Arturia e Applied Acoustics Systems. Ma passiamo a vedere cos’é il virtual analog piú in dettaglio.

Il *VA* é una delle due possibili applicazioni sonore della sintesi a modelli fisici, ossia del tipo di sintesi che non cerca di emulare un suono partendo dalla sua forma d’onda[7], ma parte dall’osservazione dei meccanismi che generano quel suono. Nel caso questa sintesi si applichi a strumenti elettronici analogici si parla di *Virtual Analog*, nel caso invece di strumenti tradizionali mantiene il nome di ”sintesi a modelli fisici”. In quest’ultimo caso, il processo di sintesi parte

[5]”Introduction to the special issue on virtual analog audio effects and musical instruments”, IEEE transactions on audio, speech, and language processing, VOL.18, N. 4, Maggio 2010

[6]”The brief history of virtual analog synthesis”, J. Pekonen, V. Välimäki

[7]Tecnica utilizzata nelle sintesi basate sui modelli del segnale

dallo studio separato di due blocchi funzionali: l'eccitatore, ossia la parte che provoca l'oscillazione, e il risonatore, ossia la parte in cui la vibrazione ha luogo[8]. Nel caso del *VA* invece, non essendo presente una vera e propria oscillazione, la sintesi parte dall'analisi del circuito elettrico dello strumento e tramite simulazioni arriva ad un sistema di equazioni che lo descrive completamente[9].

Questo metodo consente quindi di raggiungere una simulazione piú accurata rispetto ai tipi basati sui modelli del segnale, in quanto riesce a riprodurre anche le imperfezioni dello strumento mantenendo il suono piú "vivo", piú espressivo e con una minore staticitá.

[8] "Sintesi per modelli fisici", dispense per il corso di sistemi di elaborazione per la musica, F. Avanzini

[9] Fine a cui giunge anche la sintesi applicata a strumenti tradizionali

Capitolo 3

Lo Studio di Fonologia RAI

3.1 Storia

Lo Studio di Fonologia nasce nel 1955, periodo del secondo dopoguerra dove le radio erano un mezzo sperimentale e ospitavano gran parte della ricerca legata al suono. In breve tempo lo studio diventó uno dei piú importanti centri internazionali affiancando i giá affermati centri di Colonia (Studio fur Elektronische Musik) e di Parigi (Groupe de Recherches Musicales), focalizzati su filosofie compositive differenti: il primo portavoce della musica elettronica, ossia della musica creata con suoni ottenuti esclusivamente mediante generatori di frequenza, mentre il secondo orientato verso la musica concreta, ovvero ”elaborazione elettronica di suoni o rumori, diremo cosí naturali, prodotti al fine di tale successiva elaborazione”[1].

É ai maestri Luciano Berio e Bruno Maderna che si deve il riconoscimento della fondazione dello Studio, essendo essi riusciti tramite la composizione *”Ritratto di cittá”*, prima opera elettroacustica per voce e nastro, a convincere i dirigenti RAI sulle potenzialitá e sulle possibilitá di uno strumento di questo tipo. Non avendo comunque fondi adeguati ad un progetto di questa portata, lo Studio cominció pian piano a riempirsi di dispositivi artigianali realizzati dal fisico Lietti recuperando materiale di scarto dai magazzini RAI. Venne realizzato il primo oscillatore, a cui seguí un secondo e un terzo, fino ad arrivare ai nove oscillatori che furono il cuore dello Studio, che, a differenza degli altri due centri europei, cercava di mantenere un approccio neutrale sulla filosofia di composizione mantenendo possibile il realizzare musica a partire sia da suoni pre-registrati sia dagli oscillatori.

La complessitá dei componenti presenti, rendeva necessaria la presenza costante di un tecnico che conoscesse a fondo i propri mezzi e che riuscisse a fare da tramite tra i compositori e gli strumenti elettronici. Marino Zuccheri fú l’unico tecnico durante tutto il periodo di attivitá dello Studio, ”maestro tra i maestri, maestro del suono tra i maestri della musica”[2]. L’importanza che rivestiva, non era solo legata alla funzione di tecnico; era lui a provare ogni nuovo componente che Lietti progettava, valutandone la qualitá e l’utilitá musicale, e descrivendo eventualmente come modificarlo. Inoltre era lui stesso ad aiutare i giovani compositori che soggiornavano nello

[1] Radiocorriere 1966, ”Dieci anni di attivitá dello Studio di Fonologia della RAI”
[2] Giovanni Belletti ”Marino Zucchieri in Fonologia”, 2008

Studio per una durata di pochi mesi, tempo non sufficiente ad impadronirsi a pieno del funzionamento degli strumenti a disposizione. Al termine del periodo, dovendo questi avere in mano una composizione che giustificasse i loro studi, si ritrovarono svariate volte con una composizione originale di Zuccheri. La disposizione delle apparecchiature non era fissa, poteva essere modificata a seconda delle esigenza per permettere una comoditá operativa che riuscisse a sormontare i limiti che uno strumento cosí imponente poteva comportare. Dopo i giá citati nove oscillatori, fu costruito un nuovo componente: il generatore di rumore. Questo fu progettato quando i compositori iniziarono a percepire un limite compositivo e ideologico nel lavorare solamente con materiale periodico e statico prodotto dagli oscillatori.

Nel 1968, visto il periodo positivo dello Studio, ci fu un rinnovo delle apparecchiature, sostituendo quasi tutti gli strumenti costruiti in precedenza con dispositivi in commercio. Lo Studio continuó la propria attivitá fino al 28 febbraio 1983, giorno in cui Marino Zuccheri andó in pensione; riportando una sua testimonianza: ”sono andato via dalla RAI come quando si va a casa, ho chiuso la porta dietro alle spalle ed é finita la Fonologia”[3]. Dopo gli anni 60 infatti le radio smisero di essere dei mezzi sperimentali, e la ricerca si spostó nelle universitá proprietarie dei primi calcolatori. Per questo lo Studio fu sempre meno frequentato da compositori e finí la sua storia[4].

3.2 Componenti

In questa sezione verranno presentati i vari componenti[5] presenti nello Studio fin dalla sua apertura e quelli che sono stati inseriti durante i primi anni di attivitá [3].

Oscillatori Gli oscillatori sono stati ideati e costruiti dal fisico Lietti; generano un’onda sinusoidale alla frequenza scelta tramite le due manopole. La manopola sinistra serve per selezionare il range di frequenze a cui si é interessati, mentre la manopola destra serve per muoversi all’interno del range prefissato per selezionare la frequenza di oscillazione desiderata. Nello studio erano presenti nove oscillatori di questo tipo, ognuno con range diversi perché progettati per lavorare come uno strumento acustico (es. chitarra) in cui ogni corda puó riprodurre un range di note differente dalle altre corde. In questo modo, il compositore poteva disporre degli oscillatori per creare fin da subito degli accordi e quindi avere un’idea immediata del risultato.

[3]Parete 1967 di Marino Zuccheri,ed. Die Schachel, Milano, 2005
[4]Intervista a Maddalena Novati, DigiMag 55, giugno 2010
[5]Le immagini riportate sono proprietá di ASdF RAI Milano e di RAI Teche Milano

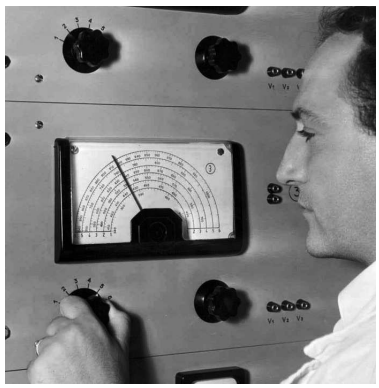


Figura 3.1: *Marino Zuccheri e uno degli oscillatori.*

Mixer Il mixer era composto da nove manopole per la regolazione del volume dei singoli oscillatori.

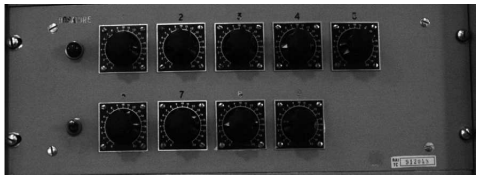


Figura 3.2: *Mixer.*

Generatore di rumore bianco Il generatore di rumore bianco consentiva appunto di creare rumore bianco; era dotato di interruttore per l'accensione e un controllo di volume.

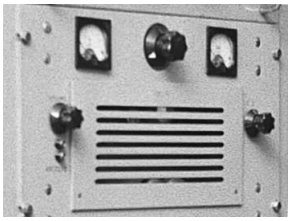


Figura 3.3: *Generatore di rumore bianco.*

Filtri d'ottava Il banco dei sei filtri d'ottava tarati sulle frequenze indicate in figura.

Figura 3.4: *Banco filtri d'ottava.*

Selezionatore d'ampiezza Il selezionatore d'ampiezza venne realizzato nel 1957 e utilizzato per la realizzazione di alcune importanti composizioni tra cui "Scambi" di Henry Pousseur. Il suo funzionamento é simile in prima approssimazione a quello di un noise-gate[6], in quanto dato un segnale in ingresso restituisce la sola parte del segnale che supera una certa soglia, impostata tramite l'apposito rotore. L'ingresso piú appropriato é il rumore bianco, in quanto é un'onda aperiodica che presenta ampiezza non costante. Il selezionatore é dotato anche di un altro rotore con cui é possibile selezionare tra due diversi tempi di rilascio: 1ms oppure 10ms.

Figura 3.5: *Selezionatore d'ampiezza.*

Filtro passabanda variabile Il filtro passabanda variabile KrohnHite 310A é uno dei pochi dispositivi commerciali presenti nello Studio fin dai suoi esordi. É formato da un filtro passa-basso e un filtro passa-alto in serie. É possibile selezionare la frequenza di taglio dei due filtri attraverso le due manopole superiori, scegliendo un valore tra 20 e 200; le manopole inferiori invece selezionano il fattore moltiplicativo (x1, x10, x100, x1000). É possibile quindi selezionare una frequenza di taglio fra 20Hz e 200kHz.

[6]Nel capitolo 4 verrá approfondito il suo funzionamento mostrando anche la differenza tra questo componente e un noise-gate

Figura 3.6: *Filtro passabanda variabile.*

Modulatore ad anello Il modulatore ad anello era costituito da due ingressi relativi ai due segnali che dovevano essere modulati e un'uscita. L'effetto era quello di "generare somma e differenza delle componenti del timbro" [7].

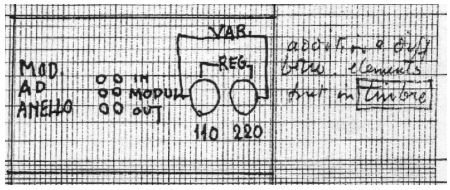


Figura 3.7: *Modulatore ad anello nello schema di John Cage.*

Comparatore Il comparatore serviva per controllare se le frequenze generate da due oscillatori erano in rapporto armonico.

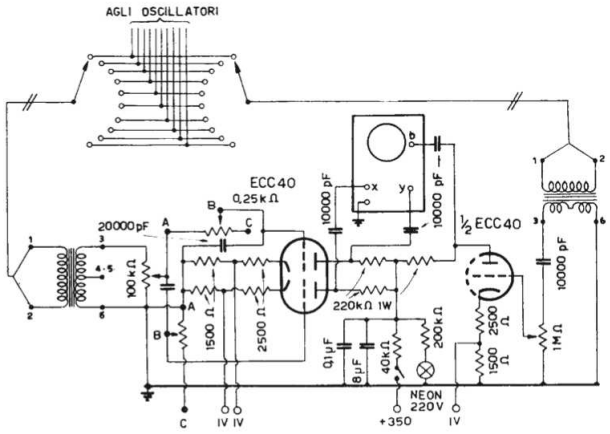


Figura 3.8: *Schema elettrico del comparatore.*

[7] Evoluzione dei mezzi tecnici dello studio di fonologia musicale di Milano, Antonio Rodá

Filtro passa-banda selettivo L'analizzatore d'onda 736A della General Radio era un dispositivo commerciale utilizzato nello Studio come filtro passa-banda selettivo con larghezza di banda 2Hz.

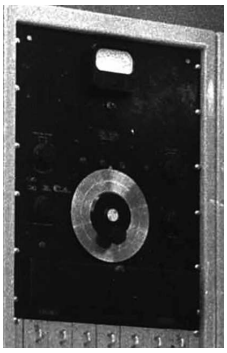


Figura 3.9: *Analizzatore d'onda.*

Modulatore bilanciato Il modulatore bilanciato fa parte dei componenti che sono stati introdotti nello Studio dopo i primi anni di attivitá. É funzionalmente simile al modulatore ad anello, anch'esso infatti genera somma e differenza delle frequenze agli ingressi, ma lo fa utilizzando un doppio circuito di modulazione che elimina le non idealitá nei componenti moltiplicatori.

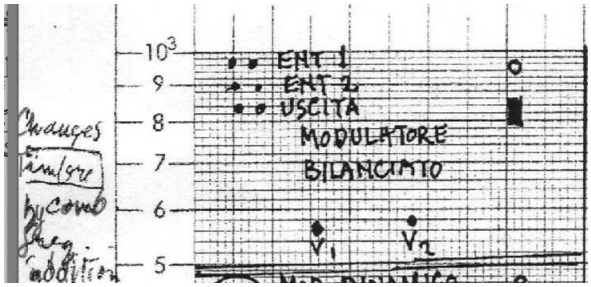


Figura 3.10: *Modulatore bilanciato nello schema di John Cage.*

Modulatore dinamico Il modulatore dinamico, inserito nello Studio nel 1959, modula l'ampiezza del segnale nel primo ingresso utilizzando l'ampiezza del segnale nel secondo ingresso.

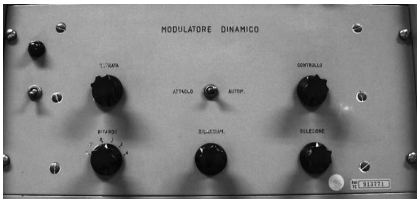


Figura 3.11: *Modulatore dinamico.*

Modulatore d'ampiezza Il modulatore d'ampiezza consentiva di modulare l'ampiezza del segnale d'ingresso con un segnale sinusoidale a bassa frequenza compresa tra 0.5 e 20Hz generando un effetto tremolo.

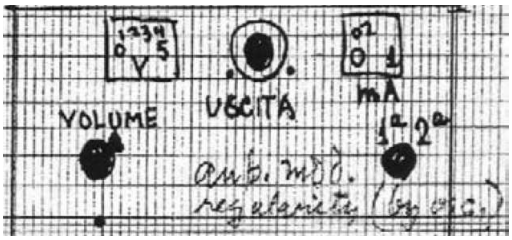


Figura 3.12: *Modulatore d'ampiezza nello schema di John Cage.*

Generatore di toc Affiancava il modulatore d'ampiezza: gli impulsi generati dal dispositivo venivano utilizzati per controllare l'inviluppo di ampiezza dei suoni all'ingresso del modulatore di ampiezza.

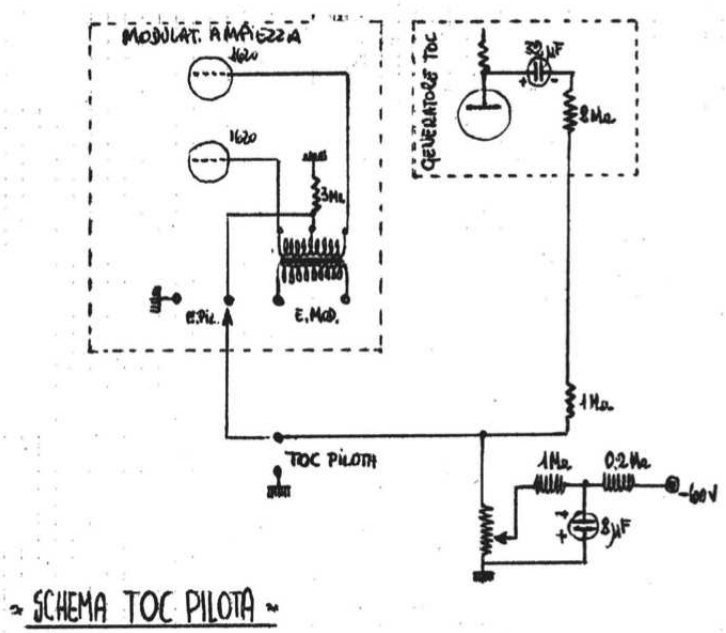


Figura 3.13: *Schema elettrico del generatore di toc e del modulatore d'ampiezza.*

Traspositore di frequenza Il traspositore di frequenza é stato progettato e realizzato da Lietti nel 1959. Disponeva di un solo comando di accensione e realizzava la modulazione in frequenza del segnale in ingresso.

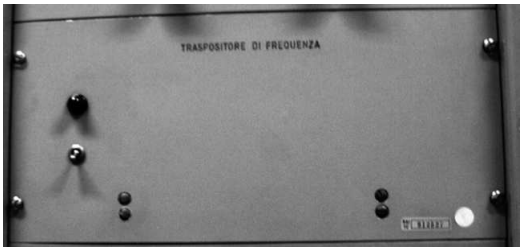


Figura 3.14: *Traspositore di frequenza.*

Banco di filtri passa-alto e passa-basso Il banco di filtri passa-alto e passa-basso é stato incluso tra il 1957 e il 1958; le specifiche si trovano in figura.

(a) (b)

Figura 3.15: *Frequenze di taglio dei filtri passa-alto (a) e passa-basso (b).*

3.3 Rinnovo del 68

In questa sezione vengono presentati i componenti commerciali che sono stati introdotti nello Studio nel rinnovo del 1968 sostituendo gran parte dei vecchi componenti artigianali.

Oscillatori Wavetek Due modelli di oscillatori Wavetek (110 e 115) sono stati introdotti nel rinnovo. Il Wavetek 110 permetteva di generare onde sinusoidali, triangolari e quadre con frequenza di oscillazione compresa fra 0.005 Hz e 1 MHz, selezionabile tramite un rotore con valori da 0 a 10 e un rotore con funzione moltiplicativa. L'oscillatore era dotato di sei uscite separate a seconda dell'onda utilizzata.

Figura 3.16: *Oscillatore Wavetek 110.*

Il modello 115 invece permetteva anche la generazione di rampe, la sincronizzazione di fase e il controllo di frequenza.

Figura 3.17: *Oscillatore Wavetek 115.*

Oscillatore Heathkit Commercializzato a partire dal 1958, la sua data di inclusione nello Studio rimane incerta. Permette la generazione simultanea su due uscite di onde quadre e sinusoidali. Il controllo della frequenza avviene tramite due manopole: quella superiore puó assumere valori compresi fra 20 e 200, mentre quella inferiore seleziona il fattore moltiplicativo (x1, x10, x100, x1000, x10000). Complessivamente quindi l'oscillatore puó generare onde nell'intervallo compreso fra 20Hz e 1MHz. Altri controlli sono dedicati alla selezione del valore massimo di tensione e dell'ampiezza effettiva in uscita, e alla selezione del range per l'onda quadra.

Figura 3.18: *Oscillatore Heathkit AG10.*

Oscillatore a battimento Altro dispositivo con data di inserimento incerta, anche se la datazione 1964 del manuale rende plausibile un suo inserimento nel 1968. Normalmente gli oscillatori a battimento venivano utilizzati nelle trasmissioni radio per demodulare un segnale audio che era stato trasmesso a frequenze piú elevate.

Figura 3.19: *Oscillatore a battimento Brüel & Kjaer.*

Tone burst generator Ha probabilmente sostituito il generatore di toc del vecchio Studio. La sua funzione é simile ad un gate audio con stati di funzionamento aperto e chiuso. Nello stato aperto il segnale all'ingresso viene duplicato all'uscita, nello stato chiuso non sará presente alcun segnale in uscita. Tramite due rotori é selezionabile il tempo di apertura e di chiusura del gate.

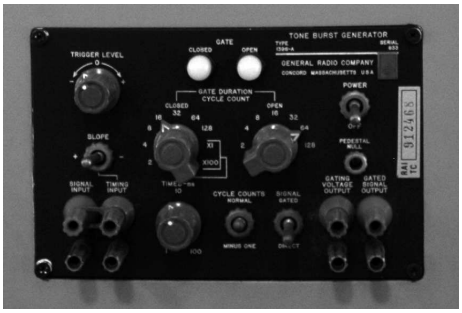


Figura 3.20: *Tone burst generator, General Radio 1398A.*

Generatore di rumore Brüel Kojoer Componente che ha sostituito il generatore di rumore a valvola realizzato da Lietti; é possibile scegliere tra la generazione di rumore bianco nell'intervallo di frequenza fra i 20 e i 20000Hz oppure la generazione di rumore rosa, ossia rumore con energia decrescente con la frequenza di 3dB per ottava.

Figura 3.21: *Generatore di rumore Brüel Kjoer, 1402.*

Filtro passa-banda variabile Componente KrohnHite 310AB42 inserito nello Studio per affiancare il filtro passa-banda variabile KrohnHite 310A giá presente. I due dispositivi hanno le stesse funzioni.

Figura 3.22: *Filtro passa-banda variabile, KrohnHite 310AB42.*

3.4 Conservazione: progetto D.R.E.A.M.

Avendo presentato la storia dello Studio e le prime apparecchiature artigianali che hanno permesso ai compositori di esplorare nuovi territori musicali, si intuisce l'importanza che questo periodo di sperimentazione ha avuto nella nascita della musica elettronica e nasce spontaneo chiedersi se e come é possibile salvare le memorie e i suoni di questo periodo. Maria Maddalena Novati, consulente musicale RAI, a questo proposito ha avviato un progetto che ha mosso i primi passi recuperando la configurazione finale dello Studio da vecchi magazzini RAI e mostrandola al pubblico nel Museo degli Strumenti Musicali del Castello Sforzesco a Milano. Questo progetto venne denominato D.R.E.A.M. acronimo di Digital Re-working/Re-appropriation of Electro-Acoustic Music e, con il sostegno dell'Unione Europea e la partecipazione di partner come l'Universitá di Padova, la Aalborg University Copenhagen, la Middlesex University, il comune di Milano e la RAI, poté crescere e stabilire nuovi obiettivi. Essendo partito dal conservare lo Studio nel suo complesso, poteva ora focalizzarsi in aspetti specifici come la conservazione dei suoni e dei macchinari. Si avvió quindi un processo di catalogazione delle opere composte nello Studio e delle persone che avevano fatto parte di questa realtá o l'avevano in qualche modo

vissuta; inoltre si cominció il riversamento in digitale dei nastri per fare in modo che le opere non venissero perse o danneggiate. Questo processo ha portato ad una postazione interattiva touch screen posta nel Museo degli Strumenti Musicali in cui il visitatore puó leggere informazioni sui compositori, ascoltare le opere e vedere fotografie e filmati dell'epoca. Un altro obiettivo era relativo alla conservazione dei suoni dei componenti dello Studio per fare in modo che le persone potessero avere un contatto diretto con quei suoni e che potessero sperimentare una filosofia compositiva riportata a tecniche molto semplici. É stata quindi sviluppata una postazione interattiva ricreando alcuni componenti dello Studio nella loro interfaccia e nel loro suono, utilizzando le nuove tecnologie nel campo dell'informatica musicale. Infine é stato pubblicato un libro, in versione italiana e in versione ampliata in inglese pubblicato da Ricordi, che racchiude i motivi del progetto di conservazione e altri aspetti riguardanti lo Studio come la storia, i componenti e il catalogo delle opere [3]. Il progetto non é ancora concluso; questi sono solo i primi obiettivi portati a termine.

Figura 3.23: *Lo studio nel 1956.*

Capitolo 4

Progetto

4.1 Introduzione

In questo capitolo verrá descritta in dettaglio la realizzazione del software in ambiente *Pure Data*. L'esposizione parte dagli obiettivi principali, per poi spostarsi sui dettagli implementativi delle varie patches e della loro organizzazione. Nel successivo capitolo saranno presentati gli sviluppi di questo codice all'interno del progetto D.R.E.A.M. che ha portato a costruire un'installazione multimediale interattiva basata sullo Studio di Fonologia RAI all'interno del Museo degli strumenti musicali nel Castello Sforzesco di Milano.

4.2 Obiettivi

Il lavoro svolto si é posto come fine la simulazione ideale di alcuni componenti dello Studio e la loro interazione utilizzando gli oggetti nativi del linguaggio *Pure Data*. Un virtual analog dello Studio, inteso come simulazione funzionale, non circuitale, dello strumento nel suo complesso. La simulazione ha voluto ricreare i componenti originali dello Studio nel periodo dei primi anni di attivitá, sono stati quindi presi in considerazione i componenti costruiti da Lietti in quanto essi rappresentano un patrimonio da salvare, a differenza dei componenti commerciali introdotti nello Studio al momento del rinnovo del '68. Ogni scelta durante questo processo di modellazione é stata presa considerando il progetto nella sua completezza, ossia pensando agli sviluppi futuri che questo software avrebbe dovuto subire.

4.3 Implementazione

4.3.1 Scelta componenti

Il processo d'implementazione é partito dal capire quali componenti modellare affinché si potesse godere di un'esperienza sonora semplice nell'utilizzo ma il piú possibile vicina a quella originale,

pur utilizzando un numero limitato di dispositivi[1]. Si é partiti dagli oscillatori, le basi per la generazione del suono, e dallo scegliere tre oscillatori sui nove disponibili. La scelta é ricaduta sul primo, secondo e nono oscillatore per avere la possibilitá di spaziare in range di frequenze molto diversi. Altro componente fondamentale per la generazione del suono é il generatore di rumore, fondamentale in quanto unico a produrre materiale sonoro non periodico e quindi alla base del pensiero di alcune opere composte nello Studio. Passando ai componenti dedicati alla trasformazione del suono, si é scelto il selezionatore d'ampiezza e il banco di filtri ad ottava.

4.3.2 Collegamento componenti

Il secondo passo é stato quello di decidere come collegare i componenti che sono stati scelti per la modellazione. Per ragioni di semplicitá d'utilizzo, pensando allo scopo finale del progetto, si é deciso che i collegamenti fra i dispositivi fossero definitivi e non modificabili dall'utente, a differenza dei collegamenti mobili presenti nello Studio. Inoltre pensando ad un'installazione che possa essere utilizzabile anche da due persone contemporaneamente, si sono suddivisi i componenti in due catene audio separate. La prima composta da:

- oscillatori
- miscelatore

La seconda composta da:

- generatore di rumore
- banco di filtri ad ottava
- selezionatore d'ampiezza
- miscelatore

Nella prima catena audio, l'ordine dei componenti poteva essere uno solo: i tre oscillatori producono tre forme d'onda che vengono sommate dal mixer.

[1] Questo numero limitato di componenti deriva dagli sviluppi futuri dell'installazione interattiva e dai problemi di spazio nel Museo degli Strumenti Musicali di Milano.

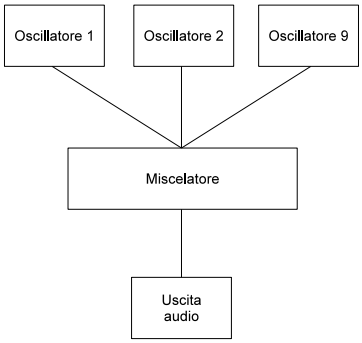


Figura 4.1: *Schema a blocchi della prima catena audio.*

Per quanto riguarda la seconda catena audio invece, erano possibili diverse configurazioni; é stata scelta quella ritenuta piú interessante dal punto di vista sonoro. Qui infatti il selezionatore d'ampiezza si trova come ultimo stadio della catena di trasformazione, ed é possibile scegliere la quantitá di rumore clean e di rumore filtrato da trasformare tramite il selezionatore. Il mixer viene quindi dotato di sette controlli pre-selezionatore e di uno post-selezionatore cosí organizzati:

- sei per il rumore filtrato a bande in ingresso al selezionatore
- uno per il rumore non filtrato in ingresso al selezionatore
- uno per controllare l'uscita del selezionatore

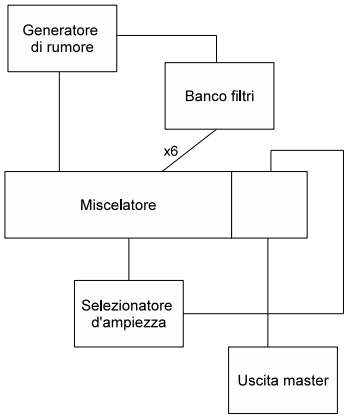


Figura 4.2: *Schema a blocchi della seconda catena audio.*

4.3.3 Simulazione interfacce

Dopo aver deciso come collegare i vari componenti, si é passati a studiarne l'aspetto e i vari controlli che essi presentavano. Facendo riferimento a [3], in particolare al capitolo scritto da Antonio Rodá, si sono estrapolate le funzioni di ogni singolo rotore ed é stata possibile la distinzione fra due tipi diversi di controlli: discreti e continui. Si parla di controlli discreti riferendosi a rotori che possono muoversi solamente in determinate posizioni (esempio per scegliere un numero intero tra 1 e 6); si parla invece di controlli continui per quei rotori che sono liberi di muoversi da un minimo ad un massimo in maniera continua, senza scatti. Altri controlli presenti in alcuni componenti erano gli switch, ossia dei comandi ad interruttore che possono assumere due valori, rappresentando per esempio la funzione di acceso/spento (ON/OFF) di un dispositivo oppure altre funzioni a seconda del componente. Oltre a questi controlli a rotore e ad interruttore che sono stati visti, erano presenti anche dei controlli visivi, dei display, che mostravano per esempio la frequenza dell'onda generata da un oscillatore oppure la tensione in uscita da un componente.

Passiamo ora ad analizzare ogni singolo componente in dettaglio, a studiarne i controlli e a ricrearli opportunamente nel linguaggio Puredata.

Oscillatori Gli oscillatori, come é giá stato detto nel capitolo precedente, erano dotati di due controlli rotativi: quello a sinistra é un controllo discreto in cui é possibile selezionare una fra i sei range a disposizione, quello a destra é un controllo continuo che permette di selezionare la frequenza. É inoltre presente un display che mostra la frequenza della sinusoide creata. Altri dettagli che si possono notare nella foto, sono quattro connettori per uscite ausiliarie; questi non sono stati presi in considerazione nel modello. Passando all'implementazione, per il controllo rotativo é stato scelto l'oggetto *knob* del linguaggio Pd, oggetto che di default puó assumere valori continui tra 0 e 127. Questo controllo si é quindi rivelato adeguato per il controllo rotativo continuo cambiando per semplicitá d'utilizzo il range di valori tra 0 e 1; per il controllo discreto invece, si é potuto comunque utilizzare lo stesso oggetto, cambiando il range fra 0 e 5 e trattando i numeri in uscita dal *knob* tramite un oggetto *int* che filtra i valori eliminandone i numeri decimali, restituendo cioé un intero compreso tra 0 e 5. In questo modo il controllo si comporta ancora in maniera continua nella variazione della posizione, ma risponde in maniera adeguata nel cambio discreto di range. Probabilmente l'oggetto piú adeguato in questa prima fase del progetto sarebbe stato un *select* che permette di scegliere fra un determinato numero di valori (esempio 0,1,2,3,4,5). Non é stato utilizzato questo oggetto prima di tutto per una scelta grafica che da parte mia voleva rappresentare i componenti anche nella loro interfaccia (entro i limiti imposti dal linguaggio), non solo nella loro funzione, e successivamente per implementazioni di funzioni che verranno presentate in seguito. Per quanto riguarda lo switch di accensione/spegnimento dell'oscillatore, é stato utilizzato l'oggetto *toggle* che puó assumere solamente due valori: 0 oppure 1. Per ultimo vediamo il display che nel componente reale é composto da un insieme di sei scale, una per ogni range, e da un ago che indica il valore di frequenza. Nella simulazione é stato utilizzato un semplice oggetto *number box* che ha la funzione di mostrare numericamente il suo input.

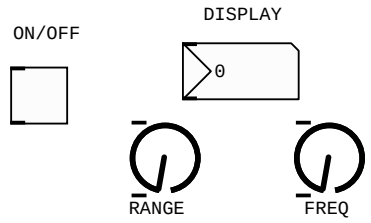


Figura 4.3: *Interfaccia oscillatore.*

Primo mixer É formato da tre rotori continui che sono stati modellati da *knob*, come spiegato nel paragrafo precedente.

Figura 4.4: *Interfaccia mixer della prima catena audio.*

Generatore di rumore É dotato di un controllo di accensione modellato da un *toggle*, di un rotore a due valori (con funzione sconosciuta) modellato da un altro *switch*, due display che misurano il valore in uscita in decibel e millivolt modellati da due *number box*, e due rotori continui dedicati al volume di uscita e al volume d'ascolto dell'uscita ausiliaria che non é stata implementata nel modello.

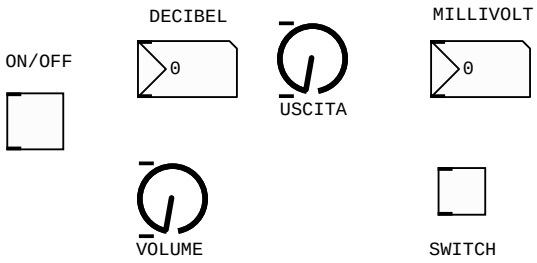


Figura 4.5: *Interfaccia del generatore di rumore.*

Banco di filtri d'ottava Essendo i filtri pre-impostati, si é deciso di includerli senza implementare l'interfaccia grafica in quanto non necessaria. Nel componente reale infatti non erano inclusi controlli, era presente solo una tabella con i valori relativi ai sei filtri.

Selezionatore di ampiezza Simile all'interfaccia dell'oscillatore, é dotata di un controllo di accensione modellato da un *toggle*, due rotori continui (con valori da 0 a 1) e un display modellati come negli altri componenti e una lampadina che, presentando solo due valori (accesa o spenta) é stata modellata da un semplice *toggle*.

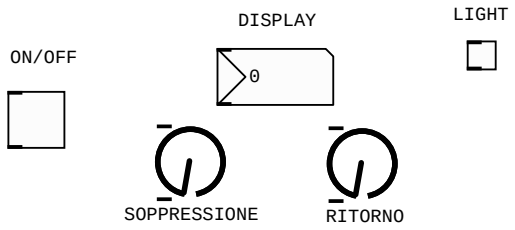


Figura 4.6: *Interfaccia del selezionatore.*

Secondo mixer Formato da otto rotori continui che sono stati modellati da *knob*, come spiegato precedentemente.

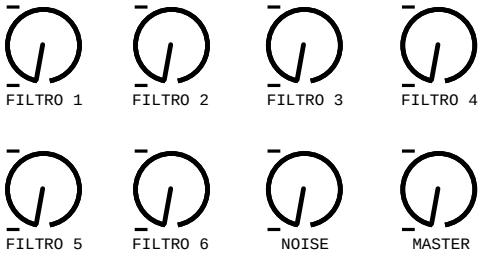


Figura 4.7: *Interfaccia mixer della seconda catena audio.*

4.3.4 Simulazione idealistica dei dispositivi

Dopo aver modellato le interfacce, é ora possibile passare alla simulazione idealistica dei componenti. Partendo dai controlli presentati sopra, verranno sviluppate singole patch per ogni componente in modo da testarne il corretto funzionamento.

Oscillatori Gli oscillatori producevano onde sinusoidali probabilmente non perfette, includendo alcune imperfezioni. Essendo questa una modelizzazione ideale, si possono trascurare le imperfezioni e si puó assumere che le onde generate dagli oscillatori siano delle sinusoidi perfette. Nel modello quindi potró utilizzare l'oggetto Pd *osc*$\sim$ come generatore di forme d'onda seno. Questo oggetto sará la parte fondamentale del modello; dopo aver deciso per il suo utilizzo, quello che rimane da fare é ricreare il comportamento dell'oscillatore, ossia l'accensione, la

divisione delle frequenze generabili in range, la selezione della frequenza tramite rotore e la visualizzazione della frequenza nel display. Sfruttando il lavoro svolto precedentemente, si é creata una patch con l'interfaccia dell'oscillatore, si é aggiunto l'oggetto *osc*$\sim$ per la generazione del suono. Come si é giá visto, le frequenze generate dagli oscillatori erano divise in range; i valori dei range sono riportati in tabella.

		range					
		1	2	3	4	5	6
osc	1	40-60	60-90	90-120	120-180	180-270	270-400
osc	2	200-240	240-300	300-370	370-450	450-720	570-720
osc	3	340-410	410-490	490-580	580-700	700-820	820-1000
osc	4	720-800	800-890	890-950	950-1100	1100-1230	1230-1370
osc	5	1020-1080	1080-1150	1150-1230	1230-1320	1320-1400	1400-1500
osc	6	1300-1400	1400-1550	1550-1700	1700-1850	1850-2050	2050-2250
osc	7	2000-2300	2300-2600	2600-2900	2900-3300	3300-3700	3700-4200
osc	8	4200-4900	4900-5700	5700-6500	6500-7500	7500-8700	8700-10100
osc	9	40-90	90-220	220-600	600-1400	1400-3600	3600-9000

Tabella 4.1: *Range degli oscillatori. Dati tratti da [3]*

Dovendo modellare solo gli oscillatori numeri uno, due e nove, ci riferiamo ai valori relativi solo a questi. Rivediamo i valori di output dei *knob* dell'interfaccia: quello di sinistra seguito da un *int* produce valori interi da 0 a 5, quello di destra valori continui da 0 a 1. Ponendo un *select* che filtra i numeri interi in ingresso dal *knob* sinistro dirigendoli su sei output distinti, collegandoli ognuno ad un *message* contenente il fondo scala del range corrispondente, e collegando questi all'*osc*$\sim$, é possibile muovendo il *knob* in questione generare una delle sei onde con frequenza pari al fondo scala dei vari range (vedi figura).

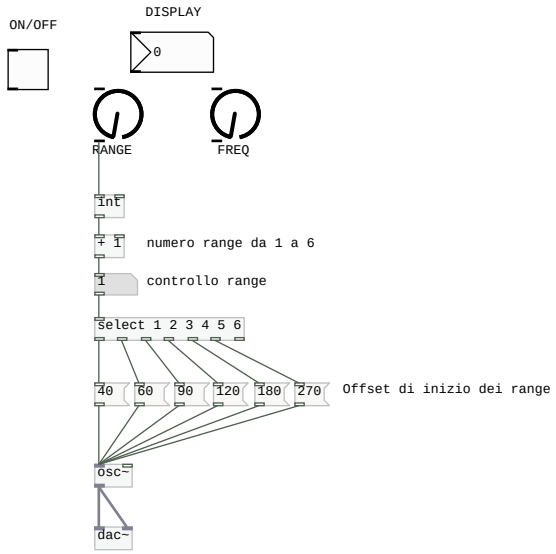


Figura 4.8: *Inizio patch primo oscillatore.*

Passiamo a collegare il *knob* destro: per fare in modo che questo si muova all'interno del range selezionato dal *knob* sinistro, si puó sommare il fondo scala del range con il valore del *knob* destro moltiplicato opportunamente in modo che il valore massimo raggiunto dalla somma sia il limite superiore della scala. Nel momento in cui includiamo somme e moltiplicazioni, diventa obbligatorio pensare alle giuste temporizzazioni (tramite *trigger* e *bang*) per fare in modo che la moltiplicazione sia eseguita prima della somma e che ogni modifica nella posizione dei due *knob* aggiorni il valore risultante. Si é poi passati a ricreare l'accensione e lo spegnimento del componente tramite un *toggle* utilizzato in una moltiplicazione audio: questo oggetto restituisce in output valore 0 quando l'oscillatore é spento, quindi il segnale viene moltiplicato per 0 annullandone l'ampiezza; restituisce invece valore 1 quando l'oscillatore é acceso, quindi il segnale viene moltiplicato per 1 non modificandole l'ampiezza. Al *toggle* é stata tolta l'istantaneitá tramite un *line*$\sim$ che esegue un breve fade-in quando l'oscillatore viene acceso e un fade-out quando viene spento, in modo da evitare dei click[2]. L'ultimo aspetto che manca da riprodurre é la visualizzazione nel display della frequenza dell'onda generata. Nel componente reale, la frequenza veniva selezionata tramite i rotori, poi veniva misurata dopo la generazione dell'onda e il valore veniva passato al display. Colleghiamo quindi il valore che viene passato in input all'oggetto *osc*$\sim$ anche nell'input del *number box* che simula il display. Inoltre per una simulazione piú accurata, pensando che un componente analogico spento non produce segnale, serve prestare attenzione e fare in modo che, nel momento in cui lo switch dell'oscillatore é in posizione OFF, il display visualizzi il valore zero. A questo scopo é possibile aggiungere un oggetto *expr* che analizzi il valore del *toggle* che modella lo switch, e a seconda del suo valore decide se mandare al display la frequenza data dalla somma precedentemente spiegata oppure zero. Un altro dettaglio é stato incluso nel modello per renderlo un po' simile al componente reale. Analizzando le simulazioni dei circuiti originali in SPICE[3] eseguite dal dottor Marchetto[4] si é notato che nel momento in cui l'oscillatore risponde ad un comando di cambio range, questo non lo fa istantaneamente ma per alcuni millisecondi produce un segnale nullo e poi cambia range. Nel modello che stiamo sviluppando, come si é visto, il *knob* sinistro non é effettivamente a valori discreti, ma il movimento é continuo; quindi si é deciso di utilizzare la soglia fra due range[5] come soglia in cui viene generato segnale nullo. In questo modo, cambiando range si passerá per una posizione che comanda all'oscillatore di generare segnale nullo prima di cambiare la frequenza dell'onda sinusoidale. Vista questa modifica, si é pensato di modificare il range di valori assunti dal *knob* sinistro in questo modo: dato che l'output del *knob* é seguito da un oggetto *int* che restituisce il numero eliminandone i decimali, é possibile far variare l'output del *knob* da 0 a 5.9 facendo in modo che:

- un output nell'intervallo [0, 0.9[[6] selezioni il primo range
- un output nell'intervallo]1.0, 1.9[[7] selezioni il secondo range

[2] Effetto dovuto all'interruzione dell'onda in una posizione ad ampiezza diversa dallo zero
[3] *Simulation Program with Integrated Circuit Emphasis*,software di simulazione circuitale dei circuiti analogici
[4] Ricercatore dell'universitá di Padova coinvolto nel progetto D.R.E.A.M.
[5] Le soglie di cui si parla sono quindi tra 0.9 e 1.0, tra 1.9 e 2.0, tra 2.9 e 3.0, tra 3.9 e 4.0, tra 4.9 e 5.0
[6] Intervallo con estremo inferiore compreso ed estremo superiore escluso
[7] Intervallo con estremi esclusi

- un output nell'intervallo]2.0, 2.9[selezioni il terzo range
- un output nell'intervallo]3.0, 3.9[selezioni il quarto range
- un output nell'intervallo]4.0, 4.9[selezioni il quinto range
- un output nell'intervallo]5.0, 5.9][8] selezioni il sesto range

e che gli intervalli fra i range producano un'onda a frequenza nulla. Qui sotto é possibile vedere la patch complessiva del primo oscillatore.

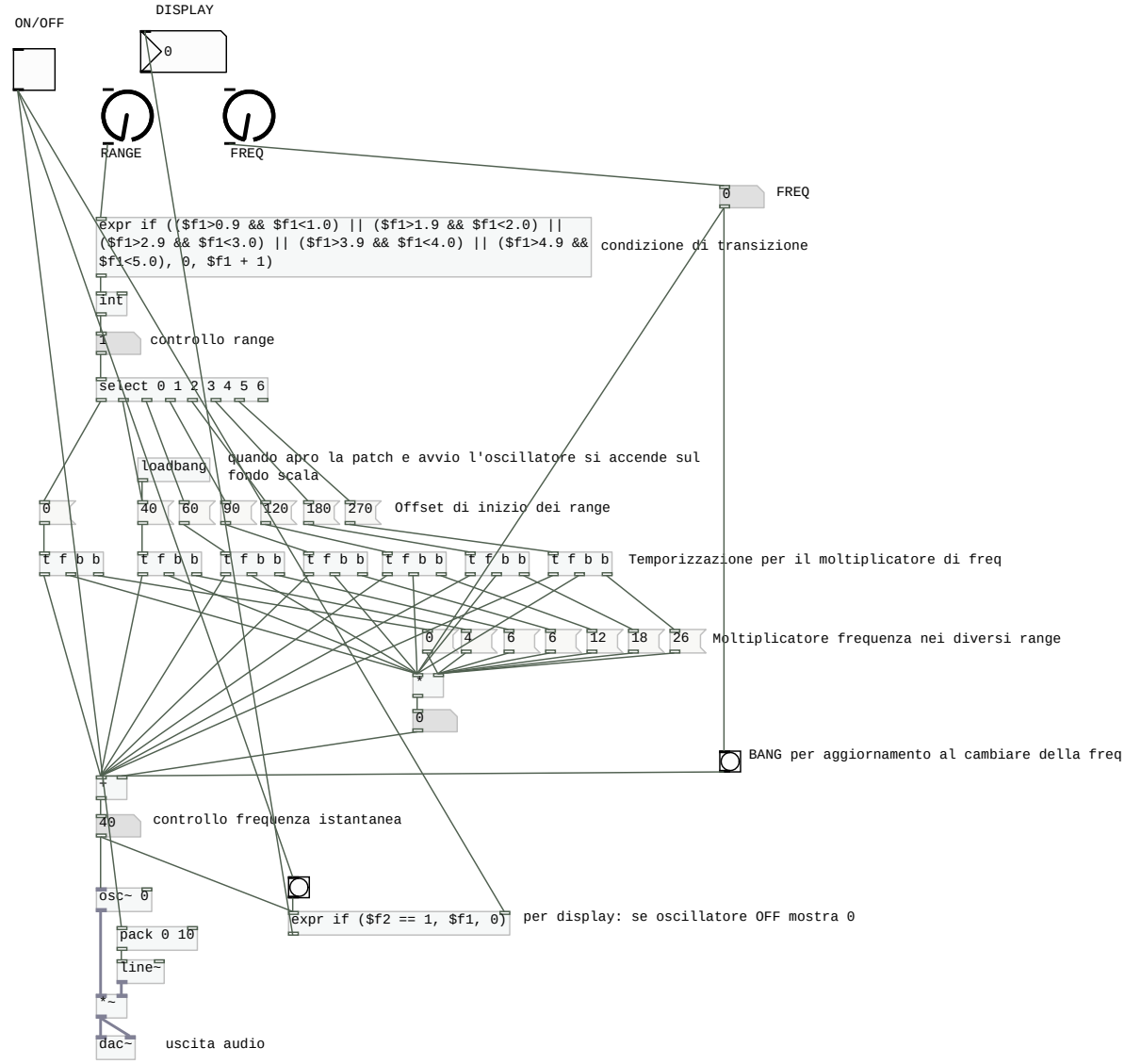


Figura 4.9: *Patch primo oscillatore.*

[8]Intervallo con estremo inferiore escluso ed estremo superiore compreso

Primo mixer Il mixer, come si é giá visto, ha la funzione di miscelare le onde sinusoidali prodotte dai tre oscillatori. L'operatore fondamentale per sommare due segnali audio in Pd é il +∼; si eseguirá quindi la somma dei segnali provenienti da due oscillatori, a cui poi si sommerá il segnale proveniente dal terzo[9]. Per fare in modo che i *knob* controllino i volumi di ogni oscillatore, si utilizza l'operatore *∼ moltiplicando il segnale sinusoidale per il valore di output del *knob* corrispondente. Al valore uscente del *knob* si é applicato un *line*∼ per fare in modo che l'ampiezza dell'onda vari non istantaneamente, evitando click. Prima di mandare il segnale mixato all'uscita, si é applicato un blocco limiter, in modo che il segnale non vada in distorsione superando il valore massimo di ampiezza. Nella patch il segnale proveniente dagli oscillatori é stato indicato con tre *inlet*∼; per testare il corretto funzionamento della patch, questi si possono sostituire con tre semplici oggetti *osc*∼ impostati a tre diverse frequenze.

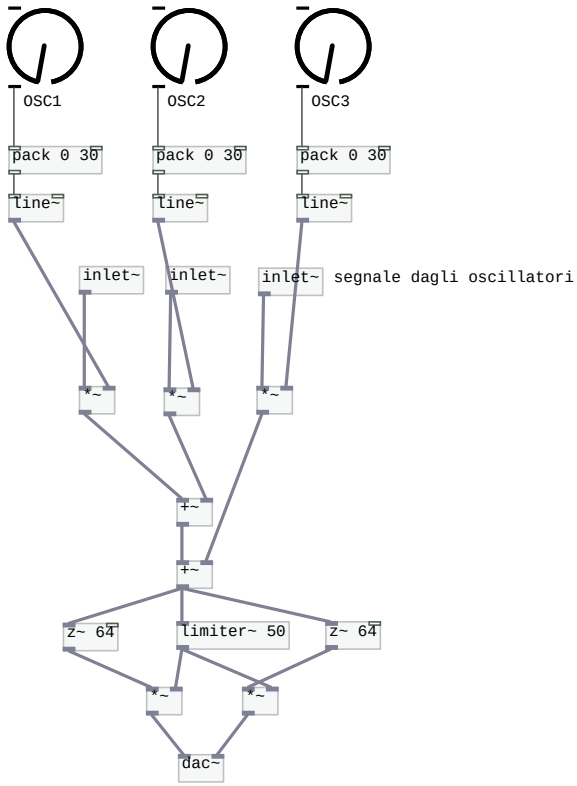


Figura 4.10: *Patch primo mixer.*

Generatore di rumore Il generatore di rumore produce rumore bianco; l'oggetto in Pd che svolge questa funzione é il *noise*∼. Partendo dall'interfaccia giá modellata, bisogna aggiungere il comportamento di accensione e spegnimento nel modo giá visto negli oscillatori (operatore *∼ sfruttando il valore del *toggle* che modella lo switch) e il controllo di volume come giá visto

[9]Nel progetto finale il mixer é stato dotato anche di uno switch per l'accensione.

nel mixer (operatore *∼ moltiplicando il segnale per l'output del *knob*). Per quanto riguarda invece il doppio display relativo alle misure del segnale in uscita dal componente in decibel e in milliVolt, si sono utilizzati gli oggetti *env∼* e *dbtorms∼*. Come si vede dalla patch, il *knob* volume relativo ad un pre-ascolto in cuffia, e lo switch con funzione non ancora determinata, non sono stati implementati.

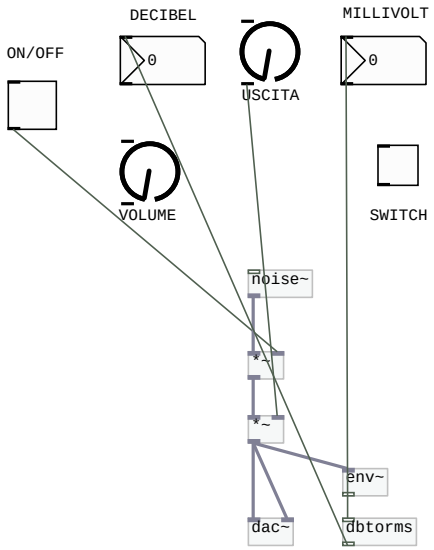


Figura 4.11: *Patch generatore di rumore.*

Filtri d'ottava Il banco di filtri d'ottava non é dotato di interfaccia in quanto, come giá detto, i filtri sono impostati su frequenze determinate (vedi tabella) e non sono modificabili.

	1	2	3	4	5	6
Hz	200-400	400-800	800-1600	1600-3200	3200-6400	6400-12800

Tabella 4.2: *Frequenze dei filtri. Dati tratti da [3]*

Per modellare i filtri passabanda, si é utilizzato l'oggetto Pd *bp∼* impostando opportuni valori di frequenza centrale e fattore di merito (Q). L'*inlet∼* rappresenta il segnale proveniente dal generatore di rumore; per testare il funzionamento della patch si puó sostituire con un semplice oggetto *noise∼* e collegare un filtro alla volta all'uscita audio.

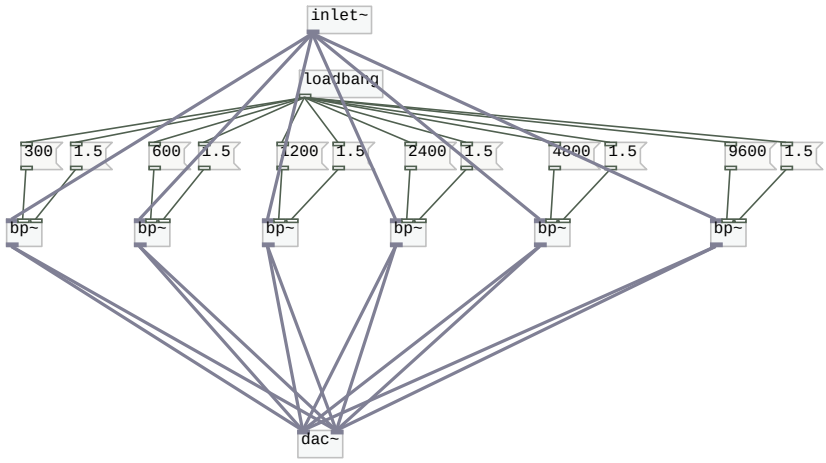


Figura 4.12: *Patch filtri d'ottava.*

Secondo mixer Il mixer della seconda catena audio é modellabile come il mixer della prima; cambiano solamente il numero di *knob* e di *inlet*∼, ossia il numero di segnali: i primi sei provengono dai filtri, il settimo direttamente dal generatore di rumore e l'ultimo dal selezionatore d'ampiezza. Dato che il selezionatore d'ampiezza e il mixer non possono essere collegati come in figura

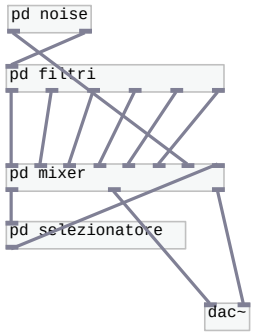


Figura 4.13: *Configurazione con DSP loop.*

altrimenti danno vita ad un DSP loop[10], si é sfruttato un costrutto *send*∼ - *receive*∼, che al suo interno comprende un ritardo che permette di risolvere il problema. Per testare questa patch é possibile sostituire gli *inlet*∼ con degli oggetti *osc*∼ impostati su frequenze diverse e l'*outlet*∼ con un *dac*∼; il blocco *send*∼ - *receive*∼ invece verrá testato nel momento in cui le patch verranno unificate.

[10]Configurazione in cui l'output di un componente é collegato all'input di un componente che sta al di sopra di questo. Nel mondo analogico é possibile questa configurazione in quanto il segnale é pressoché immediato; Pd invece tratta i segnali in blocchi quindi non puó includere in un calcolo un segnale che non é ancora stato trattato.

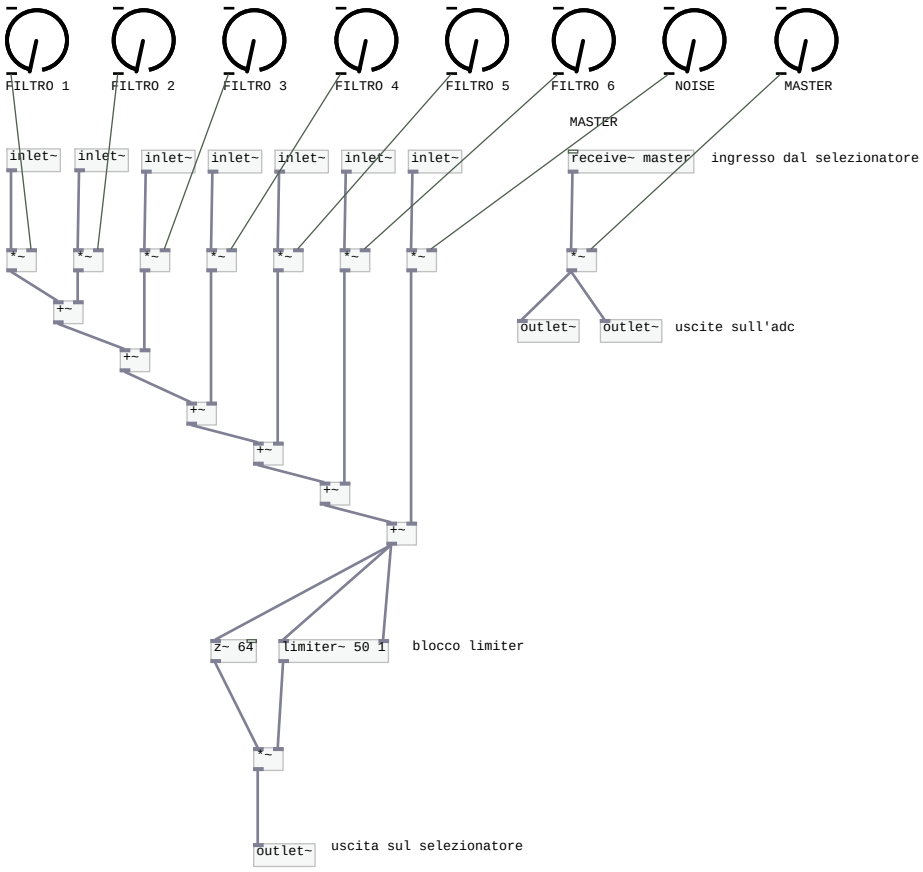


Figura 4.14: *Patch secondo mixer.*

Selezionatore d'ampiezza Come descrive Roberto Doati in un articolo sulla composizione "Scambi" di Henri Pousseur[11] il selezionatore d'ampiezza "a partire da un fenomeno sonoro complesso, seleziona, cioé lascia passare, solo quella porzione di segnale che supera una determinata ampiezza il cui valore (detto soglia) puó essere scelto a piacere"; l'affermazione trova conferma nelle simulazioni SPICE svolte [1].

[11] Roberto Doatii "Il caso filtrato" presente su "I quaderni della civica scuola di musica" N.21-22, dicembre 1992

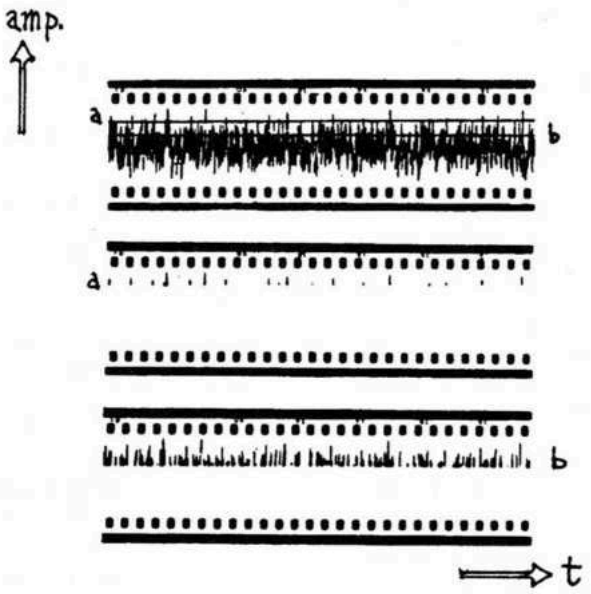


Figura 4.15: *Rumore bianco filtrato tramite il selezionatore d'ampiezza a due diverse soglie. Disegno tratto da Gravesaner Blätter n.13 VI 1959*

Essendo qui sviluppato un modello ideale e semplificato in alcuni aspetti, si é scelto di ricreare un comportamento diverso, ossia quello di un tipico noise gate: questo componente blocca il segnale che non supera la soglia impostata; detto in maniera equivalente, lascia passare il segnale quando questo supera la soglia impostata. La differenza fra il comportamento reale e il comportamento modellato sembra minima, ma non é cosí: il componente reale, come si vede in figura, si puó dire che tagli le forme d'onda passando all'output solo le "punte" (a seconda dell'impostazione del livello di soglia); il noise gate invece lascia passare tutta l'onda nel momento in cui questa ha ampiezza che supera la soglia. Per modellare il comportamento semplificato, l'operatore utilizzato é $>$. L'ampiezza di ogni campione del segnale viene confrontata con il livello di soglia impostato tramite il *knob* sinistro: se é maggiore la sua ampiezza non viene modificata e il campione viene riprodotto; se invece é minore della soglia la sua ampiezza viene azzerata in un intervallo di tempo impostato dal *knob* destro, ed in output avremmo segnale nullo. Il tempo impostato tramite il rotore destro viene detto "tempo di rilascio"ed é impostabile scegliendo fra due valori: 1ms oppure 10ms[12]. Passando allo switch di ON/OFF, analizzando le simulazioni SPICE, si é visto che quando il selezionatore d'ampiezza é spento, non lascia passare segnale. Questo é stato modellato come sopra utilizzando un *$\sim$ e sfruttando il valore di uscita del *toggle* che modella lo switch di accensione. Non conoscendo l'esatta funzione della lampadina nel selezionatore, si é scelto di utilizzarla come indicatore visivo per mostrare l'apertura e la chiusura del noise gate, ossia quando il selezionatore lascia passare il segnale oppure lo blocca[13]. L'ultimo dettaglio da implementare riguarda il display, che é giá stato modellato nell'interfaccia. Questo

[12]"Conservazione e fruizione della liuteria elettronica: il caso dello Studio di Fonologia della Rai di Milano" di Antonio Rodá

[13]A causa delle temporizzazioni della comunicazione fra i componenti hardware utilizzati nell'installazione

riporta la misurazione in decibel del segnale in uscita dal selezionatore ed é stato implementato misurando appunto l'uscita e collegandola al *number box* che simula il display.

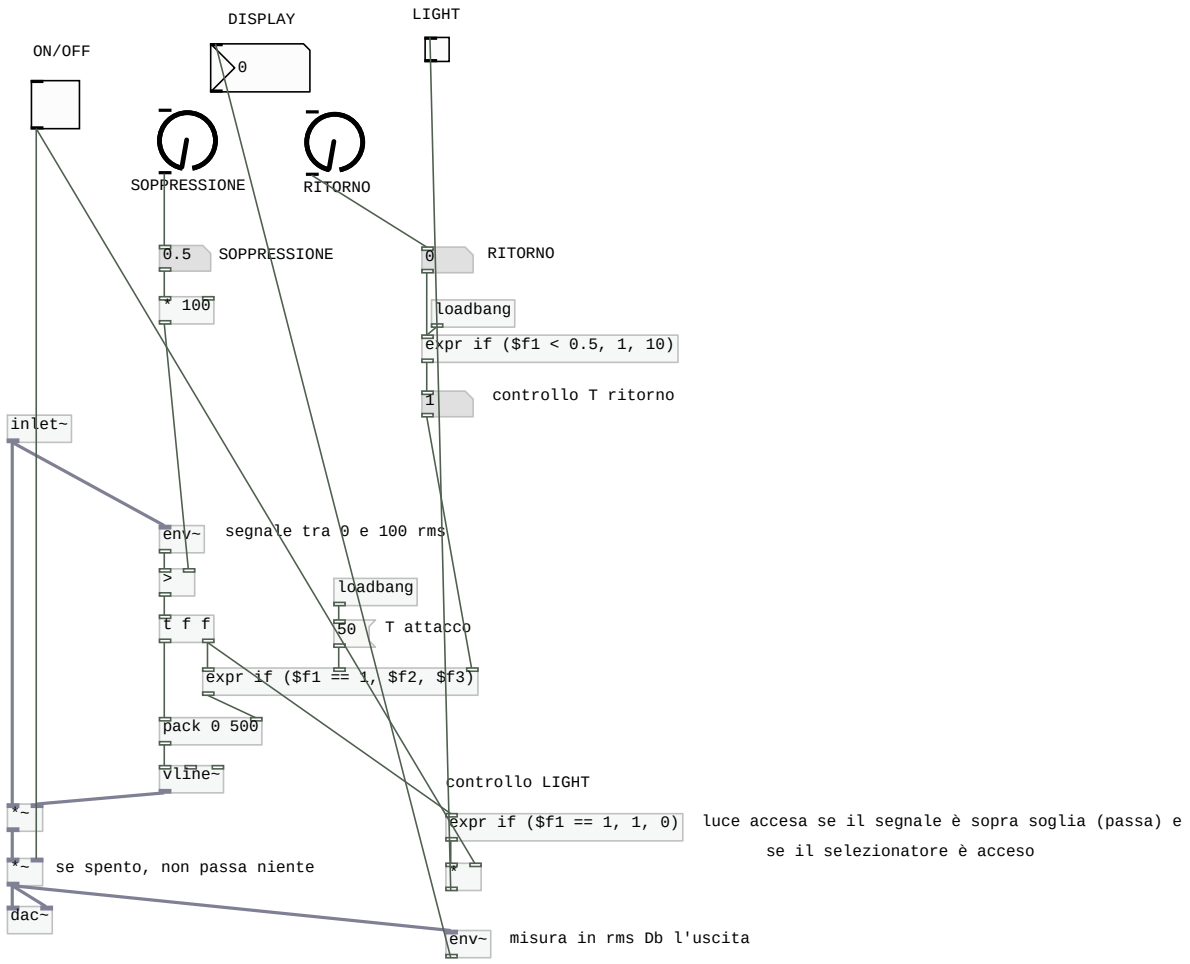


Figura 4.16: *Patch selezionatore d'ampiezza.*

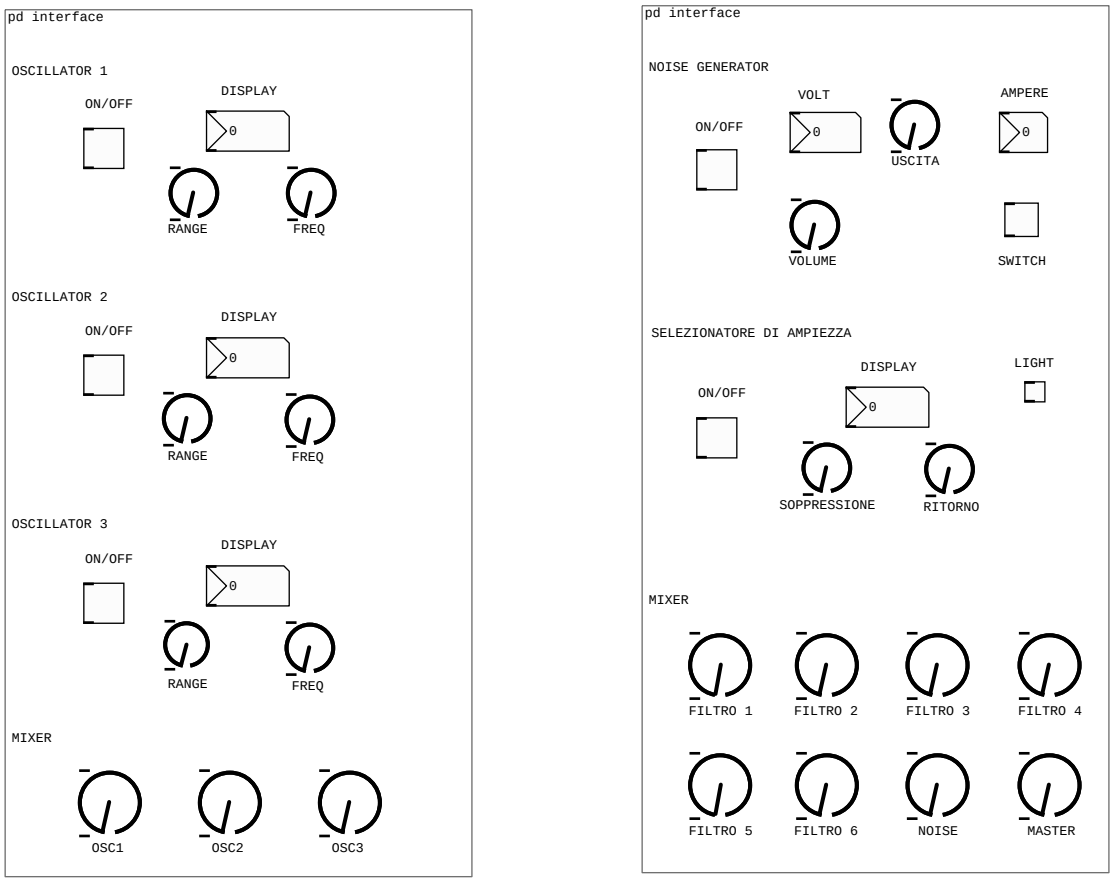
4.3.5 Organizzazione in patch e comunicazione

Dopo aver modellato le interfacce e i componenti in patch separate, si é passati a decidere un'organizzazione in patch che potesse dare dei benefici nel futuro del progetto, ossia quando, come verrá spiegato nel prossimo capitolo, le interfacce modellate saranno sostituite da interfacce tangibili. Si é scelta un'organizzazione in 4 patch:

- 2 patch divise contenenti i due rack di controlli
- 2 patch divise contenenti le due catene audio formate dai modelli dei componenti

interattiva, la lampadina é diventata un indicatore di accensione e spegnimento del selezionatore.

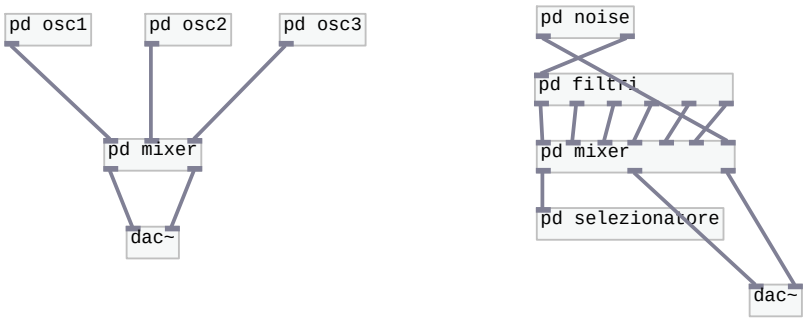
Le patch contenenti le interfacce sono state scollegate ed organizzate come in figura.



(a) (b)

Figura 4.17: *Le patch di interfacce (a) e (b).*

Le patch invece relative alla parte vera e propria di generazione e modifica del suono, sono state organizzate come giá mostrato in figura 4.1 e 4.2, utilizzando delle subpatch e adattandole opportunamente tramite *inlet*$\sim$ e *outlet*$\sim$ (vedi figura).



(a) (b)

Figura 4.18: *Le patch con catene audio (a) e (b).*

Avendo diviso i controlli e i componenti in patch diverse, si presenta il problema della comunicazione fra patch per fare in modo che tutto funzioni come quando ogni interfaccia era collegata agli oggetti da controllare. Nel capitolo 2 é stato presentato il protocollo OSC; é a questo che si ricorre quando si ha bisogno di stabilire una comunicazione fra patch diverse all'interno di Pd. Per prima cosa analizziamo la direzione dei messaggi: questi partono dai controlli sulle interfacce per arrivare alle patch dei componenti, inoltre altri messaggi percorrono la direzione inversa per visualizzare dei dati sui display oppure sulle lampadine. Partiamo dalle patch dei controlli e vediamo come si é preparata la comunicazione sfruttando questo protocollo. La prima cosa da fare é la preparazione dei messaggi da inviare, organizzandoli utilizzando i tag opportuni che permettano, una volta ricevuti, di capire a che componente sono destinati. Nella tabella seguente sono stati riassunti i tag utilizzati in due livelli: il primo livello indice il componente, il secondo livello indica il controllo specifico.

componente	tag primo livello	tag secondo livello
oscillatore 1	osc1	on range freq
oscillatore 2	osc2	on range freq
oscillatore 3	osc3	on range freq
generatore di rumore	noise	on volume uscita switch
selezionatore d'ampiezza	sel	on soppr ritorno
mixer	mixer	osc1 osc2 osc3 filtro1 filtro2 filtro3 filtro4 filtro5 filtro6 noise master

Tabella 4.3: *Tag utilizzati per la comunicazione*

I messaggi relativi ai controlli dei due mixer, come si vede in tabella, sono stati accorpati utilizzando lo stesso tag di primo livello: *mixer*. Dai tag di secondo livello é comunque possibile filtrare i messaggi ed instradarli al mixer corretto. L'oggetto principale su cui si basa il protocollo in Pd é l'oggetto *sendOSC* che, dopo aver instaurato una connessione UDP ad una porta del localhost (qui é stata utilizzata la porta 9000), invia sulla porta decisa i messaggi che riceve.

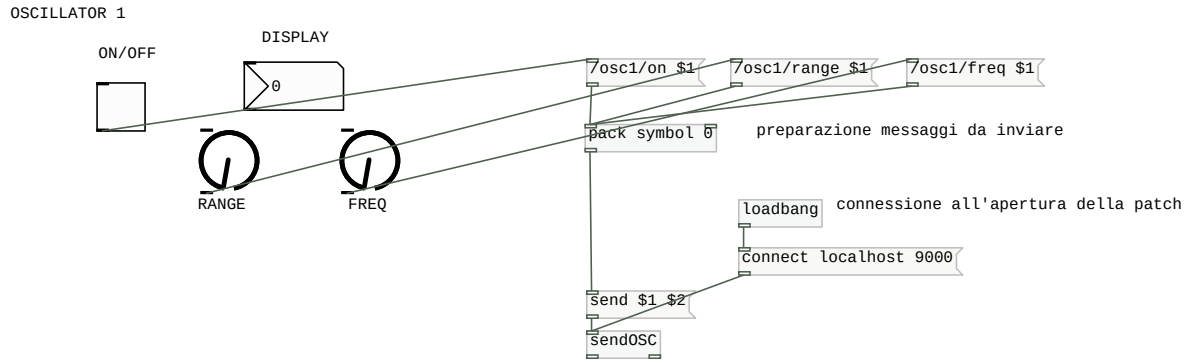


Figura 4.19: *Esempio preparazione messaggi per il primo oscillatore.*

L'altro oggetto fondamentale per il protocollo é il *dumpOSC* che rimane in attesa dei messaggi su una porta e li inoltra nel suo *outlet*. Dato che l'oggetto *dumpOSC* non é utilizzabile in piú copie associate alla stessa porta di comunicazione, si é utilizzato un costrutto *send - receive* per fare in modo che questo potesse essere richiamato all'interno di ogni patch o subpatch quante volte fosse necessario per mantenere il tutto facilmente comprensibile. Nelle patch con i componenti, é quindi presente un *dumpOSC* che riceve i messaggi in arrivo e un *send* che inoltra i messaggi a tutti i componenti. All'interno di ogni subpatch che modella un componente, é presente un *receive* che riceve i messaggi inoltrati e un oggetto *OSCroute* che filtra i messaggi utilizzando i tag; ogni *OSCroute* utilizza i tag di un livello. In questo caso sono stati utilizzati due di questi oggetti in cascata per filtrare i messaggi prima per componente e poi per controllo (vedi figura).

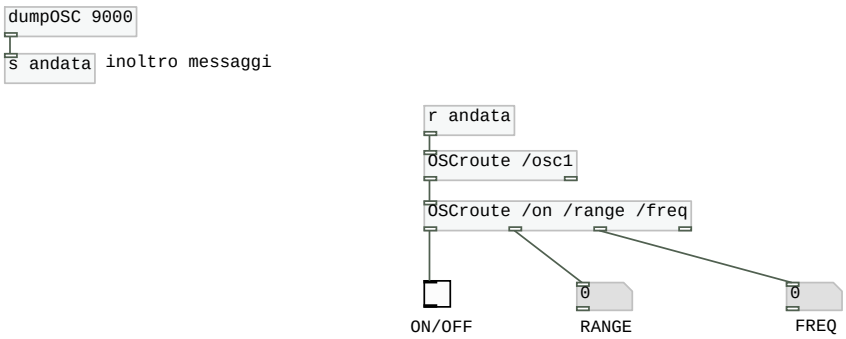


Figura 4.20: *Esempio ricezione e inoltro messaggi per il primo oscillatore.*

I messaggi invece che percorrono la direzione inversa, ossia dalle patch dei componenti alle patch di controllo, utilizzano una diversa porta ma sono stati trattati esattamente allo stesso modo: é stata instaurata una connessione sulla porta 8000 del localhost, é stato utilizzato un oggetto *sendOSC*, sono stati preparati i messaggi organizzandoli tramite i tag (vedi tabella), si é utilizzato un oggetto *dumpOSC*, un costrutto *send-receive* per inoltrare i messaggi e infine si sono utilizzati degli oggetti *OSCroute* per filtrare i messaggi e farli giungere a destinazione.

componente	tag primo livello	tag secondo livello
oscillatore 1	osc1	display
oscillatore 2	osc2	display
oscillatore 3	osc3	display
generatore di rumore	noise	db mv
selezionatore d'ampiezza	sel	display light

Tabella 4.4: *Tag utilizzati per la comunicazione di ritorno*

4.4 Risultati

Come giá presentato nel paragrafo dedicato agli obiettivi, lo scopo di questo lavoro era quello di creare una simulazione idealistica di parte dello Studio. Questa simulazione doveva essere il primo approccio alla costruzione di un modello funzionante che potesse essere utilizzato nel futuro del progetto come base di partenza funzionante in cui sostituire i modelli ideali con modelli fisici accurati e i modelli dei controlli con delle interfacce tangibili. Questo sará l'argomento del prossimo capitolo.

Capitolo 5

Un'installazione interattiva

5.1 Introduzione

Quest'ultimo capitolo presenta le modifiche e le estensioni che il software mostrato nel capitolo precedente ha subito nella conclusione del progetto D.R.E.A.M. Il capitolo é suddiviso in tre sezioni piú una: interfacce tangibili, virtual analog, comunicazione e presentazione dell'installazione. Il seguente lavoro viene presentato brevemente in quanto é stato svolto da persone del Centro di Sonologia Computazionale dell'Universitá di Padova e della Aalborg University Copenhagen; il sottoscritto ha partecipato solo a parte delle modifiche del software e al suo collaudo.

5.2 Interfacce tangibili

Dato lo scopo del progetto di creare una postazione interattiva in cui l'utente potesse vivere l'esperienza del creare musica come se fosse nello Studio di Fonologia, serviva ricreare delle interfacce il piú possibile simili a quelle originali che sostituissero i modelli d'interfaccia che erano stati inclusi nelle patch *Pure Data*. Il software sviluppato di sicuro puó anche essere controllato direttamente tramite la GUI sviluppata oppure sarebbe stato possibile controllarlo, con le dovute modifiche, con un controller MIDI, ma in questo modo si sarebbe persa gran parte dell'esperienza originale che invece si crede sia fondamentale preservare essendo importantissima l'interazione fisica con lo strumento.

Le interfacce originali possono essere viste come banali rispetto ai controller che in questi ultimi anni vengono sviluppati, ma se queste vengono considerate nella loro unione come uno strumento elettronico, allora diventano particolarmente interessanti dal punto di vista dei controlli e del design. Erano infatti strutturate in un modo che rendesse possibile instradare il suono in maniera libera fra i vari componenti, e questo era fondamentale nella fase di composizione.

Una differenza che non puó essere eliminata riguarda il funzionamento di questo modello che si diversifica dall'originale avendo delle interfacce che devono essere collegate al computer che genera e modifica il suono, diversamente dalle interfacce originali che erano collegate ai circuiti elettrici in cui il suono veniva creato. L'approccio applicato é il tipico approccio utilizzato in

questi anni, in cui un'interfaccia MIDI controlla uno strumento virtuale in esecuzione in un computer.

Non essendo presenti progetti riguardo i pannelli originali, le informazioni sono state estrapolate da quattro fonti:

- fotografie
- annotazioni dei compositori
- testimonianze dirette di persone che sono entrate in contatto con lo Studio in quegli anni
- parti dello Studio sopravvissute

Ogni pannello era stato costruito in modo che fosse possibile spostarlo nei telai a seconda dei componenti utilizzati. Questo significa che i pannelli originali avevano tutti la stessa larghezza, altezza multipla di una certa unitá standard e fori in determinate posizioni che coincidessero con i fori dell'armadio in cui erano posizionati.

Sono quindi stati creati dei pannelli a rack standard, come quelli originali, e si sono cercati dei componenti (rotori, switch, display) simili. Avendo deciso di implementare solo tre dei nove oscillatori, il pannello mixer in cui apparivano i nove rotori relativi ai nove oscillatori, é stato costruito con soli tre rotori; gli altri sono stati comunque indicati nel pannello per renderlo simile all'originale. Infine é stata decisa la posizione delle interfacce: tenendo conto che i componenti nello Studio non avevano una posizione fissa in quanto potevano essere spostati a seconda delle esigenze, si é cercata la configurazione che appare piú spesso nelle fotografie. Le interfacce sono quindi state organizzate in questo modo: un primo armadio rack contenente gli oscillatore e il mixer, un secondo armadio contenente il noise generator, i filtri, il selezionatore d'ampiezza e il mixer (vedi figura).

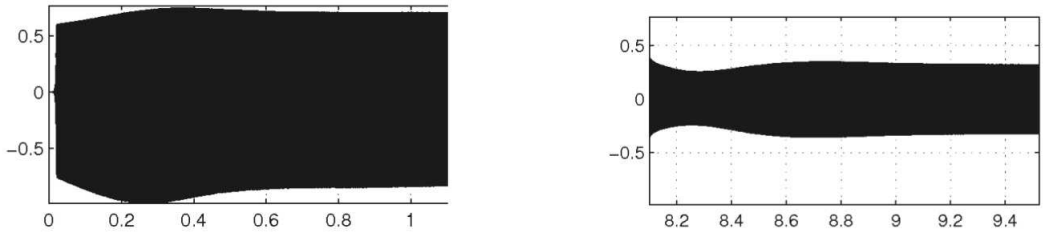
Figura 5.1: *Interfacce. Tratta da [3]*

Per i dettagli rimando al capitolo "Novel interfaces for controlling musical expression: historical overview and current directions" di S.Serafin e S.Gelineck [3].

5.3 Virtual analog

I modelli ideali di alcuni componenti sono stati rimpiazzati da dei modelli *virtual analog* scritti in C e inglobati nel software come oggetti *external*. In particolare sono stati modellati i tre oscillatori e il selezionatore d'ampiezza. Utilizzando l'approccio che parte dagli schemi elettrici dei componenti é possibile sormontare l'apparente problema della "non esistenza" degli oscillatori. Essendo stati questi costruiti da Lietti utilizzando materiale recuperato nei magazzini Rai, nel momento in cui lo Studio chiuse, questi oscillatori vennero smontati e probabilmente i componenti vennero utilizzati per altri scopi. Fortunatamente gli schemi elettrici sono stati conservati e questo ha permesso di studiarli e costruire delle simulazioni SPICE dei circuiti per comprenderne il funzionamento.

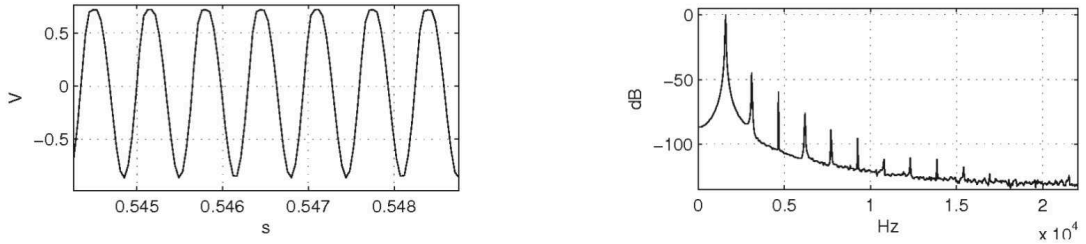
Tralasciando l'analisi del circuito, é interessante invece riportare alcuni comportamenti non-ideali riscontrati nelle simulazioni, e che tramite la sintesi a modelli fisici sono stati conservati anche nell'oggetto modellato. La figura 5.2 mostra due transienti non-ideali: il primo relativo all'inviluppo di accensione dell'oscillatore, il secondo relativo al cambio di range di frequenze.



(a) (b)

Figura 5.2: *Transienti all'accensione (a) e al cambio di range (b). Tratta da [3]*

Un altro comportamento notato é la presenza di distorsione nell'onda sinusoidale ad alti valori di ampiezza; é possibile notarla anche nell'analisi spettrale in quanto sono presenti diverse armoniche oltre a quella fondamentale, diversamente da un'onda sinusoidale ideale che come visto nel primo capitolo presenta solo un picco di informazione nella fondamentale.



(a) (b)

Figura 5.3: *Onda seno con distorsione (a) e spettro (b). Tratta da [3]*

Per quanto riguarda il selezionatore d'ampiezza invece, questo esiste ancora e fa parte dell'esibizione al Museo degli Strumenti Musicali nel Castello Sforzesco a Milano. É stato quindi possibile studiarlo da vicino per verificare l'attendibilitá degli schemi circuitali e per integrarli con valori di elementi che non erano indicati. Al selezionatore é stato applicato lo stesso procedimento visto per gli oscillatori.

Per i dettagli rimando al capitolo "Virtual analogue instruments: an approach to active preservation of the Studio di Fonologia Musicale" di F.Avanzini e S.Canazza [3].

5.4 Comunicazione

Nel software sviluppato nel capitolo precedente, la comunicazione fra interfacce e patch di generazione del suono era stata affidata al protocollo OSC. Ora che le interfacce modellate sono

state rimpiazzate da interfacce tangibili e che le patch sono state aggiornate con dei componenti *virtual analog*, quello che manca é la comunicazione tra controlli e computer. Visti i costi molto contenuti e la sua natura open-source, si é utilizzato Arduino.

Arduino é un hardware sviluppato presso l'Interaction Design Institute di Ivrea[1]. Puó essere utilizzato per progetti stand-alone oppure puó interagire con software come Max/MSP, Pure Data, SuperCollider, Adobe Flash. Si basa su un circuito stampato che integra un microcontrollore dotato di input/output, un regolatore di tensione e un'interfaccia USB per la comunicazione con il computer. É un progetto open-source in quanto le informazioni sull'hardware e i progetti sono disponibili liberamente ed é quindi possibile anche costruirsi un clone di Arduino o una versione modificata a proprio piacimento. All'hardware é affiancato un ambiente di sviluppo integrato (IDE) multipiattaforma basato su un linguaggio di programmazione semplice e derivato da C e C++, chiamato *Wiring*. Per scrivere un programma eseguibile, l'utente deve solo definire due funzioni:

- setup(), é la funzione utilizzata per i settaggi iniziali
- loop(), é la funzione invocata ripetutamente durante l'esecuzione del programma

Anche se l'utilizzatore scrive i propri programmi utilizzando questo linguaggio, questi vengono in realtá caricati sull'Arduino in linguaggio C/C++. Infatti quando si seleziona "Upload to I/O board", una copia del codice viene trascritta in un file temporaneo aggiungendo automaticamente un *header* e l'implementazione di una funzione *main()*.

In appendice viene riportato il codice *Wiring* che controlla l'Arduino nel progetto. Questo codice é stato tratto da una libreria pubblica e successivamente modificato da Steven Gelinek[2] per adattarlo al caso in questione. Sono state implementate due funzioni: *readpins()* per leggere lo stato dei controlli, divisa per controlli continui (analogici) come i rotori della frequenza degli oscillatori e controlli discreti (digitali) come i rotori del cambio range degli oscillatori, e *writepins()* per modificare lo stato dei controlli come i display degli oscillatori e le lampadine di ON/OFF. In *Pure Data* il collegamento viene effettuato in una subpatch che impone all'Arduino di eseguire le seguenti operazioni ogni 75ms:

- lettura del valore dei controlli analogici (25ms)
- lettura del valore dei controlli digitali (25ms)
- scrittura dei valori dei controlli di feedback visivi (25ms)

La comunicazione tramite porta USB si basa sull'oggetto *comport*; questo viene utilizzato comunemente per leggere e scrivere dati tramite porta seriale, USB, Bluetooth e altro. Ogni controllo nelle interfacce viene misurato con un numero da 0 a 255, si sono quindi modificati i valori ricevuti nelle patch e quelli inviati alle interfacce. Dopo la ricezione dei valori dei controlli, viene eseguito il route di questi per indirizzarli ai destinatari e far funzionare la patch correttamente. Infine vengono inviati i valori di feedback visivo alle interfacce adattandoli opportunamente al range da 0 a 255. Di seguito si puó vedere la subpatch di controllo.

[1] Istituto di formazione post-dottorale fondato da Olivetti e Telecom Italia

[2] Ricercatore della Aalborg University coinvolto nel progetto DREAM

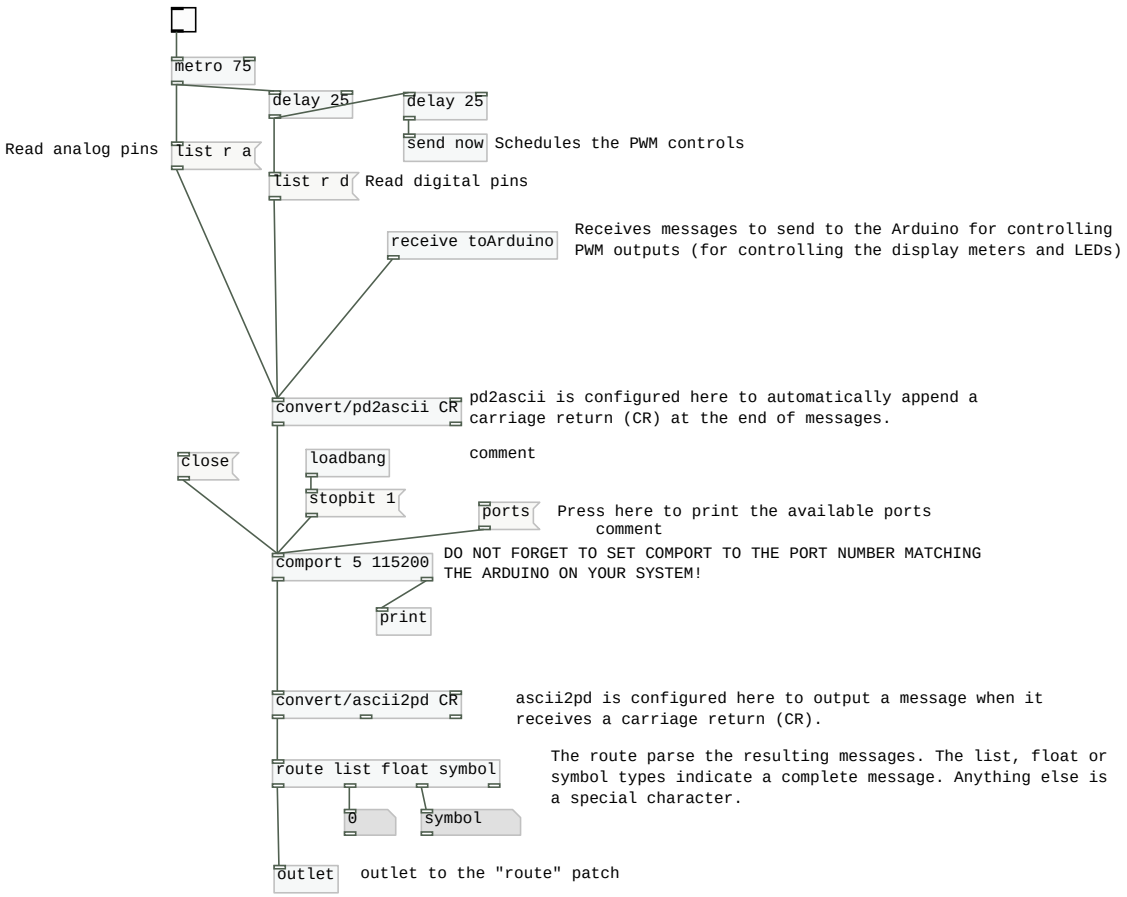


Figura 5.4: *Subpatch Arduino*

5.5 Evento DREAM

L'installazione é stata presentata il 15 giugno 2012 nel "Museo degli strumenti musicali" all'interno del Castello Sforzesco a Milano. Nell'evento sono stati presentati i risultati di questo lavoro e di altri due progetti relativi a due installazioni interattive: una catalogazione delle opere, foto e compositori consultabile tramite tablet dai visitatori, e un'installazione in cui é possibile ricreare la propria versione di "Scambi", l'opera di H. Pousseur.

(a) (b)

Figura 5.5: *Primo (a) e secondo (b) rack dell'installazione. Tratta da http://dream.dei.unipd.it*

Appendici

Appendice A

Codice

A.1 Codice Arduino

```
/*
 ARDUINO CODE for the DREAM interface
 by Steven Gelineck

 Heavily based on the SimpleMessageSystem
 found here: http://arduino.cc/playground/Code/SimpleMessageSystem
 Base: Thomas Ouellet Fredericks
 Additions: Alexandre Quessy

 */

// Include de SimpleMessageSystem library
#include <SimpleMessageSystem.h>

void setup()
{
    Serial.begin(115200); //Baud set at 115200
}

void loop()
{

  if (messageBuild() > 0) { // Checks to see if the message is
                          //complete and erases any previous messages
    switch (messageGetChar()) { // Gets the first word as a character
    case 'r': // Read pins (analog or digital)
      readpins(); // Call the readpins function
      break; // Break from the switch
    case 'w': // Write pin
      writepin(); // Call the writepin function
    }

  }
```

```
34
35  }
36
37  void readpins(){ // Read pins (analog or digital)
38
39    switch (messageGetChar()) { // Gets the next word as a character
40
41      case 'd': // READ digital pins
42
43      messageSendChar('d');  // Echo what is being read
44      for (char i=26;i<54;i++) {
45        messageSendInt(digitalRead(i)); // Read pins 26 to 53
46      }
47      messageSendInt(digitalRead(22)); // Reading also pin 22
48                                       //(for the mixer switch)
49
50      messageEnd(); // Terminate the message being sent
51      break; // Break from the switch
52
53    case 'a': // READ analog pins
54
55      messageSendChar('a');  // Echo what is being read
56      for (char i=0;i<16;i++) {
57        messageSendInt(analogRead(i)); // Read pins 0 to 15
58      }
59      messageEnd(); // Terminate the message being sent
60
61    }
62
63  }
64
65  void writepin() { // Write pin
66
67    int pin;
68    int state;
69
70    switch (messageGetChar()) { // Gets the next word as a character
71
72      case 'a' : // WRITE an analog pin (PWM outputs for meters and LEDs)
73
74      pin = messageGetInt(); // Gets the next word as an integer
75      state = messageGetInt(); // Gets the next word as an integer
76      pinMode(pin, OUTPUT); //Sets the state of the pin to an output
77      analogWrite(pin, state); //Sets the PWM of the pin
78      break;  // Break from the switch
79
80  /*
81      // WRITE a digital pin (Not used here)
82    case 'd' :
83
84      pin = messageGetInt();  // Gets the next word as an integer
```

```
85      state = messageGetInt();  // Gets the next word as an integer
86      pinMode(pin,OUTPUT);  //Sets the state of the pin to an output
87      digitalWrite(pin,state);  //Sets the state of the pin HIGH(1) or LOW(0)
88  */
89
90    }
91
92  }
93
```

Bibliografia

[1] S. Canazza, A. Rodá, M. Novati, and F. Avanzini. Active preservation of electrophone musical instruments. the case of the ”liettizzatore” of ”studio di fonologia musicale” (rai, milano). In *Proc. Int. Conf. Sound and Music Computing (SMC2011)*. Padova University Press, July 2011.

[2] R. McAulay and T.F.Quatieri. Speech analysis/synthesis based on a sinusoidal speech model. In *IEEE Trans, Acoustics, speech and signal process*, 1986.

[3] Maria Maddalena Novati and John Dack, editors. *The studio di fonologia, a musical journey 1954-1983*. Ricordi, 2012.

[4] G. De Poli, A. Piccialli, and C. Roads, editors. *Representations of musical signals*. MIT Press, 1991.

[5] Miller Puckette. *Pd documentation*.

[6] Miller Puckette. *Theory and Techniques of Electronic Music*. World Scientific Publishing Co. Pte. Ltd., 2006.

[7] C. Roads and J. Strawn, editors. *Foundations of computer music*. MIT Press, 1985.