UNIVERSITÀ DI PADOVA **DIPARTIMENTO DI INGEGNERIA**

TESI DI LAUREA MAGISTRALE

CaRo 2.0: PROGETTAZIONE E SVILUPPO DI UN SISTEMA INTERATTIVO PER L'ESECUZIONE ESPRESSIVA DELLA MUSICA

Corso di Laurea Magistrale in Ingegneria Informatica

Laureando: Dany Nichele

Relatore: Prof. Antonio Rodà

Correlatore: Prof. Sergio Canazza

Anno Accademico 2014/2015

Sommario

La capacità di suscitare emozioni è da sempre associata agli esseri umani, in questa tesi si vuole presentare un algoritmo che sia in grado di simulare l'abilità umana di trasmettere emozioni eseguendo delle partiture musicali.

Se i fattori emotivi sono fondamentali per l'esistenza della musica, diventa allora spontaneo chiedersi come la musica riesca a influire sulle persone e in quale maniera diverse strutture musicali riescano ad evocare differenti emozioni. Una volta appresi tali legami, è possibile costruire un modello e di conseguenza un algoritmo che simuli la capacità umana di trasmettere emozioni eseguendo partiture musicali. Scopo di questa tesi è descrivere la progettazione e lo sviluppo di una applicazione, chiamata CaRo 2.0. Il programma è in grado di riprodurre lo spartito applicando opportune deviazioni alla struttura originale del brano, concordemente all'espressività impartita dall'utente, sfruttando i legami appresi nella fase precedente.

Nel primo capitolo si illustrano quali possono essere le varie tipologie di spazio che sono usate per la rappresentazione dell'espressività contenuta all'interno di un brano. La trattazione continua analizzando la correlazione che intercorre tra caratteristiche musicali ed emozioni. Il capitolo primo si conclude analizzando i diversi fattori che influenzano una valutazione musicale eseguita attraverso sistemi automatici.

Il secondo capitolo descrive l'architettura del programma, la struttura delle classi e le librerie utilizzate per lo sviluppo. Sono riportate e commentate le funzioni più importanti dell'applicazione.

Il terzo capitolo si occupa della user interface, sono illustrate nel dettaglio tutte le funzioni e menù presenti nell'applicazione. La trattazione si conclude con il quarto capitolo nel quale si riportano i risultati di alcuni test dell'applicazione effettuati in laboratorio.

A Mio Nonno

“La musica è una delle vie per le quali
l’anima ritorna al cielo”

-Torquato Tasso-

Ringraziamenti

Sono passati due anni dalla mia tesi triennale e le persone che ho ringraziato allora sono le stesse che desidero ringraziare ora per questo importante traguardo della mia vita; mi ritengo un uomo fortunato perché sono circondato da persone che credono in me, mi sostengono e mi vogliono bene, è anche grazie a loro se sono arrivato a questo risultato.

Ringrazio il Professor Antonio Rodà per i preziosi insegnamenti, consigli e per il tempo dedicato alla mia tesi, insieme al Professor Sergio Canazza che mi ha fatto appassionare al mondo dell'informatica musicale. Ringrazio la mia famiglia, mia mamma, mia grande fonte di forza insieme a mio fratello, Igor, che sa sempre come movimentarmi la giornata. Proseguo ringraziando i miei amici, grandi compagni di feste e di vita; presenti anche nei momenti di bisogno. A questo proposito vorrei ringraziare, in maniera particolare, Andrea B. e Francesco F. Un ringraziamento anche agli amici del coro, Academia Ars Canendi, e della scuola di musica con cui ho passato momenti indimenticabili e grandi emozioni, in concerto e in compagnia. Ringrazio, infine, tutti coloro che mi hanno dato delle lezioni di vita, che hanno saputo ascoltarmi quando ne avevo più bisogno, che erano presenti regalandomi un sorriso e dandomi la forza di andare avanti; ringrazio anche quelli che hanno ostacolato la mia strada, rendendomi, così, ancora più forte.

Dany Nichele

Indice

Elenco delle figure

Capitolo 1

Sistemi automatici e modelli per l'esecuzione espressiva della musica

Introduzione

Uno dei principali motivi per cui si prende parte ad attività musicali, componendo, eseguendo, o semplicemente ascoltando è dato dal fatto che la musica è capace di suscitare delle emozioni profonde e significative.[1] La musica, però, non si ferma solo a trasmettere emozioni, la si trova, infatti, utilizzata in svariati campi e forme, ad esempio come mezzo didattico, di supporto in caso di disabilità; usata anche nell'ambito della sonificazione[2]. Nella trattazione seguente ci si limiterà a un'analisi sul contenuto emozionale trasmesso con l'esecuzione musicale. Le emozioni, all'interno di un brano musicale, possono andare da un puro godimento estetico per un costrutto sonoro, alla gioia o alla malinconia, che la musica a volte evoca o sostiene, al semplice sollievo dalla monotonia, dalla noia, dalla depressione, che le esperienze musicali quotidiane possono fornire.

La musica, inevitabilmente, ha la capacità di elevare il livello della vita emotiva. Ovviamente, individui e società fanno uso della musica anche per altri motivi, alcuni di essi già ricordati prima. Poiché molte attività musicali hanno anche un

[1] I primi due paragrafi sono una versione riveduta e aggiornata di http://tesi.cab.unipd.it/43095/1/Tesi.pdf

[2] L'idea alla base della sonificazione è quella di utilizzare dei procedimenti matematici per tradurre una successione di punti (ad esempio un grafico) o di colori (ad esempio una mappa) in una successione di note, in modo da generare un segnale percettibile con l'udito, dando così all'utilizzatore un'idea in tempo reale della situazione senza bisogno che questo interpreti un grafico, azione che comporterebbe un dispendio più tempo oltre che una capacità di lettura dello stesso.

carattere sociale, possono acquisire molti significati anche su questo piano, offrendo così ricompense sociali a chi vi partecipa. Per esempio, la conoscenza di certi tipi di musica è un prerequisito per essere considerati membri a pieno titolo di numerose sub-culture.

Se i fattori emotivi sono fondamentali per l'esistenza della musica, diventa allora spontaneo chiedersi come la musica riesca a influire sulle persone. A prima vista, un evento musicale può sembrare solo una raccolta di suoni di varia altezza, durata, e altre qualità misurabili. Se si pensa però al "suono organizzato" di Varèse, definito da DeLisa (2005) come un procedimento compositivo attentissimo all'auscultazione dei fenomeni sonori più sottili, alle interferenze provocate da un determinato incontro di timbri, agli agglomerati dei suoni, agli armonici superiori esaltati dall'intervento di un certo strumento piuttosto che un altro; si capisce che un'esecuzione musicale è un insieme di molti aspetti, non tutti così evidenti. La mente umana attribuisce a questi suoni un significato, si cercherà, quindi, di creare una corrispondenza biunivoca tra caratteristiche intrinseche del pezzo musicale e uno spazio, che meglio possa rappresentare il contenuto emozionale del pezzo; si analizzerà il rapporto che esiste tra musica ed emozioni. La struttura musicale fornita dalla partitura unita all'espressività data dall'esecutore diventano un elemento unico, che farà piangere o ridere, che piacerà o meno, che commuoverà o lascerà indifferente l'ascoltatore.

L'espressività di un brano musicale si trova, quindi, in diversi livelli di analisi; in prima battuta si trova il messaggio che il compositore intende trasmettere attraverso la scrittura dello spartito, con l'ausilio di segni espressivi riportati nella partitura. Si prosegue poi con l'interpretazione data dall'esecutore introducendo, intenzionalmente o meno, delle deviazioni rispetto alla partitura originale, saranno proprio queste deviazioni oggetto di studio. Si cercherà di costruire dei modelli computazionali che riescano a inglobare l'entità, il momento e il motivo dell'introduzione da parte dell'esecutore di queste variazioni; una volta costruito tale modello, sarà possibile effettuare un'esecuzione del brano automatica che tenga anche conto dei parametri espressivi. All'ascoltatore è lasciato il compito di interpretare il suono cercando di estrapolarne il contenuto emozionale.

1.1 Modelli automatici per l'esecuzione

La caratteristica che sta alla base di un sistema automatico per l'esecuzione espressiva della musica, è la capacità di convertire una partitura musicale in una performance che include deviazioni di tempo, intensità, timbro, e altre caratteristiche

che non sono riportate esplicitamente all'interno della partitura. Sebbene lo spartito sia il mezzo per comunicare l'intenzione espressiva del compositore, questo non contiene una descrizione completa dell'espressività, infatti, sono riportati nella partitura solo informazioni riguardanti la struttura ritmica e melodica del pezzo; elementi, ad esempio come: tempo e timbro sono a discrezione dell'esecutore. Per riuscire, quindi, ad avere una performance automatica partendo da una partitura, si dovranno costruire dei modelli che rappresentino l'espressività da dare al brano, unita poi alle informazioni provenienti dallo spartito, renderà possibile l'esecuzione espressiva tramite l'ausilio di mezzi automatici come un calcolatore. Il controllo espressivo è diventato, così, un'area di grande interesse nel campo del *Sound and Music Computing*[3] e del *Music Information Retrival.*[4] Generalmente, le principali strategie per progettare un sistema di esecuzione automatica, sono il metodo di analisi per misura e il metodo di analisi per sintesi. Di recente si sono utilizzate anche tecniche che fanno ausilio dell'intelligenza artificiale *(machine learning e case-based reasoning).*

1.1.1 Metodo di analisi per misura

Il primo metodo, analisi per misura, è basato sull'analisi delle deviazione che contraddistinguono un'esecuzione fatta da un essere umano. Lo scopo è quello di cogliere le regolarità presenti all'interno di esecuzioni espressive e descriverle poi mediante modelli matematici come viene esposto in Repp (1992) e ripreso ad esempio in Yang and Chen (2011): viene usato un modello di regressione per ottenere una densità di probabilità che descrive l'espressività di un pezzo. Il metodo di analisi per misura inizia dalla selezione del tipo di performance. L'esecutore potrà eseguire il pezzo in maniera libera oppure gli verrà imposto un certo tipo di espressività da riprodurre; dipenderà dallo scopo dello studio il grado di libertà della performance. In esecuzione si misureranno le proprietà fisiche di ogni singola nota. I parametri del brano misurabili, e quindi soggetti a variazioni durante l'esecuzione, sono molteplici: durata, inviluppo temporale, frequenza, vibrato

[3]Gruppo di ricerca in ambito scientifico, educativo e di diffusione di discipline legate all'applicazione di nuove tecnologie alla musica e al suono. Le attività del gruppo sono sempre state basate su un approccio interdisciplinare attraverso la collaborazione tra ricercatori e musicisti. Le attività di ricerca principali sono: sintesi sonora basata su modelli, rendering 3D, analisi e modellizzazione di contenuto emozionale ed espressivo in esecuzioni musicali e altri.

[4]MIR-Tecniche di Music Information Retrieval vengono usate al fine di estrarre caratteristiche significative direttamente dal segnale musicale e permettere all'utente di interagire con il sistema per mezzo di interfacce di alto livello come ad esempio selezionare tutti i contenuti multimediali aventi lo stesso tempo o altre proprietà desiderate.

ecc. La decisione su quante variabili considerare e la modalità di misurazione dipenderà dallo scopo dell'esperimento, dallo strumento utilizzato e dalle proprietà tecniche dello stesso. Una volta misurate le variabili fisiche della performance è necessario valutare affidabilità e consistenza dei dati misurati, classificando le esecuzioni in diverse categorie a seconda dei dati raccolti. Seguirà poi la scelta delle variabili più significative al fine della ricerca, le quali saranno usate per formulare un modello matematico. Il numero di variabili da considerare ai fini dello studio è sempre un punto delicato e ancora un problema aperto in quanto varia da caso a caso. Talvolta si esegue un'analisi multidimensionale, considerando quindi un maggior numero di variabili, con l'obbiettivo di identificare dei pattern indipendenti, si rimanda questa trattazione al capitolo successivo, in cui verranno discusse le caratteristiche di vari spazi usati per la rappresentazione dell'espressività trasmessa con l'esecuzione di un brano.

1.1.2 Metodo di analisi per sintesi

Il metodo di analisi per sintesi prende in considerazione diverse versioni del brano in cui le variabili, scelte per la ricerca (intensità, durata, ecc.), variano in maniera sistematica. Si crea un modello matematico dell'intera performance, viene poi formulato un giudizio basato sulla versione modellizzata del brano in relazione agli aspetti selezionati; bisogna, ovviamente, conoscere le variabili in gioco e la loro scala di rappresentazione. Del giudizio espresso dall'ascoltatore, deve essere valutata l'affidabilità e, inoltre, i vari ascoltatori, devono essere classificati in gruppi. All'inizio di una ricerca, quando vengono reclutati degli ascoltatori per avere un riscontro riguardante espressività percepita, viene loro sottoposto, preliminarmente, un questionario nel quale ogni persona è tenuta a rispondere ad alcune domande (età, sesso, nazionalità, esperienze musicali, ecc.); riuscendo così a collocare ogni individuo in un determinato insieme di appartenenza. Come evidenziato in Rentfrow and Gosling (2003), la personalità incide sulle preferenze musicali e quindi anche sulla percezione e valutazione del contenuto emotivo. Il giudizio sull'espressività è influenzata, inoltre, dalla cultura di appartenenza; come trattato in un confronto transculturale in Balkwill et al. (2004). Il metodo di analisi per sintesi procede osservando la differenza presente tra le variabili del modello matematico creato e i dati reali, nel caso non ci sia coincidenza dei risultati si esegue una modifica delle variabili considerate durante lo studio. La procedura con il metodo di sintesi continua, iterando il processo, fino a quando le variabili selezionate per lo studio convergono alla situazione reale. Il modello matematico così costruito fornirà una buona rappresentazione della situazione reale, proprio

perché creato tramite aggiustamenti progressivi delle variabili da considerare ai fini della ricerca.

1.1.3 Machine Learning

Un modello basato sul riconoscimento automatico delle emozioni musicali tramite il metodo di *machine learning*, è sviluppato cercando di trovare una relazione tra caratteristiche musicali e valori emozionali percepiti dal brano. Si procede, pertanto, con il formulare delle ipotesi iniziali sugli aspetti della performance che si vogliono modellare; successivamente si cerca una corrispondenza del modello testandolo con dati provenienti da situazioni reali. Come si tratterà nel capitolo successivo, si è visto che, assegnando semplicemente un valore alle emozioni, la valutazione, sul contenuto emotivo, sarà diversa a seconda dell'individuo interpellato. Per risolvere questo tipo di problema viene introdotto da Yang and Chen (2011) un nuovo approccio che rappresenta le emozioni con una distribuzione di probabilità, in uno spazio che meglio possa rappresentare il contenuto emotivo del pezzo. Sviluppando, inoltre, una metodologia che ricava la distribuzione dell'emozione contenuta all'interno di una clip, tramite la stima da campioni di tipo discreto, utilizzando tecniche di tipo *machine learning*; un algoritmo di fusione integrerà poi diverse caratteristiche dell'espressività, trovandone una rappresentazione opportuna, potenziando, inoltre, il modellamento delle emozioni percepite. Lo scopo, delle tecniche di *machine learning*, è di scoprire complesse dipendenze in un insieme molto grande di dati, senza formulare ipotesi preliminari e, quindi, non esplicitando la natura dei dati, si ha quindi la possibilità di ottenere nuove informazioni evitando qualsiasi assunzione di carattere musicale.

1.1.4 Case-Based Reasoning

Il Case-Based Reasoning (CBR) è basato sul confronto tra casi che presentano tra loro delle affinità. Con questo tipo di approccio, quando si deve risolvere una problematica, si va alla ricerca all'interno di una banca dati di un problema simile già risolto, pensando che questo possa avere uguale soluzione o comunque molto simile. Effettuando una ricerca e indicando al sistema le caratteristiche fondamentali della problematica in questione, il CBR interviene confrontando le informazioni fornite dall'utente con le informazioni presenti nell'archivio (come mostrato in *Figura 1.1*); questi sistemi infatti, sono in grado, analizzando un elevato numero di casistiche, di generalizzarne le caratteristiche e individuare le similitudini tra casi diversi. In questo modo riescono a proporre come risultato

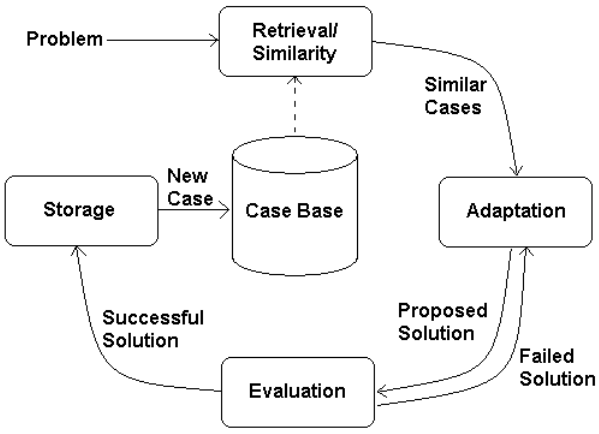

Figura 1.1: *Raffigurazione del processo Case-Based Reasoning*

della ricerca i contenuti che saranno utili per la risoluzione del problema. Si fa uso di questa metodologia, ad esempio nella ricerca Zeng et al. (2009), per il riconoscimento dell'espressività in maniera multimodale. Si incrociano cioè dati provenienti da espressioni spontanee del viso, movimenti della testa e movimenti del corpo. Vengono cercati, all'interno di un database, problemi simili a parità di dati raccolti al fine di riconoscere lo stato emotivo dell'individuo. Nel caso si riesca a trovare una soluzione non esistente di un determinato problema, questa verrà inserita all'interno della base di dati; potendo, così, essere utile per la risoluzione di problematiche future. Il sistema CBR è utile quando si ha a disposizione un insieme considerevole di problemi già risolti, disporre però di un archivio di soluzioni in una certa quantità e oltretutto organizzati in maniera efficiente, in modo da favorirne la ricerca, risulta essere difficile.

1.2 Modelli per la rappresentazione di emozioni ed espressività

1.2.1 Spazi espressivi

Generalmente è impossibile comporre, eseguire o ascoltare un pezzo musicale senza avere un coinvolgimento di tipo affettivo. Si andrà prima di tutto ad analizzare vari metodi e approcci per riuscire a rappresentare le sensazioni provate. Un problema centrale del riconoscimento automatico delle emozioni riguarda la

modellizzazione dello spazio espressivo, questo può avere una rappresentazione di tipo discreto oppure continuo.

Per quanto riguarda l'approccio discreto al problema, si procede con la divisione delle emozioni in un certo numero di categorie; tramite tecniche di *machine learning*[5] si attribuisce un' etichetta che caratterizzi ciascuna categoria. L'approccio di tipo continuo usa valori numerici plottati nello spazio bidimensionale, tridimensionale o in generale multidimensionale a seconda di quanta accuratezza si vuole dare al modello; lo spazio creato sarà comunque di tipo continuo. Tramite un modello di regressione sarà possibile rappresentare il contenuto emozionale in uno spazio che meglio lo rappresenti; di seguito sono esposte pertanto le varie tipologie di spazio emozionale, sottolineandone le rispettive proprietà.

1.2.1.1 Spazio emozionale discreto

Le esperienze che portano a delle emozioni, possono essere considerate come eventi distinti gli uni dagli altri; come affermato da: Barett (1998); Yang and Chen (2012). Alla base di questo approccio c'è il concetto di emozione primaria, emozione cioè che non si può scomporre in altre emozioni più semplici. Esiste pertanto un certo numero limitato di emozioni universali considerate classi primarie come: *happiness, sadness, anger, fear, disgust* e *surprise*; secondo Ekman oppure secondo Izard: *happiness, sadness, hostlity, guilt, surprise* e *interest*; tramite combinazione di emozioni primarie si possono ricavare tutte le altre emozioni definite secondarie. Esistono molte varianti alle classi prima citate, proprio per la difficoltà di riuscire a capire quali siano le emozioni che stanno alla base e quali loro combinazioni. La rappresentazione discreta considera le emozioni come eventi che si manifestano in un individuo in modo consequenziale, almeno per un intervallo di tempo abbastanza piccolo. Come evidenziato da Barett (1998), alcune esperienze riportano un alto grado di correlazione tra due diverse classi di emozioni considerate come primarie, questo denota che in determinate circostanze, una persona non è in grado di separare nettamente due diverse emozioni; si sottolinea, infatti, l'alto grado di correlazione tra lo stato d'animo *anxiety e depression*, appartenenti concettualmente a due classi di emozioni discrete distinte. Questo tipo di approccio è particolarmente adatto in soggetti che non si limitano a valutazioni di sola positività o negatività dell'evento, ma considerano anche il grado di attivazione della manifestazione. È però, evidente, il problema di granularità e ambiguità; lo scopo di un modello dovrebbe essere quello di rappresentare il più

[5] Tramite l'apprendimento automatico si cerca di estrapolare nuovi elementi in comune e complesse dipendenze su insiemi molto grandi di dati; (si veda § 1.1.3)

fedelmente possibile i dati reali, in questo caso, però, la soggettività dell'emozione e vincoli imposti dal linguaggio non permettono di avere una rappresentazione che rispecchi appieno la situazione reale. A parità di contenuto emozionale il giudizio da parte di individui diversi potrebbe non essere lo stesso.

L'approccio discreto al problema si presenta pertanto piuttosto limitativo, poteva però risultare comodo da un punto di vista computazionale: inserimento delle etichette, quindi stringhe rappresentanti le emozioni, in basi di dati facili da gestire. La modellizzazione discreta diventa ancora più limitativa se si pensa a una difficoltà di interpretazione degli aggettivi espressi in lingue diverse dalla propria, dando così sfumature semantiche ancora più ampie. È pertanto opportuno, analizzare altri tipi di approcci, che riescano a risolvere le problematiche evidenziate; prima di passare però all'analisi di tipo continuo sarà presentato un altro tipo di approccio, che tenta di eliminare le ambiguità semantiche, raggruppando in insiemi gli aggettivi che descrivono emozioni simili.

Clusters emozionali

Un altro tipo di approccio di tipo discreto è la *checklist* di aggettivi introdotta da Hevner (1936). Attraverso uno studio, Hevner, introdusse otto insiemi contenenti aggettivi con lo stesso contenuto emozionale, disponendoli in maniera circolare, come mostrato in *Figura 1.2*. Spostandosi da *cluster* a *cluster*, in maniera circolare, crescerà la differenza del contenuto emotivo tra i vari insiemi, fino a raggiungere il massimo della discrepanza nella posizione diametralmente opposta. La divisione di Hevner (1936) fu successivamente rivisitata da Farnsworth (1969) e recentemente cambiata con l'aggiunta di una nuvola in più per un totale di nove da Shubert (2003). Sedici aggettivi sono stati cancellati dalla lista originaria che ne conteneva 67, poiché non rispecchiavano appieno la semantica di nessun insieme. Farnsworth (1969) aveva introdotto dieci *clusters*, ma il decimo insieme, conteneva solo l'aggettivo *frustrated*: è stato pertanto eliminato nella successiva revisione. Gli *clusters* contengono, appunto, più aggettivi proprio per eliminare il più possibile l'ambiguità; la presenza di un insieme con solo un termine si discosta dalla struttura del metodo in divisione per *clusters*. Nella rivisitazione di Shubert (2003), viene chiesto ai partecipanti della ricerca di esprimere, tramite un indice numerico da 0 a 7, quanto gli aggettivi, presentati in ordine casuale, potessero descrivere il brano ascoltato. Gli aggettivi che ottenevano una valutazione media minore di 4 venivano scartati da quella che doveva essere la lista finale. Raccolti tutti i dati ed eliminata, come detto in precedenza, il decimo *cluster*, quello che si ottiene è la lista mostrata in *Figura 1.3*. Si nota, osservando la tabella, che sono

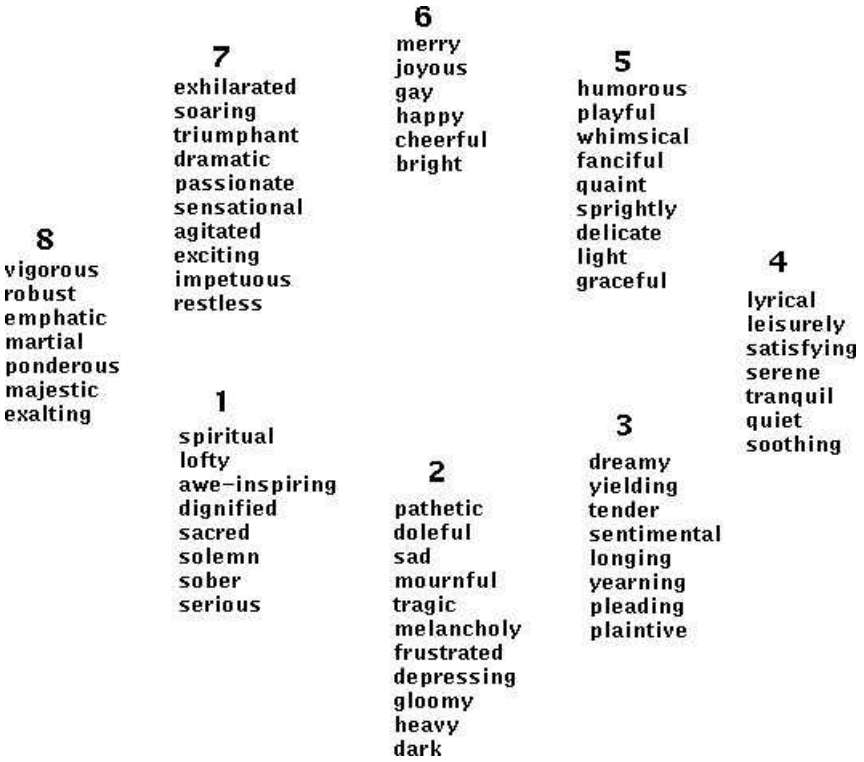

Figura 1.2: *Gli otto clusters emozionali introdotte da Hevner*

state invertite gli insiemi H e I, l'insieme E posizionato dopo il G, si possono pertanto disporre i vari insiemi, partendo dal primo quadrante, in modo circolare nella successione: A, B, C , D, F, G, E, I, H; il *cluster* H si ricongiunge con la A per chiudere il cerchio. Il *cluster* G, per problemi di simmetria tra i quadranti e per mantenere una forma circolare uniforme, è stata omessa dalla rappresentazione grafica. Sette aggettivi, che preventivamente erano stati tolti da Farnsworth, sono stati ripristinati. Uno dei problemi dell'approccio discreto, consiste nel numero delle classi delle emozioni primitive, che è di gran lunga inferiore in comparazione alla ricchezza delle emozioni musicali percepite da un individuo. Usare una granularità più alta non risolve il problema perché il linguaggio utilizzato, per descrivere le emozioni, è intrinsecamente ambiguo e varia da persona a persona. Diventa impensabile, per un'analisi della psicologia delle emozioni, aumentare il numero di classi di suddivisione. Si presenta ora un approccio di tipo continuo, permettendo, così, una caratterizzazione più puntuale delle emozioni.

UPDATED HEVNER ADJECTIVE LIST

	M	SD	Source		M	SD	Source
Cluster A				Cluster F			
Bright	6.04	1.63	HF,6,a	Dark	5.84	1.95	HF,2,f
Cheerful	5.35	1.82	HF,6,a	Depressing	4.64	2.31	HF,2,f
Happy	5.29	1.97	HF,6,a	Gloomy	4.64	2.18	HF,2,f
Joyous	5.22	2.01	HF,6,a	Melancholy	5.36	2.17	H,2,f
Cluster B				Mournful	4.64	1.76	HF,2,f
Humorous	4.67	2.25	H,5,	Sad	5.53	2.02	HF,2,f
Light	6.02	1.73	HF,5,b	Solemn	5.09	2.11	HF,1,f
Lyrical	6.23	1 74	HF,4,c	Cluster G			
Merry	4.30	1 88	HF,6,a	Heavy	5.61	1.72	H,2,
Playful	5.17	2 01	HF,5,a	Majestic	5.81	1.87	HF,8,g
Cluster C				Sacred	4.92	2.35	HF,1,g
Calm	6.06	1.76	HRW,4,	Serious	5.08	2.15	HF,1,
Delicate	5.13	2.15	HF,5,c	Spiritual	5.30	2.06	HF,1,g
Graceful	6.02	1.71	HF,5,c	Vigorous	5.06	1.84	HF,8,g
Quiet	5.69	1.97	HF,4,c	Cluster E			
Relaxed	5.04	2.05	RW,,	Tragic	4.71	1.99	HF,2,f
Serene	4.66	2.15	HF,4,	Yearning	4.10	2.39	HF,3,e
Soothing	5.69	1.92	HF,4,c	Cluster I			
Tender	4.26	2.18	HF,3,c	Agitated	4.86	2.06	HF,7,I
Tranquil	5.29	1.97	HF,4,c	Angry	4.07	2.53	RW,,
Cluster D				Restless	4.63	2.23	H,7,
Dreamy	4.79	1.89	HF,3,d	Tense	5.71	1.77	RW,,
Sentimental	4.48	2.18	HF,3,d	Cluster H			
				Dramatic	6.54	1.52	HF,7,h
				Exciting	5.58	1.96	HF,7,h
				Exhilarated	4.27	2.08	HF,7,h
				Passionate	5.92	2.00	H,7,
				Sensational	4.83	2.31	H,7,
				Soaring	4.25	2.27	H,7,
				Triumphant	5.27	2.03	HF,7,g

Figura 1.3: *Aggiornamento della lista di Hevner Shubert (2003)*

Note- Le colonne mostrano: la media delle valutazioni date in una scala da 0 a 7 (M), la deviazione standard (SD), l'origine dell'aggettivo (H = lista aggettivi di Hevner, F = Farnsworth, R = Russel, W = Whissell), indicate poi il numero di originario di appartenenza alle nuvole di Hevner e la lettera originaria di appartenenza delle nuvole di Farnsworth. La sequenza delle nuvole è A, B, C , D, F, G, E, I, H per ritornare poi alla A.

1.2.1.2 Spazio emozionale continuo

Mentre l'approccio di tipo discreto si focalizza principalmente nel distinguere un'emozione da un'altra, l'approccio dimensionale continuo si focalizza sul posizionare le emozioni in un grafico cartesiano di dimensione bassa, solitamente due o tre dimensionale, con l'intento di rappresentare le emozioni umane. La dimensione interna, dello spazio che andrà a rappresentare le varie emozioni, viene

trovata analizzando la correlazione tra i termini affettivi. A questo fine viene svolta una ricerca e ai soggetti sottoposti alla stessa è richiesto di quantificare con un numero, per esempio utilizzando la scala Linkert a 7 punti (0 = non presente, 3 = moderatamente presente, 6 = certamente presente), l'ammontare di emozione descritta da un certo aggettivo contenuta nello stimolo musicale; questa procedura viene ripetuta per un numero abbastanza grande (60-80) di aggettivi, in modo da formare un quadro abbastanza completo. Tramite poi, tecniche di analisi, vengono ricavate un numero di dimensioni fondamentali, solitamente due o tre; successivamente si plotteranno i dati nel sistema cartesiano creato. Data la complessità nel rappresentare lo spazio espressivo, i modelli continui usati sono molteplici, caratterizzati da dimensioni, parametri misurati e approccio di misurazione diversi.

Valence and Arousal

Lo spazio espressivo continuo *valence-arousal* è uno spazio bidimensionale nel quale le componenti fanno riferimento a delle esperienze soggettive. Le due dimensioni sono considerate essenziali e indipendenti, oltre che ad essere il più possibile ortogonali. La prima componente cartesiana è *valence*, rappresenta quanto i sentimenti di una persona sono influenzati dalle valutazioni positive o negative di persone, cose o eventi; tipicamente posta sull'*asse x*, con valenza positiva verso destra. La seconda componente è *arousal*, rappresenta il grado di attivazione, cioè quanto una persona è disposta a compiere delle azioni o meno, quest'ultima posta sull'*asse y*, con attivazione positiva al crescere con il verso convenzionale dell'asse.

Viene analizzato da Barett (1998) il tipo di approccio da adottare per rappresentare le emozioni; *valence-arousal* viene utilizzato in soggetti che prediligono la facilità di riconoscimento della prima delle due componenti, cioè il grado di positività. Viene evidenziata un'alta correlazione tra le emozioni soggettive di simile valore di *valence*, ciò potrebbe indicare che questi individui stiano riportando parecchi stati insieme, oppure potrebbe essere sintomo di un'incapacità nel riuscire a discernere due stati comunemente considerati distinti. Nella ricerca sulle emozioni fatta da Vieillard et al. (2008), viene usato un modello bidimensionale *valence-arousal* per rappresentare le varie emozioni. Vengono scelte tre emozioni definite di tipo primitivo, quali: *happy, sad, scarry*; viene inserito inoltre l'aggettivo *peaceful*, anche se non considerata un'emozione base, viene fatta questa scelta per fornire un aggettivo opposto al termine *scary*, visto che *happy* e *sad* sono in contrasto. In realtà, nella lista delle emozioni base di Ekman, non era presente

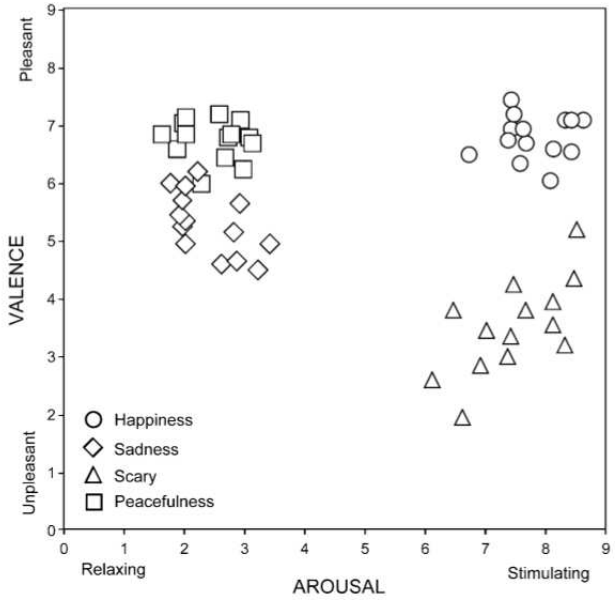

Figura 1.4: *Rapprensentazione stimoli musicali nel piano valence-arousal Vieillard et al. (2008)*

Note: Media di valence-arousal su una scala di 10 punti. Ogni simbolo rappresenta uno stimolo musicale

l'aggettivo *scary* ma bensì *fear*, viene fatto questo cambio perché quest'ultimo aggettivo viene considerato troppo ambiguo per la descrizione delle emozioni di uno stimolo musicale. Un esecutore può esprimere *fear* suonando delicatamente oppure suscitare la stessa emozione nell'ascoltatore suonando con veemenza. Per ogni pezzo musicale vengono calcolate le medie per i valori di *valence* e *arousal*, esprimendoli con una scala da 0 a 9, e riportati nel corrispondente spazio bidimensionale, come si può vedere in *Figura 1.4*. Gli aggettivi *happy* e *peaceful* vengono classificati come piacevoli mentre *scary* come non piacevole. L'emozione *sad* viene collocata dalla parte di piacere dell'asse di *valence*. Lungo l'asse *arousal*, cioè quella che rappresenta l'attivazione, *sad* e *paeceful* hanno ricevuto un valore basso; un valore alto, quindi grande attivazione, per *happy* e *scary*. Osservando i risultati della ricerca, si nota, nella dimensione *arousal*, un distacco sostanziale tra i due gruppi di stimoli musicali, cioè tra quelli con bassa e alta attivazione; per quanto riguarda la dimensione di *valence*, la distanza tra i vari campioni diventa molto meno significativa fino, nel caso di *scary* e *paeceful*, a trovare un'intersezione delle emozioni. L'emozione *happy* è stata riconosciuta nel 99% dei casi,

seguono poi *sad* con l'84% e *scary* con l'82%; l'emozione meno riconosciuta risulta essere *paeceful* con il 67% e per un 12% viene valutata in modo ambivalente; con il 75% viene scambiata con il *scary*, che risulta avere valori simili in termini di *valence*. In precedenza, era già stato sottolineato da Barett (1998), che esisteva una grande correlazione per quanto riguarda, emozioni con valore simile in *valence*; ne è testimonianza anche quest'ultimo esempio.

Il modello *valence-arousal* risulta essere un buon modello di rappresentazione delle varie emozioni, anche se si nota una dipendenza accentuata in aggettivi con valori in *valence* comparabili. L'emozione *paeceful*, nello studio precedente, non viene considerata come basilare, è pertanto naturale che questa abbia una certa correlazione con altre emozioni, in quanto, considerata proprio combinazione di emozioni base. Da tenere ben presente, la relazione che esiste tra l'esternazione dell'emozione e la risposta biologica, un'esperienza soggettiva non è altro che una traduzione di un fenomeno biologico in una rappresentazione cosciente dello stato. Giudizi uguali possono pertanto essere espressi per descrivere emozioni diverse in individui distinti. Non esiste, quindi, un modello universale di rappresentazione emozionale; verranno discusse in seguito circostanze che possono influire sull'emotività di una persona.

Modello circolare per Valence-Arousal

Si passa ora a esaminare un modello di rappresentazione che mette assieme due modelli già visti. La rappresentazione introdotta da Russel (1980), consiste in un modello bidimensionale *valence-arousal* con l'assegnazione in varie posizioni, dello spazio bidimensionale, di etichette rappresentanti varie emozioni, si viene a formare una struttura circolare, come si può vedere da *Figura 1.5*. Le considerazioni per questo modello sono del tutto analoghe a quelle fatte per i modelli precedentemente presentati, infatti, se il modello a *clusters* di Hevner (1936) (*Figura 1.2*) lo si inserisce opportunamente nel piano bidimensionale, considerando cioè la dipendenza con gli assi di *valence-arousal*, si ottene praticamente il modello appena presentato. Altro esempio si ha in Vieillard et al. (2008), si vede da *Figura 1.4* il posizionamento di alcune emozioni proprio nel piano bidimensionale in questione. Viene riportato da Yang and Chen (2012) il pensiero tratto da un articolo di Sloboda and Juslin (2001), nel quale si afferma come da un punto di vista teorico la dimensione di attivazione, cioè *arousal*, è una delle maggiori variazioni percettibili nell'ambito delle emozioni, insieme alla dimensione *valence*, la percezione cioè di qualcosa di piacevole o meno, ha una diretta correlazione su quello che è il comportamento di un individuo. Affermando, così, che

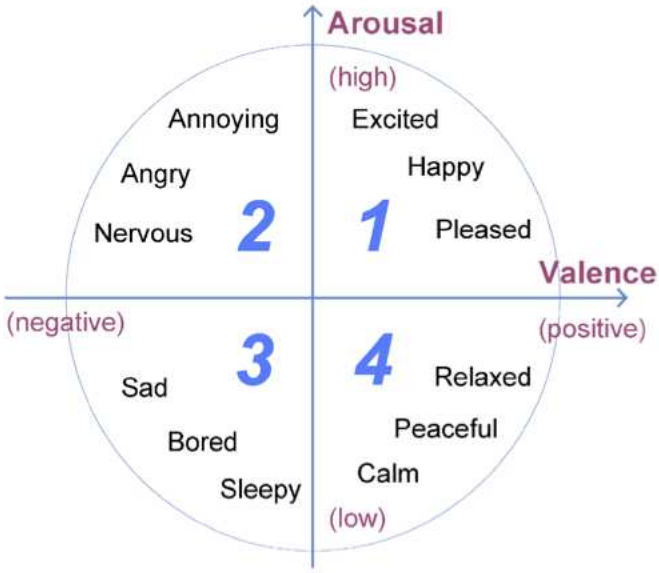

Figura 1.5: *Rapprensentazione stimoli musicali nel piano valence-arousal (Russel) Yang and Chen (2012)*

Note: Le posizioni delle varie emozioni non sono esatte ma approssimative.

il modello *valence-arousal* di Russel (1980) è il "cuore" dell'emozione, rappresenta cioè concretamente l'esperienza emozionale a un livello crudo e primitivo. Descrivere le emozioni usando un modello bidimensionale, però, non è libero dalle critiche del caso. Si sottolinea, infatti, come l'emozione *anger*, nello spazio *valence-arousal*, viene rappresentata nelle vicinanze immediate di *fear* (entrambe nel secondo quadrante), pur essendo molto differenti in termini di reazioni biologiche nell'organismo; un'emozione è di tipo attivo l'altra passivo. In risposta a questa mancanza, sembra opportuno introdurre una terza dimensione o più, per poter avere un quadro completo di rappresentazione dello spazio emozionale.

1.2.1.3 Spazi emozionali con dimensione superiore a due

Il problema su quante dimensioni usare, per poter rappresentare lo spazio emozionale, è tuttora aperto. Nella ricerca di Fontaine et al. (2007) si cerca di trovare una soluzione a questa grande problematica. Durante lo studio vengono osservati i cambiamenti attraverso sei attività: stima di eventi, cambiamenti psicologici (sen-

sazioni corporali), espressioni motorie (faccia, voce, gesti), tendenza nell'azione, esperienze soggettive e regolazioni emozionali; nella maggior parte degli studi precedenti si sono osservati solo una o due di queste caratteristiche. Per ottenere, in modo evidente, quale sia la dimensione minima ottima a rappresentare lo spazio emozionale si usa un campione eterogeneo di persone, nello specifico: inglesi, francesi e tedeschi; chiedendo loro di valutare 24 termini emozionali mettendoli in scala, considerando nella valutazione 144 possibili caratteristiche. Per ridurre la dimensione dello spazio emotivo, osservando le componenti lungo ogni dimensione, viene usato il PCA (*principal component analysis*), per trovare la minor varianza nei dati raccolti. L'elaborazione dei dati produce un modello composto da quattro dimensioni, quali: *evaluation-pleasanteness, potency-control, activation-arousal, unpredictabily.* L'interpretazione delle quattro dimensioni è basato sulle relazioni tra le 144 caratteristiche emozionali e sulle coordinate cartesiane dei 24 termini emozionali. In *Figura 1.6* sono rappresentate le coordinate cartesiane, dei 24 termini emozionali, nel nuovo spazio di rappresentazione in quattro dimensioni. Le quattro dimensioni non sono rappresentabili graficamente contemporaneamente, vengono così presentati, dei grafici cartesiani divisi, ciascuno con ascissa *valence*. La prima dimensione, *evaluation-pleasantness*, può essere interpretata come tendenza all'azione contro l'assenza di movimento. La seconda dimensione, *potency-control*, è caratterizzata dal controllo, sensazione di potenza, dominazione o l'essere dominati. In questa dimensione emozioni come *pride, anger, contempt* sono in opposizione a *sadness, shame, despair* (*Figura 1.6*). La terza dimensione, *activation-arousal*, è principalmente caratterizzata dall'attivazione, come il rapido battito del cuore oppure la prontezza nell'agire. L'ultima dimensione, *unpredictabily*, è caratterizzata dalle cose nuove, dalla novità, sorpresa e tutto ciò quindi, che non può essere previsto. Si può affermare che le dimensioni più importanti sono le prime tre, l'ultima è degna di nota nel caso in cui si considerino le reazioni istantanee a stimoli nuovi, di sorpresa e pertanto non familiari.

Viene presentato da Schimmack and Reisenzenin (2002) un ulteriore spazio emozionale, questa volta composto da tre dimensioni, quali: *Energetic Arousal, Tense Arousal e Valence*; si ha pertanto una divisione di quello che prima era chiamato solo l'asse *arousal* in due assi che riportano il termine *arousal* nel finale del loro nome, si cercherà pertanto di provare, che *Energetic Arousal e Tense Arousal* non sono combinazione di *Valence e Activation*, ma bensì due distinte dimensioni; valide per essere incluse nella formazione di un nuovo spazio rappresentativo. Attraverso *energetic arousal* si va a rappresentare la sensazione di addormentamento oppure il grado di energia posseduta, per quanto riguarda *tense arousal*, invece,

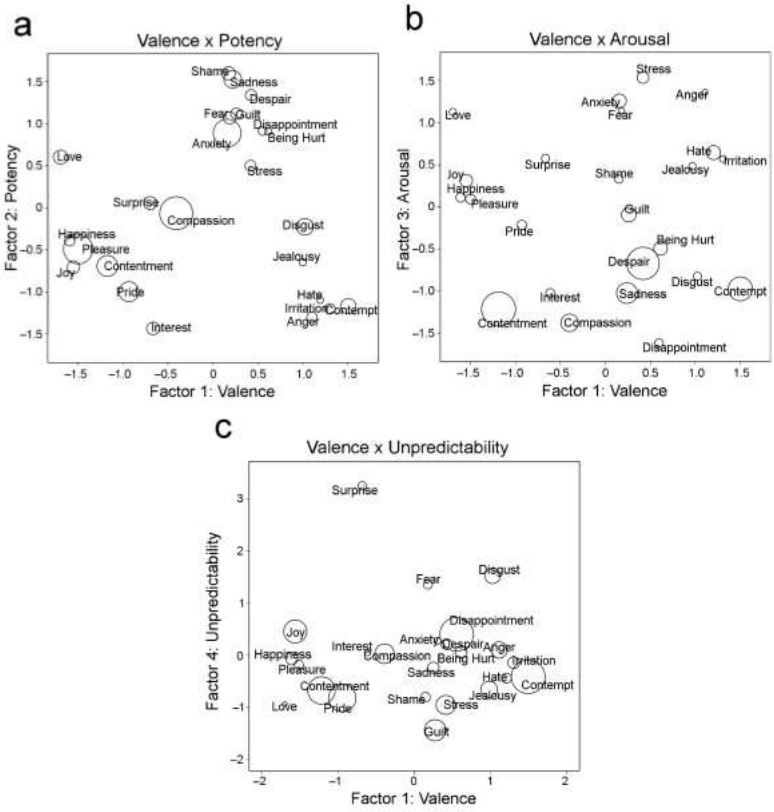

Figura 1.6: *Rapprensentazione dei 24 termini emozionali nello spazio quadridimensionale Fontaine et al. (2007).*

Note: I punti medi dei vari cerchi rappresentano le coordinate medie lungo le tre lingue. Il diametro dei cerchi rappresenta la media della distanza euclidea tra le coordinate delle tre lingue; più piccolo è il cerchio, più simili sono i termini rispettivi attraverso le tre lingue. I tre grafici raffigurano i termini emozionali nelle cordinate: (a) Evaluation-Pleasantness x Potency-Control, (b) Evaluation-Pleasantness x Activation-Arousal, (c) Evaluation-Pleasantness x Unpredictability.

quantifica il grado di nervosismo o di calma di un individuo. La ricerca sottolinea, in prima battuta, che le due nuove dimensioni sono scatenate da cause differenti; per esempio *energetic arousal* è influenzato dal battito cardiaco che corrisponde al grado di attività delle celle del cervello che regolano l'organismo, cosa che non succede per *energetic arousal*. Si evidenzia, inoltre, che le due dimensioni possono crescere in direzioni opposte; per esempio, secondo uno studio del livello di zuccheri contenuti nel sangue, si osserva che *energetic arousal* decresce, mentre *tense arousal* cresce all'aumentare della glicemia. Terzo motivo, che sostiene la tesi di due dimensioni indipendenti è la diversità delle conseguenze; per esempio:

numerosi studi dimostrano che *energetic arousal* rappresenta meglio le operazioni cognitive di *tense arousal*. Lo studio di Schimmack and Reisenzenin (2002) presenta, quindi, un nuovo approccio per risolvere la controversia riguardante la dimensione dell'attivazione. In un primo momento i ricercatori considerano *energetic arousal* e *tense arousal* due dimensioni separate, in quanto pensavano fossero causate da fattori differenti e basate su sistemi neurologici diversi. In contrasto a questa tesi c'è l'ipotesi di: *valence activation hypotesis*, la quale afferma che *tense e energetic arousal* siano combinazione della stessa dimensione di attivazione con la valenza. In linea con quest'ultima ipotesi si è trovato una correlazione positiva tra le due componenti e *valence* dovuta, però, soltanto alle correlazioni tra i residui, eliminando, quindi, la correlazione tra i residui, si trova una legame tra le due dimensioni prossimo allo zero. Il modello bidimensionale di attivazione, *Energetic Arousal, Tense Arousal*, non può essere preso da solo come rappresentativo, poiché non tutte le emozioni riescono a trovarne una rappresentazione, non si riesce nemmeno a trovare una corrispondenza biunivoca con *valence-activation*. Uno spazio completo potrebbe essere quello tridimensionale, come anticipato in precedenza, cioè *Energetic Arousal, Tense Arousal e Valence*, anche se, le dimensioni nuove di attivazione, presentano con *valence* un alta correlazione, si ha pertanto un modello piuttosto ridondante.

Il numero ottimo di dimensioni, da introdurre in uno studio, dipende dalle domande che la ricerca si pone. Per ricerche che si concentrano soltanto sull'attivazione potrebbe essere sufficiente una sola dimensione (*arousal*). Per una ricerca che mira nella caratterizzazione di termini emozionali, con significato semantico molto simile, potrebbero non essere sufficienti neanche quattro dimensioni. Si può affermare che, in via generale, già quattro dimensioni riescono a dare una caratterizzazione molto buona dello spazio emozionale. I risultati di Fontaine et al. (2007), portano ad affermare che un semplice modello bidimensionale, come *valence-arousal*, non riesce a caratterizzare in maniera opportuna emozioni come *anger e fear*, che sono invece nettamente separate nella dimensione *potency-control* e *unpredictability*; come si può vedere da *Figura 1.6*. Un modello come quello *valence-arousal*, bidimensionale, riesce a rappresentare degli stati emotivi da un punto di vista comprensivo, cioè, una volta che questi si manifestano nel nostro organismo si riesce a darne una caratterizzazione da un punto di vista logico. La dimensione *potency-control*, invece, riesce a dare una connotazione a emozioni, che si manifestano sia in maniera comprensiva, sia a emozioni che si manifestano all'interno dell'organismo come, ad esempio, sintomi grastrointestinali; delineandone l'impronta biologica. La quarta dimensione dello spazio, cioè *unpredictability*, svolge anch'essa un ruolo molto importante; un individuo di

fronte a uno stimolo mai provato o che non si aspetta di provare avrà una reazione diversa, rispetto a una situazione già famigliare. L'influenza dell'ambiente e le circostanze, in cui si trova una persona al momento dell'emozione, sono elementi che influisco ampiamente sull'intensità e modo della manifestazione della stessa. Non esiste pertanto, tenendo in considerazione quanto discusso in precedenza, un numero di dimensioni ottimale da prendere in considerazione, la decisione della cardinalità dello spazio rappresentativo emozionale deve essere fatta di caso in caso per non incorrere ad un sovra o sotto dimensionamento dello stesso, avendo, nel primo caso delle coordinate dimensionali ridondanti tra loro, mentre, nel secondo, una carente caratterizzazione delle emozioni.

1.2.2 Struttura musicale ed emozioni

Ci sono diversi fattori che influenzano le emozioni percepite in musica. Emozioni differenti sono associate a caratteristiche musicali diverse. Per esempio, mentre *arousal* è legato al Tempo (veloce/lento), volume (alto/basso), e timbro (intenso/morbido), *valence* è legata al modo (maggiore/minore) e all'armonia (consonante/dissonante). Si nota, inoltre, che una percezione emozionale non è raramente legata a una sola caratteristica, ma a una combinazione di esse. L'emozione percepita è frutto di una combinazione tra vari fattori, quali l'esecuzione del performer, ma prima ancora alla scrittura della partitura e ovviamente dall'intenzione del compositore. Il ricercatore per valutare l'emozione, può avvalersi di diversi metodi: può chiedere una valutazione diretta all'ascoltatore riguardante le sue emozioni provate durante l'ascolto oppure procedendo con la variazione di determinati parametri intrinseci del pezzo (Tempo, ritmo, modo) uno alla volta, estrapolando un legame tra la struttura musicale ed il contenuto emozionale espresso.

1.2.2.1 Emozioni e caratteristiche musicali

Si cercherà ora di elencare le principali caratteristiche di un brano musicale cercando di legarle con le emozioni tipiche che si manifestano al variare delle stesse.

Tempo Come evidenziato da Balkwill et al. (2004), il Tempo veloce è associato a: *happiness, joy, anger, fear, activity*; quindi ci saranno alti valori di entrambe le componenti *valence-arousal*; con un Tempo lento, si vanno ad esprimere emozioni come: *sadness, disgust, borredom, solemness*; caratterizzate per avere alta *valence* mentre bassa attivazione, quindi, un valore basso in *arousal*.

Modo Canazza et al. (2011) analizza la dipendenza delle emozioni al variare del modo del brano, caratteristica intrinseca dello stesso, si evince pertanto la caratterizzazione nei modi maggiori di emozioni con alta *valence* ma bassa *arousal*, emozioni come: *serene, happiness, joy*; il modo minore è caratterizzato da alto valore in *arousal* indipendentemente dal valore in *valence*, identificando emozioni come: *sadness, tension, disgust, anger*.

Intensità A un brano con molta intensità verrano associati aggettivi come: *intensity, tension, power, anger*; che hanno un alto valore in *arousal*; per brani con bassa intensità avviene l'associazione di aggettivi come: *softness, tendeness, sadness*; caratterizzati, quindi, da valori bassi in *arousal*.

Altezza dei suoni La presenza di picchi elevati comporterà all'espressione di: *exciting, happy, potency, surprise, activity*; caratterizzati, quindi, da un alto valore della componente *arousal*, viceversa, la presenza di picchi di bassa entità denoterà emozioni del tipo: *boredom, pleasantness, sad*; la componente *arousal* sarà di valore basso.

Armonia Con l'armonia di tipo consonante si esprime: *serene, happy, relaxed*; mentre, con una di tipo dissonante aggettivi espressivi come: *disgust, anger, unpleasantness, tension*.

Tonalità Con armonia di tipo cromatico si trasmette: *angry, sad*; con armonia tonale: *joy, pacefull, happiness*; con armonia di tipo atonale si trasmette, ad esempio, emozioni come *angry*.

Timbro Assieme a Tempo e modo del brano, il timbro è un altro elemento caratterizzante l'espressività della performance, come sottolineato, per esempio, da Canazza et al. (2011). L'influenza sull'espressività, da parte del timbro, viene discussa da Eerola et al. (2012), conducendo una ricerca che cerca di capire la correlazione tra variazioni timbriche e variazioni espressive. Dalla ricerca, emerge una dipendenza del timbro principalmente da tre parametri, quali: tipo di attacco, brillantezza, che dipende a sua volta dall'energia in alta frequenza, e dal flusso spettrale o irregolarità nello spettro. Il timbro viene quindi caratterizzato da una varietà ampia di caratteristiche, che vanno dal tipo di strumento a modalità di esecuzione. In *Figura 1.7* sono riassunti alcuni parametri caratteristici di un'esecuzione e la loro correlazione con lo spazio bidimensionale *valence-arousal*, scelto per rappresentare le varie emozioni. Dai risultati dello studio si riescono a classificare le emozioni al

	Valence	Energy
	R^{2adj}	R^{2adj}
Prediction rate	.54	.74
	β	β
Attack Slope	-.17*	.32***
Envelope Centroid	.71***	-.42***
Ratio of HF-LF energy	-.52***	.71***
Spectral Skewness	-.30***	-.30**
Spectral Regularity	.03	-.13
Spectral Flux	.81***	-.09
Sub-Band No. 6 Flux	-.75***	.33**

* $p < .05$, ** $p < .01$, *** $p < .001$

Figura 1.7: *Riassunto dell'analisi di regressione Eerola et al. (2012)*

variare delle caratteristiche prima citate; si identificano, ad esempio, in suoni con molte armoniche, emozioni come: *surprise, disgust, fear, potency*. In suoni con le armoniche alte amplificate si trovano emozioni come *anger*. Emozioni come: *sadness, pleasantness, boredom*; in suoni con poche armoniche, mentre, in quelli con l'energia distribuita in bassa frequenza si trovano: *tenderness, sadness*. Strumenti a strofinio delle corde si prestano a trasmettere emozioni, quali: *sadness, anger*; strumenti a fiato, come il flauto si trovano adatti a trasmettere una sensazione di pace.

Articolazione Da Eerola et al. (2012), si evince, che articolazioni con attacco veloce, come: sforzato, marcato e staccato; sono caratterizzate da un alto valore in energia; altro elemento caratterizzante per la distinzione emozionale sarà il tipo di inviluppo. Con articolazioni di tipo staccato si avrà la comunicazione di emozioni come: *anger, fear, activity, energy*; con articolazione di tipo legato si esprimerà: *softness, solemnity, tenderness, sadness*. Come detto in precedenza anche l'inviluppo gioca un ruolo importante dal punto di vista espressivo, si avrà pertanto: con un inviluppo di tipo quadrato, l'espressione di emozioni come: *activity, surprise, anger*; con un inviluppo tondeggiante, l'espressione di emozioni come: *boredom, disgust, fear, sadness*. L'articolazione è caratterizzata dalla variazione della componente *valence*, come si può vedere da *Figura 1.8*, risultati ottenuti dalla ricerca di Eerola et al. (2012). Si nota che i suoni pizzicati sono principalmente a più alto valore in *valence*. In contrasto, si vede come, la componente *ener-*

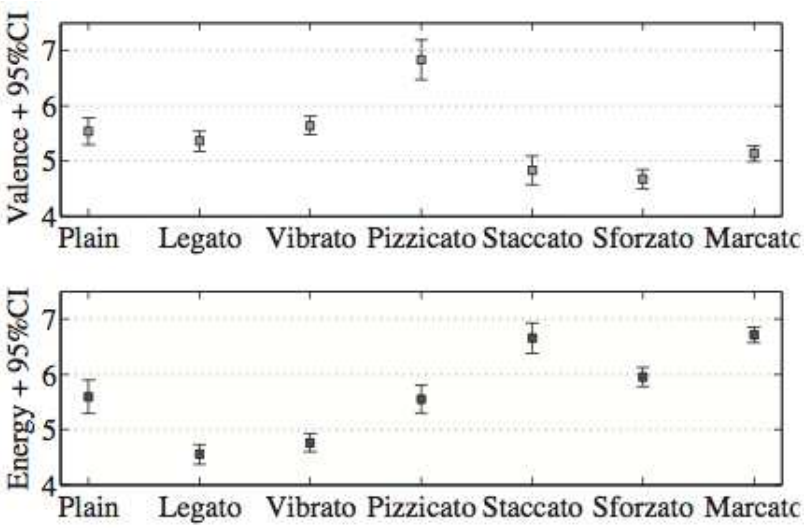

Figura 1.8: *Dipendenza da valence-energy al variare dell'articolazione Eerola et al. (2012)*

gy varia notevolmente da articolazioni di tipo sostenuto (legato, vibrato) ad articolazioni con inviluppo di tipo impulsivo (pizzicato, staccato, sforzato, marcato); caratterizzazione che non avviene per quanto riguarda la componente *valence*.

Come detto in precedenza, le intenzioni emozionali sono una combinazione di vari aspetti: indicazioni esplicitamente scritte nello spartito, tipo dello strumento ed esecuzione da parte del performer. I parametri sopracitati, essendo il mezzo per comunicare espressività, sono anch'essi combinazioni provenienti dallo spartito, strumento ed esecutore, la fusione di questi tre elementi da vita al suono; separare i fattori che influiscono sull'espressività è pertanto un'operazione non sempre facile.

1.2.2.2 Emozioni ed esecuzione

Il messaggio musicale è parzialmente codificato nello spartito. Quando si ascolta un pezzo percepiamo il contributo sia dell'esecutore, sia del compositore. Il performer del brano cerca di trasmettere, prima di tutto, la struttura imposta dalla lettura dello spartito, unendola poi a una propria interpretazione del messaggio musicale. L'esecutore, variando parametri come Tempo, articolazione, intensità

Figura 1.9: *Comunicazione emozionale dell'esecutore.*

ecc. cerca di trasmettere all'ascoltatore le proprie emozioni. Il processo è schematizzato in *Figura 1.9*. L'ascoltatore combina le varie caratteristiche, che gli vengono trasmesse, cercando di interpretare il contenuto espressivo. I parametri, contenenti espressività, derivano sia da come il suono viene prodotto, quindi dallo strumento, sia da come l'esecutore si avvale delle caratteristiche sopracitate per trasmettere emozioni.

Si elencheranno di seguito alcune importanti emozioni legandole ai parametri musicali su cui deve agire un esecutore per riuscire a comunicarle.

Happyness: Tempo veloce, poche variazioni di Tempo, alta intensità, poca variazione di intensità, staccato, alta variabilità nell'articolazione, timbro chiaro.

Anger: Tempo veloce, poca variazione di Tempo, no ritardando, alta intensità, staccato, attacchi veloci, vibrato.

Sadness: Tempo lento, finale ritardando, bassa intensità, legato, poca variazione di articolazione, attacchi lenti, leggermente vibrato.

Fear: Tempo veloce, alta variabilità di Tempo, staccato, intensità molto bassa, alta variabilità di intensità, vibrato irregolare.

Se si vuole dare un'interpretazione dal punto di vista dello spazio emozionale bidimensionale *valence-arousal*, si nota che, ad esempio: Tempo, intensità e articolazione sono principalmente legati alla componente di attivazione, quindi *arousal*; mentre il timbro, inerentemente alla sua correlazione con l'intensità, è caratterizzato da una variazione della componente *valence*. A un'intensità media e all'energia ad alta frequenza sono associate emozioni positive, mentre per valori estremi di intensità, sia alti che bassi, sono associate emozioni di tipo negativo. Si evidenzia, per esempio da Eerola et al. (2012), che alcune caratteristiche come: Tempo,

modo, timbro e articolazione sono importanti per quanto riguarda la distinzione del tipo di espressività trasmessa, definendo, ad esempio, se l'emozione è di tipo positivo o negativo; gli altri parametri sopracitati svolgono un ruolo significativo solo in una fase finale dell'inquadramento dell'emozione, apportando solo delle lievi sfumature al contenuto emotivo.

L'analisi dell'espressività, nel contempo la trasmissione dell'emozione, non è un'azione semplice da svolgere, in quanto dipende da numerose sfaccettature, esposte nella trattazione; la soggettività dell'emozione di certo non aiuta in questo compito. Sono state comunque riportate le caratteristiche generali di inquadramento dell'emozione, fermo restando che, vista la delicatezza della questione, ogni caso va trattato singolarmente per averne una caratterizzazione completa e opportuna.

1.3 Valutazione dei sistemi automatici

Una volta spiegate le relazioni che esistono tra espressività e parametri fisici dell'esecuzione e costruito un sistema che implementi il modello, ci si chiede di come venga valutata l'esecuzione automatica risultante e se sia possibile che un ascoltatore non riesca a distinguere un algoritmo che esegue il pezzo da un performer umano esperto.

1.3.1 Turing Test musicale

Si considera la possibilità che una performance automatica non venga distinta da un'esecuzione eseguita da un performer professionista reale o perlomeno venga considerata più *human-like* di una veramente eseguita da un essere umano.
In Roda et al. (2015) è riportato un esperimento di un Turing Test musicale eseguito in occasione di un concerto live di performance generate da algoritmi. Alla fine del concerto è chiesto agli spettatori di valutare quanto l'esecuzione appena ascoltata possa avvicinarsi a una umana. I parteciparti sono stati divisi in due gruppi, a uno è stato rivelato che una tra le cinque esecuzioni è eseguita da un umano, l'altro gruppo è convinto di ascoltare tutte performance provenienti da algoritmi. L'esperimento consiste nell'esecuzione di cinque pezzi appartenenti al periodo classico-romantico, eseguiti da un Disklavier; anche il pezzo eseguito da un performer umano è eseguito con tale modalità dopo averlo pre-registrato.
Il giudizio della performance è stato fortemente condizionato dalla conoscenza oppure meno della presenza di una performance umana all'interno del set di esecu-

zioni, nonostante ciò, tale esecuzione non è stata valutata come la più *human-like* delle performance neanche da spettatori esperti in ambito musicale, nella fattispecie pianisti. Il gruppo che era a conoscenza della natura delle esecuzioni, non ha espresso molti giudizi, i soggetti si sono rivelati meno propensi a mettere a confronto esecuzioni automatiche a performance eseguite da una persona. Durante il concerto live non c'è la presenza del performer sulla scena, resta ancora da investigare quanto questo possa influenzare sulla valutazione di quanto sia considerata *human-like* l'esecuzione. Le varie performance sono state ascoltate una sola volta dagli ascoltatori essendo ad un concerto live, resta da analizzare il caso in cui sia possibile riascoltare a piacimento le performance.

1.3.2 Valutazione delle performance musicali

La valutazione delle performance musicali è comune a molte pratiche musicali educative, però la ricerca che chiarisce l'insieme di fattori che influenzano tali valutazioni è ancora relativamente scarsa.
Si presentano i principali problemi che vanno ad influenzare le valutazioni sulle performance. In Shubert et al. (2014a) e Canazza et al. (2013) sono svolti due esperimenti, effettuati durante due workshops dedicati, per iniziare a capire la maniera in cui è eseguita una valutazione dell'intera performance. Gli individui sono diversi tra loro e di conseguenza reagiscono in maniera diversa a parità dello stimolo a cui sono sottoposti. Gli ascoltatori si possono classificare come *music-empathizers (ME)*, che pongono la loro attenzione sulla parte affettiva del pezzo musicale, e come *music-systemizers (MS)*, interessati alla struttura e organizzazione del pezzo musicale. Si evidenzia che le persone che forniscono una valutazione più alta a performance automatiche appartengono al secondo gruppo *(MS)*.
La prima domanda, a cui sono tenuti a rispondere i presenti alla fase finale di Rencon-SMC11[6], è atta a verificare l'aspettazione che ha un ascoltatore nei confronti di una esecuzione automatica. La maggioranza dei partecipanti (58.3%) si aspetta di ascoltare la performance come se fosse eseguita da uno studente. Viene chiesto, inoltre, di esprimere quali siano i fattori che più influenzano sulla valutazione, in modo da far riflettere i partecipanti sulla maniera in cui valutano la performance. Si possono vedere in *Figura 1.10* i fattori che maggiormente in-

[6]Rencon (Performance Rendering Contest) è un progetto di ricerca a livello mondiale nell'ambito dell'informatica musicale, il quale organizza delle gare (contest) per sistemi computazionali automatici che generano performance musicali espressive, in modo autonomo o in modo interattivo. Lo scopo è quello di selezionare il miglior sistema in grado non solo di riprodurre un brano musicale, ma soprattutto di eseguirlo aggiungendoci personalità, emotività, ed espressività.

n.	Factors influencing the judgment	Freq.
b1	is able to highlight elements related to the musical structure (phrasing, counterpoint)	77.5%
b2	contains unexpected but interesting choices	40.0%
b3	is consistent with the suitable musical style (from an historical point of view)	52.5%
b4	is consistent with the favorite style (from a subjective point of view)	12.5%
b5	is consistent with the style of a famous performer	5.0%
b6	is able to convey emotional content	57.5%
b7	contains evident musical errors (that a human would never perform)	52.5%
b8	is consistent with the actual performance context (concert hall, classroom, or automatic performance contest)	7.5%

Figura 1.10: *Fattori che influenzano maggiormente la valutazione*

fluenzano la valutazione. Si osserva che fattori di carattere tecnico, comunicativo e interpretativo predominano, fattori di tipo stilistico sono meno frequentemente considerati. Non si registrano correlazioni importanti tra la risposta a questa domanda e esperienza musicale, età o sesso. I fattori che meglio differenziano le valutazioni sono b1,b2 e b3; non si può dire la stessa cosa per b6 e b7 le quali presentano delle sovrapposizioni visibili al biplot delle analisi delle corrispondenze (*Figura 1.11*)anche se rappresentano due diversi punti di vista dell'esecuzione: tecnico vs emozionale; sono tuttavia considerati dagli ascoltatori generici come caratteristiche equamente importanti e quindi non discriminano un'esecuzione dall'altra. Il pubblico di analisti esperti, presenti al secondo esperimento, considerano gli stessi come elementi distinti tra loro e quindi idonei alla valutazione di una performance automatica. Il pubblico presente all'esperimento GATM, il secondo analizzato, è composto da musicologi italiani esperti in analisi della musica, gli spettatori risultano essere tutti *music-systemizers (MS)* e i più giovani tendono ad avere una meno negativa valutazione del valore ME. La popolazione presente ad esperimenti di questo genere dovrebbe essere mista in termini di età,

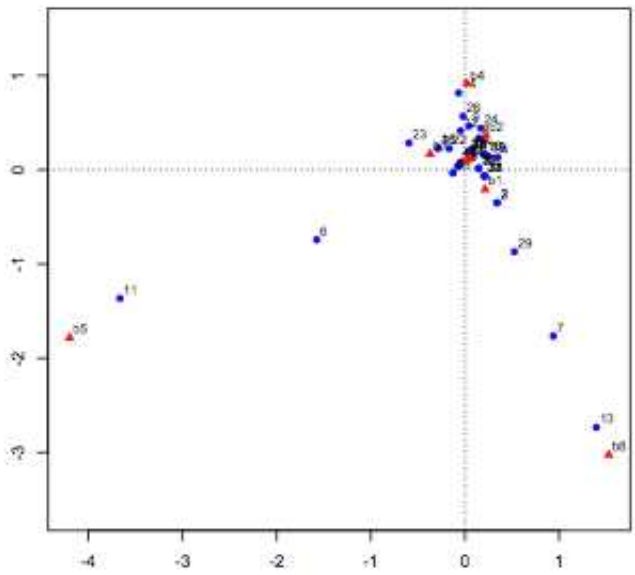

Figura 1.11: *Biplot dell'analisi delle corrispondenze sui fattori che influenzano la valutazione dei partecipanti al Recon-SMC11*

sesso, esperienza in ambito musicale, in modo da poter studiare le preferenze delle varie tipologie di ascoltatore. Si è visto precedentemente, ed è di nuovo sottolineato in Shubert et al. (2014b), come ascoltatori classificati come *music-systemizers (MS)* valutino la performance in modo diverso dagli *music-empathizers (ME)*, se la popolazione è uniforme si tende ad avere una valutazione che considera solo un punto di vista. La valutazione di una performance dipende fortemente dal livello di concentrazione dell'ascoltatore,come evidenziato in Shubert et al. (2014b), livello che risulta essere maggiore nella categoria dei *music-systemizers (MS)*. La valutazione degli stimoli è influenzata dalla durata e dal numero di pezzi musicali e compiti richiesti. Si nota che lo stesso pezzo musicale è valutato in maniera differente se presentato più volte. Si evidenzia, pertanto, che il giudizio sulle performance è sensibile anche al grado di stanchezza posseduta dall'ascoltatore; la valutazione di un'esecuzione non è un'operazione facile in quanto influenzata da fattori non deterministici e difficili da individuare. Emergono un numero importante di conclusioni dagli studi appena presentati. Per primo è che sarebbe necessario un approccio più globale e interdisciplinare, che cerchi di chiarire non solo gli strumenti di misura e i contesti musicali nei quali possono avvenire le performance, ma anche altri fattori come la personalità, e i pregiudizi socia-

li e culturali che influenzano la valutazione del giudice. Inoltre la conoscenza da parte del performer dei criteri utilizzati per la valutazione, può portare ad un miglioramento della performance. Restano ancora irrisolte alcune questioni:

- fino a che punto un giudice sia abile a fornire una valutazione affidabile sotto contesti e scopi diversi (esame di musica, audizione, competizione ecc.);
- in quali situazioni e in che modo il training può aiutare ad alleviare alcuni dei problemi che minano l'affidabilità dei giudici.

Dal un punto di vista del progresso tecnologico, questi giudizi sono utili, non soltanto per valutare i sistemi, ma anche per fornire indicazioni ai sviluppatori e ricercatori su quale sia la direzione per un miglioramento futuro. Specifiche misurazioni delle performance possono rivelare punti di debolezza sotto il profilo tecnico, questo è solo un primo miglioramento che si può apportare. Gli studi in questione aiutano a capire la correlazione che sussiste tra arte e tecnologia.

Capitolo 2

Implementazione di CaRo 2.0

Il programma CaRo, arrivato già alla versione 2.0, è stato implementato utilizzando come IDE il C++ Builder, il suo funzionamento era quindi limitato a macchine con sistema operativo di tipo Windows. Prima di poter implementare qualsiasi miglioramento, è stato necessario effettuare un porting dell'applicazione, utilizzando un editor e librerie che possano funzionare su tutte le piattaforme, per rendere il programma fruibile anche in futuro per successivi miglioramenti. Si può considerare questa come la versione 2.1 dell'applicazione.

2.1 Il modello dell'applicazione CaRo

L'applicazione CaRo è un sistema informatico in grado di eseguire una partitura musicale in maniera espressiva. Come ricordato nel capitolo precedente, il performer cerca di trasmettere, prima di tutto, la struttura imposta dalla lettura dello spartito, unendola poi a una propria interpretazione del messaggio musicale. L'esecutore, variando parametri come Tempo, articolazione, intensità ecc. cerca di trasmettere all'ascoltatore le proprie emozioni. Il software CaRo cerca di eseguire un brano musicale in maniera espressiva. La struttura base del brano, inclusi i segni di dinamica, è acquisita direttamente dalla partitura. La componente espressiva è impartita dall'utente tramite uno spazio di controllo, come si può vedere da *Figura 2.1*. Per ottenere una trasformazione della performance sulla base di differenti intenzioni espressive, è stato sviluppato uno spazio di controllo astratto chiamato *perceptual parametric space* (PPS). Il PPS consiste in uno spazio bidimensionale derivato da un'analisi multidimensionale sui risultati di test percettivi effettuati su performance musicali professionali. Questo spazio rispecchia come

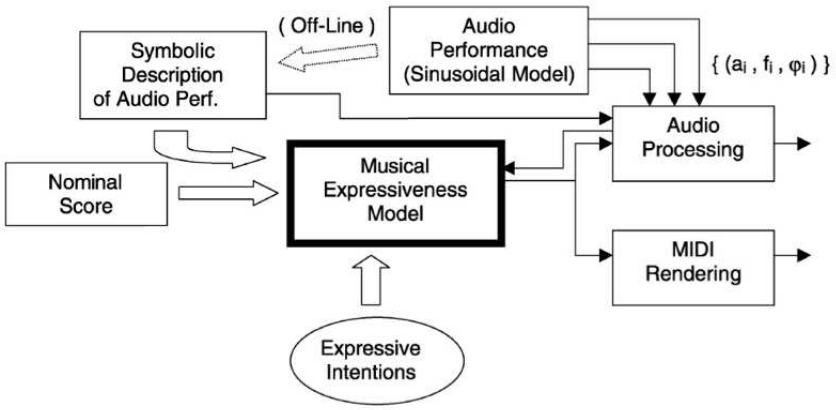

Figura 2.1: *Modello utilizzato da CaRo*

la performance musicale è organizzata nella mente di un ascoltatore: è stato provato che gli assi del PPS sono correlati ai valori acustici e musicali percepiti dagli ascoltatori stessi Canazza et al. (2011). Le componenti dello spazio sono *kinetic* e *energy*, in posizioni opportune sono collocati degli aggettivi per aiutare l'utente a dare la corretta espressività. Il software, oltre all'esecuzione di tipo espressivo, è in grado di eseguire il brano in maniera meccanica, escludendo, quindi, sia l'input dell'utente sia i segni di dinamica espressi nello spartito. La terza opzione è l'esecuzione di tipo neutro, si definisce così un'esecuzione umana senza una precisa intenzione espressiva, riprodotta in modo scolastico e senza nessun scopo artistico.

2.2 Miglioramenti rispetto la versione precedente

L'applicazione è stata migliorata in diverse direzioni. Come detto in precedenza il software allo stato attuale gira su qualsiasi piattaforma, testato largamente su MacOsx, ma compatibile con qualsiasi altro sistema operativo, grazie all'utilizzo di librerie multipiattaforma e l'ausilio di un generico editor per la scrittura del codice stesso. Verrano esplicitate in seguito le librerie usate per lo sviluppo. La versione precedente del programma necessitava, per la performance di tipo espressiva, due tipi di input da dare al programma. Un file formato MIDI, il quale conteneva le informazioni base sulle note, come ad esempio l'altezza e la durata, e un corrispondente file formato XML, che conteneva le espressività indicate nello spartito.

In fase di esecuzione del brano c'era la necessità di trovare la corrispondenza tra gli eventi musicali in formato MIDI e le rispettive dinamiche presenti nel file xml. La sincronizzazione tra i vari eventi corrispondenti era fondamentale per la buona riuscita della performance, non sempre l'esecuzione andava a buon fine, c'era il bisogno di modificare parti salienti della partitura per garantire un'esecuzione fluida. Nella versione corrente dell'applicazione è sufficiente caricare un unico file, quello XML, il programma è in grado di ricavare tutte le informazioni presenti nella partitura; sono gestiti, inoltre, tutte le casistiche di partitura, non c'è più necessità di procedere a modificare parti della composizione in fase preliminare. Il parsing del file XML è stato migliorato notevolmente, precedentemente il file veniva analizzato riga per riga come file di testo generico, senza sfruttare, quindi, la sua naturale struttura ad albero con chiave rappresentata dai tag XML. Le modifiche apportate hanno riaffiorato problematiche, che nella versione precedentemente implementata non erano presenti, per l'aiuto dato dall'uso del file MIDI abbinato. Nei prossimi paragrafi verranno spiegate in dettaglio le soluzioni adottate per lo sviluppo. Il codice è stato rigorosamente commentato in ogni sua parte per favorirne la leggibilità ed estensione futura.

2.3 Librerie utilizzate

L'applicazione fa uso di tre tipi di librerie, al fine di renderlo il più portabile possibile:

WxWidgets: libreria utilizzata per l'interfaccia grafica del programma.

MidiShare: libreria utilizzata per la gestione e costruzione degli eventi MIDI e conseguentemente della loro esecuzione audio.

LibMusicXML2: libreria utilizzata per il parsing del file XML.

2.3.1 WxWidgets

WxWidgets[1] è una libreria scritta in linguaggio C++ che permette di realizzare l'interfaccia grafica di un programma e di gestirne i vari eventi ad essa collegata. WxWidgets attualmente supporta le seguenti piattaforme:

[1]Installazione provata su MacOsX: installazione da HomeBrew digitando da terminale *brew install wxmac*. Per la compilazione e per le altre piattaforme seguire le indicazioni riportate nel sito ufficiale *https://www.wxwidgets.org*

wxGTK: un port raccomandato per Linux e altre varianti Unix, usare GTK+ version 2.6 o superiore.

wxMSW: un port per Windows 32-bit e 64-bit incluso Windows 2000/NT/XP, Windows 7, e Windows 8.

wxMac: per eseguire Carbon applications su Mac OS X 10.2 attraverso 10.6.

wxOSX/Carbon: per eseguire 32-bit Carbon-based applications su Mac OS X 10.5 e superiore.

wxOSX/Cocoa: per eseguire 32-bit and 64-bit Cocoa-based applications su Mac OS X 10.5 e superiore.

wxX11: un port per Linux and Unix che fanno uso di X11.

wxMotif: un port for Linux and Unix che usano OpenMotif o Lesstif.

Sono state scelte le librerie per la grafica WxWidgets per vari motivi: multipiattaforma, Windows, Unix, Mac OS X, e altre; non c'è la necessità di scrivere diverse versioni del software; aspetto gradevole, stabilità, performance alte e ricche di funzioni; comunità attiva di sviluppatori per il supporto.

2.3.2 MidiShare

La libreria MidiShare è stata creata nel 1989 per sopperire al problema dell'implementazione di software real-time musicale. Essa fornisce un pacchetto ad alto livello di servizi per lo sviluppo che assicura l'indipendenza dalla piattaforma, supportando infatti GNU/Linux, MacOS X e Windows. Disponibile ormai da diversi anni in forma gratuita del Development Kit, in una grande varietà di linguaggi: C, C++, Common Lisp, Pascal, Java. Oltre alle librerie MidiShare sono fornite in aggiunta altre due librerie per facilitare lo sviluppo di applicazioni musicali: una MIDI files management library e una Player library con l'intento di sviluppare sequenze MidiShare multi traccia. Un Player usa sequenze MidiShare, e fornisce funzioni per leggere e scrivere file MIDI e funzioni per convertirli nel formato delle sequenze MidiShare.

Una volta installate le librerie MidiShare, reperibili al sito *midishare.sourceforge.net*, è necessario procedere a configurare correttamente alcuni parametri. Il comportamento dei drivers è controllato usando i file INI, che sono simili a dei file di testo con le istruzioni scritte al loro interno. Ogni driver ha degli slot di input e di

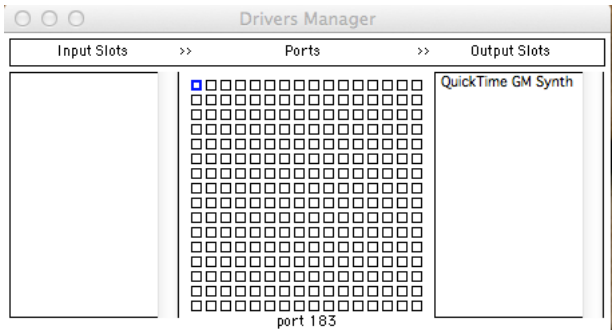

Figura 2.2: *Applicazione msDrivers*

output, che corrispondono alle porte di input e output che vengono usate. Per impostare correttamente le istruzioni presenti all'interno dei file INI basta usare l'applicazione msDrivers, un Drivers Manager che permette di settare le connessioni tra porte MidiShare e gli slot del driver. L'interfaccia dell'applicazione si presenta come in *Figura 2.2*. Per un corretto funzionamento è sufficiente scegliere all'interno della matrice la porta che si vuole associare all'output desiderato. Premere, quindi, una casella e l'output ad essa associato, sarà possibile, così, indirizzare gli eventi midi al player del computer oppure direttamente al Disklavier.

2.3.3 LibMusicXML2

La libreria MusicXML è scritta in C++, fornisce un supporto alla gestione del formato MusicXML. La libreria è un progetto open source presente al sito *libmusicxml.sourceforge.net*, consente a una più agile conversione dalla rappresentazione della memoria al formato MusicXML. La libreria risulta essere multi piattaforma. Nel paragrafo successivo verrà spiegata la struttura base del formato MusicXML. La libreria viene fornita già in formato framework, per l'installazione è sufficiente copiarlo nella cartella di defalut di sistema che contiene i framework; nel caso di MacOs X copiare libmusicxml2.framework in /Library/Frameworks.

2.3.4 Formato MusicXML

MusicXML[2] è un sistema di codifica XML che è in grado di rappresentare il sistema di notazione musicale occidentale dal diciassettesimo secolo in poi, ovvero

[2]informazioni reperite all'indirizzo *http://it.wikipedia.org/wiki/MusicXML*

quello attualmente utilizzato. La codifica è stata promossa e sviluppata da Recordare LLC, un'azienda americana che si occupa di Internet publishing per edizioni digitali di spartiti musicali. MusicXML è un formato libero, rilasciato sotto una licenza senza diritto d'autore (royalty-free), che consente l'uso e la modifica del programma a chiunque, purché si specifichi che la proprietà intellettuale del prodotto originario appartiene a Recordare LLC e, in caso di modifica, si mostri all'utente la licenza rilasciata sul sito ufficiale.

MusicXML, in quanto codifica XML, offre tutte le potenzialità di questo strumento:

1. strutturazione dei dati
2. modularità
3. estensibilità
4. scambio di dati back-office
5. possibilità di interrogazione e interazione attraverso la famiglia di tecnologie legate a XML

Lo scopo e l'uso attuale di MusicXML è l'interscambio di spartiti musicali su Internet. Negli ultimi trent'anni sono stati realizzati numerosi sistemi di rappresentazione della notazione occidentale, ma l'unico utilizzato su larga scala è il MIDI: Musical Instrumental Digital Interface. Mentre il formato MIDI nasce come supporto alle performance musicali, MusicXML si propone come uno standard di codifica dello spartito musicale in tutte le sue sfaccettature. Una volta codificato, lo spartito può essere considerato come un database semi-strutturato e dunque interrogato e rielaborato. Uno dei programmi più famosi è Finale, software che consente la creazione di uno spartito musicale anche complesso e che ne consente l'esportazione in vari formati tra cui MusicXML. Finale è stato usato durante lo sviluppo dell'applicazione CaRo per la generazione degli spartiti utilizzati per il testing.

Esistono due tipi di codifiche per rappresentare uno spartito musicale, MusicXML presenta due diverse Document Type Definition (DTD) principali e ognuna ha un elemento root diverso:

score-partwise: dove le battute sono contenute nelle parti

score-timewise: dove le parti sono contenute nelle battute

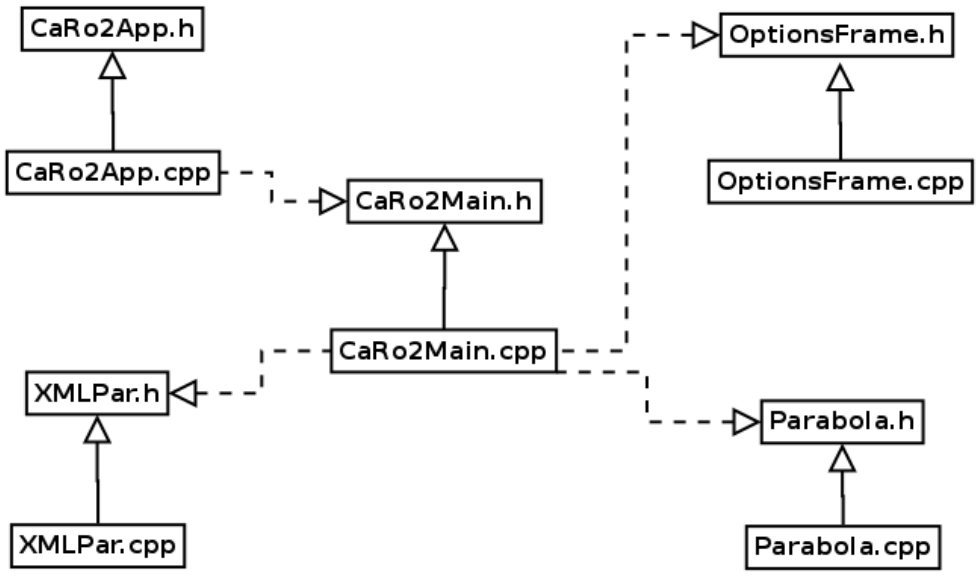

Figura 2.3: *Struttura delle classi*

Le due codifiche sono convertibili l'una nell'altra attraverso un programma XSLT fornito da Recordare. L'applicazione CaRo è stata testata utilizzando partiture codificate in maniera *score-partwise*. Sono lette, quindi, tutte le prime battute di tutti i pentagrammi presenti e poi si procede alla lettura delle battute successive. I tag più importanti utilizzati nel progetto sono: **note** e **measure**, le quali delimitano, rispettivamente, un elemento di tipo nota e uno di tipo battuta. I *Document Type Definition (DTD)* completi sono disponibili nel sito *www.musicxml.com/for-developers*.

2.4 Organizzazione delle classi

Le classi utilizzate per lo sviluppo del progetto sono state cambiate e rivisitate. Sono riportate nella *Figura 2.3* la struttura e le relazioni tra le classi. La classe *XMLPar* è usata per la gestione di un singolo elemento che rappresenta una nota, memorizzando le informazioni base e sulle dinamiche; la classe *Parabola* è usata in fase d'esecuzione di un brano per gestire in maniera graduale le dinamiche che occorrono tra un evento musicale e l'altro. La classe che contiene

il cuore dell'applicazione è *CaRo2Main*, si occupa, infatti, della gestione eventi, creazione interfaccia grafica, parsing dei file, performance in real-time. Le opzioni dell'applicazioni personalizzabili grazie alla finestra di input gestita dalla classe *OptionsFrame*. La classe *CaRo2App* permette la creazione di una nuova finestra dell'applicazione ogni volta che viene lanciata l'applicazione. Nei paragrafi successivi verranno spiegate più in dettaglio le varie classi e presentate le parti del codice più salienti.

2.4.1 Classe XMLPar

Il file MusicXML comporta l'esigenza di una struttura dati che sia in grado di estrarre le informazioni inserite in questo tipo di file, di elaborarle, e di utilizzarle al momento opportuno. Per questo fine è stata creata una classe, chiamata *XMLPar*, che offre una semplice struttura dati per organizzare e memorizzare tali informazioni, tale struttura è stata solo parzialmente modificata rispetto alla versione presente nella tesi di Massimiliano Barichello [3], è riportato di seguito quanto da lui spiegato con le modifiche opportune ove necessario. La scelta del tipo di dati da adottare per il salvataggio, e i metodi da utilizzare per l'estrapolazione e l'applicazione di tali informazioni, ha reso necessario una minima conoscenza delle informazioni musicali che possono essere aggiunte da Finale allo spartito musicale. Qui di seguito verranno illustrati tutti i componenti e le procedure che permettono a CaRo 2.0 di utilizzare i file *MusicXML*. *XMLPar* è un header file che contiene i prototipi delle funzioni e delle variabili che vengono utilizzate dalla classe *XMLPar*. In CaRo, ad ogni nota dello spartito musicale viene associato un oggetto di tipo XML. Per questo motivo XML.h ha la seguente struttura:

```
class XMLpar{
   private:
      int ID;
      int beatPos;
      int respiro;
      int accento;
      int dinamica;
      int tenuto;
      int staccato;
      int pedale;
      int legato[10][3];
```

[3] http://tesi.cab.unipd.it/35067/1/tesiBarichello.pdf

```
    int numLegato;
    int ritardando[2];
    char nota;
    int pitch;
    int chan;
    long dur;
public:
    XMLpar();
    int getID();
    void setID(int i);
    int getBeat();
    void setBeat(int i);
    int getRespiro();
    void setRespiro(int r);
    int getAccento();
    void setAccento(int a);
    int getTenuto();
    void setTenuto(int te);
    int getStaccato();
    void setStaccato(int d);
    int getLegatoNumber(int riga);
    int getLegatoStart(int riga);
    int getLegatoStop(int riga);
    int getRitStart();
    int getRitStop();
    void setLegatoNumber(int riga, int n);
    void setLegatoStart(int riga, int start);
    void setLegatoStop(int riga, int stop);
    void setRitStart();
    void setRitStop();
    int getNumLegato();
    void setNumLegato(int num);
    char getNota();
    void setNota(char n);
    int getDinamica();
    void setDinamica(int d);
    int getPedale();
    void setPedale (int p);
```

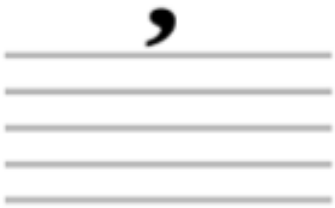

Figura 2.4: *Respiro*

```
        int getPitch();
        void setPitch(int p);
        int getChan();
        void setChan(int c);
        long getDur();
        void setDur(long d);
};
```

ID L'ID è l'identificativo della nota, che la distingue dalle altre note. Ad ogni nota dello spartito viene associato un numero intero partendo da 0.

Beatpos Posizione metrica della nota, tempo in cui occorre la nota.

respiro In inglese è detto anche Breath Mark, e il suo simbolo è una virgola posta appena dopo la nota interessata (vedi *Figura 2.4*). Negli strumenti a fiato o a voce, il respiro indica un momento in cui si può prendere respiro, mentre per gli strumenti non a fiato indica il momento in cui si può prendere una breve pausa. Nel nostro caso tale variabile può assumere due valori: 1 se la nota corrispondente ha un respiro posto appena dopo, 0 altrimenti.

accento L'accento dinamico viene indicato graficamente con la parentesi angolata (accento orizzontale o accento lungo), e viene posto sopra o sotto una nota (vedi *Figura 2.5*). La nota con l'accento deve essere eseguita con una intensità sonora più forte. La variabile accento vale 1 se la nota corrispondente ha un accento, 0 altrimenti.

dinamica In musica le dinamiche sono dei simboli posti nello spartito che in-

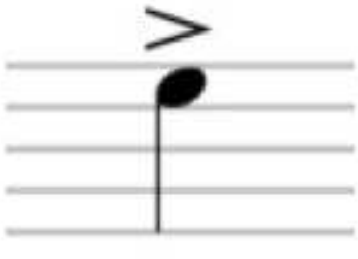

Figura 2.5: *Accento*

valore	simbolo	significato
0	nessuna dinamica	
1	pp	pianissimo (very soft)
2	p	piano (soft)
3	mp	mezzo piano (moderately soft)
4	mf	mezzo forte (moderately loud)
5	f	forte (loud)
6	ff	fortissimo (very loud)

Figura 2.6: *Dinamiche*

dicano con quale intensità devono essere eseguite le note successive. Nel pentagramma vanno messi nel punto dopo il quale vogliamo che le note vengono eseguite con quell'intensità specificata dalla dinamica. Nel programma vengono gestite solo alcune tra le tante dinamiche disponibili. La variabile può contenere i valori riportati in *Figura 2.6*.

tenuto Nel pentagramma è indicato con un trattino posto sopra alla nota (vedi *Figura 2.7*). Con il tenuto la nota diventa più lunga e un po' più forte (come un leggero accento). Tale variabile vale 1 in presenza dell'accento nella nota corrispondente, 0 altrimenti.

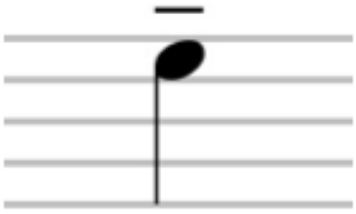

Figura 2.7: *Tenuto*

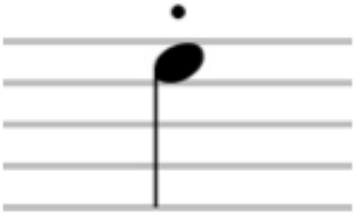

Figura 2.8: *Staccato*

staccato Detto anche Dot Staccato. Viene indicato con un puntino posto sopra o sotto ad una nota (vedi *Figura 2.8*). Tale simbolo indica che la nota è separata o sconnessa dalle note successive, quindi richiede un'esecuzione più breve della nota, normalmente la metà della sua durata originale.

pedale La funzione principale del pedale è quella di lasciar risuonare una nota anche dopo aver abbandonato il tasto. Nel programma l'informazione del pedale viene associata alla nota corrispondente oppure se non possibile, poiché associato ad una pausa, è associata alla nota di un altro rigo immediatamente successiva o in corrispondenza al simbolo start-pedale o stop-pedale inserito nello spartito. La variabile può assure i valori riportati in *Figura 2.9*.

legato[10][3] Detto anche Slur. Nel pentagramma è indicato con un arco tra due note (anche con pitch diverso), o tra un gruppo di note. Un'altra espressione musicale che viene indicata con lo stesso simbolo è il Tie, un arco che collega due note con lo stesso pitch. Il programma CaRo non memorizza in nessuna struttura le informazioni di eventuali tie inseriti nello spartito, ma gestisce comunque

valore	significato
0	nessun pedale
1	start pedale
2	stop pedale
3	start pedale e stop pedale nella stessa nota

Figura 2.9: *Pedale*

la presenza di tie al momento della lettura del file MusicXML, incrementando la durata della nota da cui inizia la legatura. Tie e legato indicano che l'esecuzione deve essere smoothly, cioè le note di una frase melodica devono essere eseguite accostate l'una all'altra senza interruzioni, in modo che il suono dell'una inizi non appena cessa quello della nota precedente. La variabile legato memorizza solo le informazioni riguardante gli slur inseriti nello spartito musicale. Tale variabile è una matrice 10x3 che viene abbinata ad una nota nella quale inizia o finisce un legato (se il legato raggruppa più di due note, le note intermedie non avranno nessuna matrice associata). La scelta di una matrice come variabile risiede nel fatto che da una nota possono partire o finire più legato contemporaneamente. Ad ogni riga della matrice corrispondono le informazioni di uno solo dei legato presenti in quella nota. Nella prima colonna viene indicato il number (l'id) del legato in quella nota, nella seconda colonna viene messo 1 se il legato parte dalla nota (0 altrimenti), e nella terza colonna viene messo 1 se il legato finisce in quella nota (0 altrimenti).
Se ad esempio dalla nota partono 3 slur e terminano 2 slur (vedi *Figura 2.10*), allora la matrice è composta da 5 righe. I number di ogni riga sono estratti dal file MusicXML, e di solito sono dei numeri interi (quindi essendo nell'esempio all'i-

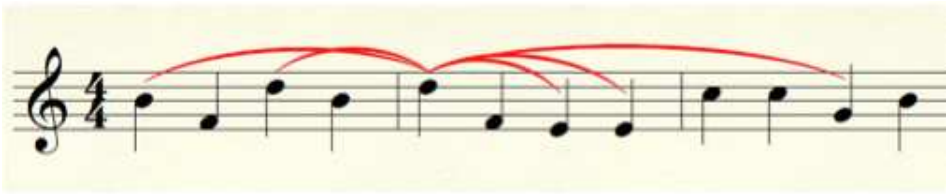

Figura 2.10: *Esempio di note con dei legato*

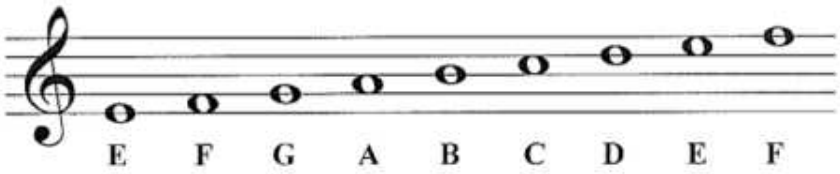

Figura 2.11: *Pitch e relative lettere*

nizio dello spartito i number andranno da 1 a 5). La matrice supporta al massimo 10 slur per ogni nota; in caso di necessità basta aumentarne le dimensioni.

ritardando Vettore che memorizza inizio e fine di un ritardando, [1 0] inizio ritardando in corrispondenza nella partitura della dicitura *rit.*, [0 1] fine ritardando in corrispondenza della dicitura *a tempo* o a fine partitura. La durata delle note comprese in tale intervallo aumenta progressivamente raggiungendo valore massimo alla fine del ritardando. Viene usato un vettore per la rappresentazione poiché un valore binario non era sufficiente, con il valore 0 non si riusciva a capire se s'intende fine ritardando o ritardando non espresso.

numLegato Tale variabile contiene il numero di legato presenti nella nota corrente. Questa informazione diventa utile nel momento in cui si vuole scandire la matrice dei legato (ad esempio per stamparne il contenuto). **nota** Contiene il valore del pitch della nota. Il pitch indica la posizione in verticale della nota nello spartito musicale, e nel file MusicXML viene indicato attraverso una lettera maiuscola (vedi *Figura 2.11*). La memorizzazione di questo tipo di informazione non è usata, è usato il valore del pitch in formato midi.

pitch Valore intero che tiene conto dell'altezza della nota (lettera) ma anche dell'ottava di appartenenza della stessa, nonché di eventuali alterazioni presenti in chiave o in locale. Il valore è espresso concordemente al formato midi, subito

interpretabile dal player. Si riporta di seguito la funzione che svolge la funzione di trasformazione da nota (lettera) a valore midi della stessa.

```
float notevisitor::getMidiPitch() const
{
    if (fType == kPitched) {
        int step = step2i(getStep());
        if (step >= 0) {
            short step2pitch [] = { 0, 2, 4, 5, 7, 9, 11 };
            float pitch = (getOctave() * 12.f)
                                    + step2pitch[step];
            return pitch + getAlter();
         }
     }
    return -1;
}
```

durata Contiene la durata della nota espressa in tick. Si riporta codice per la conversione. Durante l'esecuzione del brano la durata di ogni nota dovrà essere espressa in ms.

```
long convert2Tick(long val) { return (val*PPQ)/fDivisions; }
```

Pulses Per Quarter note (PPQ), indica il numero di tick che ci sono per una nota da quarto (o semiminima, o quarter note). Il tick è l'unità di misura del tempo interna al Player. *fDivisions* indica il valore della nota più breve presente in quella determinata battuta. *val* indica la durata della nota rispetto alla nota più breve, se si tratta della nota più breve assumerà valore 1, negli altri casi multipli.

canale Per l'esecuzione è necessario esprimere il canale su cui si vuole trasmettere l'evento musicale, tale informazione è memorizzata in questa variabile.

Nella parte pubblica della classe XMLPar oltre al costruttore ci sono un elenco di funzioni che permettono l'estrazione e l'inserimento dei valori delle variabili private sopra citate. Tali metodi sono implementati nel XMLPar.cpp che include l'header file XMLPar.h. La parte implementativa della classe è molto semplice: il costruttore pone a 0 tutte le variabili (compresi gli elementi della matrice), i metodi get ritornano il valore della variabile, mentre i metodi set impostano il valore della variabile corrispondente in base al valore passato come parametro.

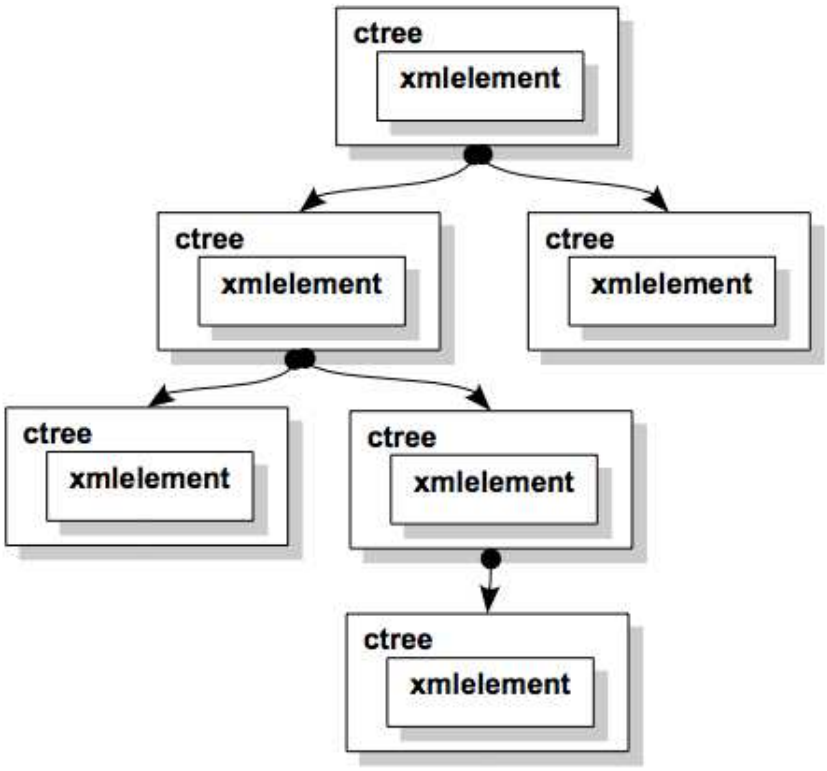

Figura 2.12: *Struttura albero di navigazione*

2.4.2 Parser del file MusicXML

Al fine di eseguire un brano è necessario preliminarmente eseguire il parse dello spartito inserendo le varie note in oggetti distinti XMLPar, tali oggetti verranno inseriti in un vettore dinamico che verrà ordinato in senso crescente rispetto all'informazione temporale di occorrenza delle varie note. Il metodo che avvia l'operazione coi parsing è *CaRo2Frame::OnOpenParseMusicXML(wxCommandEvent& event)* contenuto nella classe principale *CaRo2Main.cpp*, la funzione, tramite un dialogo con l'utente, seleziona il file MusicXML per il parsing. Una volta valutata la bontà del file viene costruito un albero di navigazione del file XML,come si può vedere in *Figura 2.12* , mantenendo la naturale gerarchia imposta dalla struttura

tipica di questo tipo di file. La navigazione dell'albero viene gestita dalla classe *midicontextvisitor*, presente nella libreria LibMusicXML2, ma reimplementata all'interno della classe *CaRo2Main.cpp*. Si riporta il codice di costruzione e navigazione dell'albero al fine del parsing:

```
mywriter writer;
mycontext v(PPQN, &writer);
unrolled_xml_tree_browser browser(&v);
browser.browse(*elt);
```

L'oggetto *mywriter* è la reimplementazione della classe *midiwriter*, presente nella libreria LibMusicXML2, adattata alle esigenze dell'applicazione; mycontext invece, come detto prima, è la reimplementazione della classe *midicontextvisitor*. Il metodo *browse* da il via alla navigazione dell'albero partendo dalla radice. Implementando vari metodi è possibile fermare la navigazione dell'albero all'incontrare o alla fine di determinati tag del file MusicXML. I tag d'interesse sono quelli riferiti alle battute e alle note, rispettivamente *mesaure* e *note*.

Parsing di una nota

Riportiamo di seguito l'implementazione del metodo che colleziona le varie informazioni riferite ad una nota:

```
void visitEnd ( S_note& elt )
{
float dur=0; //cerco la durata dell'elemento corrente
        vector<Sxmlelement>::iterator div;
        for(div = elt->elements().begin();
         div != elt->elements().end();div++)
        {
            if((*div)->getName().compare("duration")==0)
            {
                dur = (stol((*div)->getValue())*PPQN)/fDivisions;
                break;
            }
        }
        if(getType() != notevisitor::kRest
         && getType() != notevisitor::kUndefinedType)
        {

            XMLpar *n = new XMLpar();
```

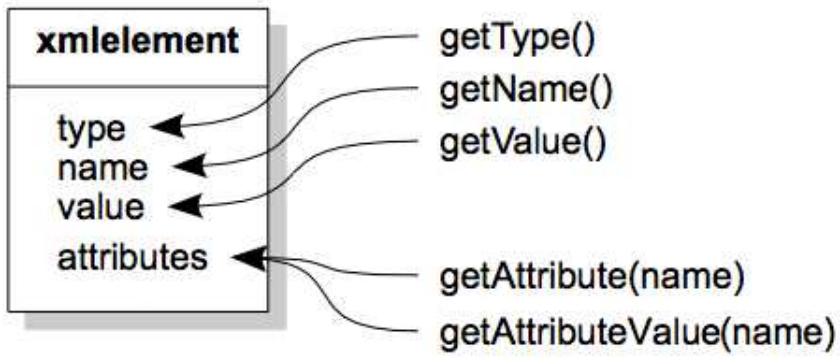

Figura 2.13: *Struttura singolo nodo*

```
resStream <<"newNote ";

if(inChord())
{
    n->setBeat(floor((float)(tot_date-dur)
     *tick2ms+0.5));
    resStream<<"inChord ";
}
else
{
    n->setBeat(floor((float)tot_date*tick2ms+0.5));
    tot_date=tot_date+dur;
    resStream<<"date "<<n->getBeat()<<" ";
}

n->setID(contaNoteIn); //set id nota
```

La navigazione, in questo caso si ferma all'incontro del tag termine nota; in questa prima parte del metodo viene da subito ricavata la durata della nota dal parametro elt di tipo S_note e riportato in tick. Ogni nodo dell'albero di navigazione è strutturato come si può vedere dalla *Figura 2.13*, il valore degli attributi può essere letto usando gli appositi metodi. La struttura dei nodi dell'albero è identica, qualsiasi sia la natura del nodo, si troverà, quindi, la stessa struttura riportata in

figura anche in corrispondenza di un elemento di tipo battuta; cambieranno solo i valori contenuti nei vari attributi dell'oggetto e la lista degli attributi.
Dopo aver ricavato la durata dell'evento si controlla che sia realmente una nota, in questo caso, è creato un nuovo oggetto XMLPar atto a contenere le varie informazioni, se la nota non fa parte di un accordo, il tempo globale, memorizzato nella variabile *tot_date* è incrementato. Alla nota è settato un id progressivo.

```
switch (fCurrentDynamics)
            {
                case 36: //pp
                    n->setDinamica(1);
                    break;
                case 48: //p
                    n->setDinamica(2);
                    break;
                case 62: //mp
                    n->setDinamica(3);
                    break;
                case 74: //mf
                    n->setDinamica(4);
                    break;
                case 88: //f
                    n->setDinamica(5);
                    break;
                case 100: //ff
                    n->setDinamica(6);
                    break;

                default:
                    n->setDinamica(0);
                    break;
            }
            resStream<<"dinamics "<<fCurrentDynamics<<" ";
            n->setPitch(getMidiPitch() + fTranspose);
            resStream<<"pitch "<<getMidiPitch() + fTranspose<<" ";
            n->setChan(0); //set canale
            n->setDur(floor((float)dur*tick2ms+0.5));
            resStream<<"dur "<<n->getDur()<<" ";

            if(fBreathMark!=NULL)
            {
```

```
            n->setRespiro(1);
            resStream<<"breath ";
        }
        if(fAccent!=NULL)
        {
            n->setAccento(1);
            resStream<<"accent ";
        }
        if(fTenuto!=NULL)
        {
            n->setTenuto(1);
            resStream<<"tenuto ";
        }
        if(fStaccato!=NULL)
        {
            n->setStaccato(1);
            resStream<<"staccato ";
        }
```

Il metodo continua con l'arricchire oggetto XMLPar di tutte le parti spiegate nel paragrafo precedente. Si identifica la dinamica della nota facendola corrispondere a un numero da 0 a 6; si setta il midiPitch, il canale e la durata portata in ms[4]. Se presenti si impostano nell'ordine: il respiro, l'accento, il tenuto e lo staccato.

```
//inserimento nella matrice legature di tipo slur
        int k=0; //indice riga matrice
        std::vector<S_slur>::const_iterator i;
        for (i = getSlur().begin(); i != getSlur().end(); i++)
        {
            resStream<<"slur: "<<(*i)
                                ->getAttributeValue("type")
            if(((*i)->getAttributeValue("type")
             .compare("start"))==0)
            {
                n->setLegatoNumber(k, (*i)
                 ->getAttributeIntValue("number",1));
                n->setLegatoStart(k, 1);
```

[4]In generale il coefficiente che permette di passare da tick a ms è pari a tick2ms=60000/(PPQ*Tempo); il valore del tempo è acquisito grazie alla reimplementazione della classe *midiwriter*, responsabile a ridefinire la viabile globale *tick2ms* ad ogni indicazione di cambio Tempo presente nello spartito

```
                n->setLegatoStop(k,0);
            }
            else if(((*i)->getAttributeValue("type")
             .compare("stop"))==0)
            {
                n->setLegatoNumber(k, (*i)
                 ->getAttributeIntValue("number",1));
                n->setLegatoStart(k, 0);
                n->setLegatoStop(k,1);
            }
            k++;
        }
        n->setNumLegato(getSlur().size());
        resStream<<"#slur "<<n->getNumLegato()

        n->setNota(getStep().at(0)); //lettera nota
```

Si passa ora all'inserimento delle legature di tipo slur collezionate all'interno della matrice, segue il set del numero delle slur e della lettera corrispondente alla nota.

```
int tie = getTie(); //numero legature di valore

        if (tie == StartStop::undefined)
        //non è definita nessuna legatura di valore,
        //posso inserire l'ogg nel contenitore
        {
            contenitore.push_back(*n);
            contaNoteIn++;
        }
        else if (tie & StartStop::start)
         //inizio di una legatura di valore
        {
            bool trovato=false;
            for(int h=0;h<nodePending.size() && !trovato;h++)
            //controllo se è una continuazione
            //di una legatura di valore, per più di due note
            {
                if(nodePending.at(h).getPitch() ==
                 n->getPitch())
                {
```

```
                nodePending.at(h).setDur(
                 nodePending.at(h)
                .getDur()+n->getDur());
                n->setDur(0);
                contenitore.push_back(*n);
                contaNoteIn++;
                trovato = true;
            }
        }
        if(!trovato) //legatura appena iniziata,
        //viene sospesa l'inserimento in
        //attesa del termine legatura
        {
            nodePending.push_back(*n);
            contaNoteIn++;
        }
        resStream<<"tie start ";

    }
    else if (tie == StartStop::stop)
    //termine della legatura di valore
    {
        for(int h=0;h<nodePending.size();h++)
        //cerco la legatura di valore che termina
        {
            if(nodePending.at(h).getPitch() ==
             n->getPitch())
            {
                nodePending.at(h).setDur
                 (nodePending.at(h)
                 .getDur()+n->getDur());
                contenitore.push_back(
                 nodePending.at(h));
                //inserimento nel contenitore
                nodePending.erase (
                 nodePending.begin()+h);
                //cancellazione dalle note sospese
                n->setDur(0);
                //set durata a 0 della nota legata
                contenitore.push_back(*n);
```

```
                    //inserimento nel contenitore
                    //della nota legata per salvare
                    //le possibili dinamiche
                    //espresse per quella nota,
                    //in particolare eventuali inizio/fine parabole
                    contaNoteIn++;
                    break;
                }
            }
            resStream<<"tie stop ";
        }
          resStream<<endl;
      }
```

Sono ora gestite le legature di valore, potenzialmente presenti tra note dello stesso pitch, se la legatura non viene trovata la nota non rimane in sospeso, viene immediatamente inserita nel contenitore definitivo. Nel caso in cui venga identificato l'inizio di una nuova legatura di valore, la nota è messa in uno stato pendente all'interno del vettore *nodePending*. Se, invece, la legatura non è appena iniziata, ma si sta protraendo da più note, non si crea un nuovo oggetto pendente ma si aggiorna, incrementando la durata, l'oggetto già presente nella struttura che ha lo stesso pitch. Solo in corrispondenza del termine della legatura di valore l'oggetto pendente viene inserito nel contenitore definitivo. Vengono salvati nel contenitore definitivo anche le note intermedie, presenti nella legatura di valore, ponendo a zero la loro durata; tale trucco serve a poter preservare eventuali dinamiche espresse in corrispondenza a tali note.

```
else if(getType() != notevisitor::kUndefinedType)
//caso pausa, non viene salvato nel contenitore
        {
            tot_date=tot_date+dur; //avanzamento tempo totale
            resStream<<"rest "<<endl;
        }
```

Il metodo termina incrementando la temporizzazione totale in caso l'evento sia una pausa, anche la pausa, infatti, è identificata dal tag *note* con l'attributo *type* che presenta il valore *rest*.

Parsing pedale e ritardando

Per un problema presente alla libreria, la navigazione non si ferma in corrispondenza del tag *pedal* e neanche in corrispondenza dei livelli immediatamente su-

periori. Risalendo nell'albero il primo tag, in cui la navigazione si ferma, risulta essere il tag *measure*, si dovrà procedere manualmente a scendere l'albero fino a trovare le informazioni riguardanti pedale e ritardando. Il metodo *void visitStart (S_measure& elt)* ferma la navigazione all'inizio di una nuova battuta, dall'elemento elt(di tipo battuta) si scende di livello procedendo per la catena: *measure-direction-direction-type-pedal*; l'estrazione dell'informazione sul pedale avviene sfruttando il seguente codice:

```
if((*div2)!=NULL && (*div2)->getName().size()!=0 &&
(*div2)->getName().compare("pedal")==0)
   {
     for(attr = (*div2)->attributes().begin();
     (*attr)->getName().compare("type")!=0; attr++)
     {}
     //trovo dei cambiamenti dello stato del pedale,
     //vengono inseriti in una mappa
     //date -- stato pedale
     if((*attr)->getValue().compare("start")==0)
     {pedal[date]="start"; }
     else if((*attr)->getValue().compare("stop")==0)
      {
          if(pedal[date]!="")
          { pedal[date]="s&s"; }
          else
           {pedal[date]="stop"; }
      }
  }
```

L'informazione riguardante il pedale è memorizzata in una map con chiave la temporizzazione in cui l'evento accade e come valore lo stato del pedale. Tale informazione non viene memorizzata subito nella struttura definitiva poiché la navigazione si ferma all'inizio della battuta, le note a cui è riferito il pedale, devono ancora essere inserite nella struttura dati. Tramite il metodo *void visitEnd (S_measure& elt)* è possibile, essendo a fine battuta, integrare le informazioni riguardanti il pedale alle note con indicazione temporale uguale a quella esplicitata nella mappa oppure immediatamente successiva.
Operazione simile alla precedente è svolta per gestire l'indicazione di ritardando, in questo caso, però, la mappa tiene traccia del numero di nota in cui occorre un evento di tipo inizio/fine ritardando. Analogamente al caso precedente la memorizzazione nella struttura definitiva avviene nell'esecuzione del metodo

void visitEnd (S_measure& elt). Si riporta il codice relativo all'acquisizione dell'informazione del ritardando:

```
if((*div2)!=NULL && (*div2)->getName().size()!=0 &&
(*div2)->getName().compare("words")==0)
     {
       if((*div2)->getValue().compare("rit.")==0)
       {rit[currNote]="rit"; }
      }
       if((*div2)!=NULL && (*div2)->getName().size()!=0 &&
       (*div2)->getName().compare("words")==0)
       {
         if((*div2)->getValue().compare("a tempo")==0)
         {rit[currNote]="a tempo"; }
       }
```

Gestione del tempo globale

Durante la scansione dello spartito c'è la necessità, come visto precedentemente, di tenere traccia dell'istante temporale in cui accadono i vari eventi. La lettura dello spartito avviene prima di tutto dalla prima battuta del primo rigo, se sono presenti due file di note, arrivati a fine battuta, il cursore torna indietro all'inizio battuta per leggere la seconda riga. Si reitera tale procedura per tutti i righi presenti nello spartito prima di passare alla lettura della seconda battute. Si comprende la necessità di dover incrementare e decrementare la temporizzazione globale concordemente agli spostamenti compiuti lungo lo spartito. I metodi *visitEnd (S_backup& elt)* e *visitEnd (S_forward& elt)* si occupano rispettivamente di decrementare e incrementare la variabile che tiene traccia del tempo totale all'incontrare, nella navigazione dell'albero, dei tag *backup* e *forward*. La funzione *OnOpenParseMusicXML* permette la stampa delle informazioni del parsing su un file di testo (InfoParse.txt), che se non presente viene creato nella cartella del programma CaRo.

2.4.3 Classe Parabola

Prima di passare alla spiegazione dell'esecuzione del brano, che utilizza le informazioni raccolte negli oggetti XMLPar grazie alla funzione *OnOpenParseMusicXML*, è necessario introdurre una nuova classe: *Parabola*, la classe non è stata modificata, si riporta pertanto quanto spiegato nella tesi di Barichello. Tale classe permette di creare nuovi tipi di oggetti, che verranno poi utilizzati per gestire

le informazioni dei legato durante l'esecuzione del file. La classe e i prototipi delle funzioni sono inserite nel file *Parabola.h*, il cui contenuto è riportato qui di seguito:

```
class Parabola{
        private:
        int number;
        //numero della parabola
        double xVertice;
        //coordinate del vertice parabola
        double yVertice;
        double xPuntoSx;
        //coordinate del punto a sinistra del vertice
        double yPuntoSx;
        double xPuntoDx;
        //coordinate del punto a destra del vertice
        double yPuntoDx;
        double a;
        //i 3 coefficienti dell'equazione della parabola
        double b;
        double c;
        public:
        //due costruttori
        Parabola();
        //inizializza tutte le variabili a 0
        Parabola(int num, double xSx, double xDx,double delta);
        //delta=distanza tra le coordinate ySx=yDx YVertice=1
        //metodi
        void setNumber(int num);
        int getNumber();
        void setxSx(double xSx);
        double getxSx();
        void setxDx(double xDx);
        double getxDx();
        void calcolaParabola(double delta);
        //calcola i coefficienti a-b-c
        void calcolaRitardando(double delta);
        //calcola i coefficienti a-b-c
        //per l'implementazione del ritardando
        double calcolaY(double coordX);
        //data una coordinata x della parabola,
```

```
        //mi restituisce la coordinata y
};
```

La variabile privata *number* permette di distinguere le parabole (cioè i diversi legato) tra di loro. Questo perché una nota può essere soggetta a più legato contemporaneamente. Il valore di questa variabile viene estratto dal file MusicXML in quanto anch'esso utilizza un numero intero per distinguere gli slur tra di loro. Gli oggetti di questa classe conterranno le coordinate dei tre punti che andranno a formare la parabola (vertice, punto a sinistra del vertice, e punto a destra del vertice), e le tre costanti *a-b-c* che vanno a formare l'equazione esplicita della parabola.

$$y = ax^2 + bx + c$$

La classe ha due costruttori: uno che inizializza tutte le variabili a 0, e uno che dando in ingresso le coordinate x dei due punti e il delta, calcola ed imposta il valore delle tre costanti che vanno a formare l'equazione della parabola. I metodi pubblici permettono di leggere e settare le variabili private, di calcolare le tre costanti dell'equazione, e di restituire la coordinata y della parabola corrispondente alla coordinata x data in input alla funzione. La classe è implementata all'interno del file *Parabola.cpp*. L'implementazione è molto semplice, e l'unico punto che è bene sottolineare è come vengono calcolati i valori di *a-b-c*. Per trovare queste costanti basta utilizzare la formula della parabola passante per il vertice ed un punto:

$$y - yVertice = a(x - xVertice)^2$$

Scegliendo come punto quello a sinistra del vertice, e sostituendo le sue coordinate al posto di *x* e *y*, siamo in grado di calcolare il valore di *a*. Poi riscrivendo la formula qui sopra con il valore di *a*, e portandola in forma esplicita, è possibile ricavare anche i valori di *b* e *c*.

2.4.4 Esecuzione di un brano

Una volta caricato ed eseguito il parsing del file MusicXML è possibile eseguire lo spartito in tre diverse modalità: meccanica, neutra ed espressiva. In base alla scelta della modalità vengono settate delle variabili booleane. Caso esecuzione meccanica:

```
isNeutra=true;
isMecchanical=true;
```

caso esecuzione neutra:

```
isNeutra=true;
isMecchanical=false;
```

mentre nel caso di esecuzione espressiva le due variabili sono impostate entrambe a *false*. Dopo aver impostato tali variabili è chiamato in tutti e tre i casi il metodo *OnMenuItemPlayWithExpressivenessSelected*, il quale si comporta in maniera diversa concordemente al valore delle variabili prima impostate. Viene fatta questa scelta poiché gran parte del codice è comune alle varie esecuzioni, si evita di replicare grandi pezzi di programma.
La funzione *OnMenuItemPlayWithExpressivenessSelected* inizia estraendo tutte le informazioni presenti nel form Opzioni[5] (ad esclusione della seconda scheda Interface Parameters), ed inserite nelle seguenti strutture:

nelle variabili *Porta*, *Tempo*, *Latenza*, e *Risoluzione*, che sono private della classe *CaRo2Main*, vengono inseriti i corrispondenti valori della prima scheda;

in *SoloPiano* viene inserito true se nella quarta scheda del form Opzioni è stata scelta l'opzione play only on channel 1, altrimenti viene inserito false;

parametriCstaccato è un array a livello globale di grandezza 4, che contiene i valori dei parametri *Ktempo*, *Mvelocity*, *Kvelocity* e *Klegato* da applicare in presenza di un staccato nel file MusicXML. Tali valori sono dei parametri moltiplicativi, e si trovano nella terza scheda del form Opzioni;

parametriCaccento è un array a livello globale che contiene i valori dei quattro parametri nel caso di un accento;

parametriCrespiro è un array a livello globale che contiene i valori dei quattro parametri nel caso di un respiro;

parametriCtenuto è un array a livello globale che contiene i valori dei quattro parametri nel caso di un tenuto;

parametriDeltaLegato è un array a livello globale che contiene i valori dei quattro parametri in presenza di un legato. A differenza dei precedenti parametri, nel caso di un legato il parametro si riferisce ad una differenza di valori (delta), utilizzato per calcolare la costante moltiplicativa da applicare;

[5] verranno spiegati in seguito tutti i parametri che possono essere settati grazie a questo input

parametriDynamics è un array a livello globale di grandezza 6 che contiene i valori di Kvelocity per ognuna delle sei dinamiche (pp, p, mp, mf, f, ff). In base alla dinamica presente nel file MusicXML, viene utilizzato il parametro moltiplicativo corrispondente;

parametriCmelodia è una variabile double che contiene un parametro moltiplicativo di valore maggiore di 1, che viene utilizzato per aumentare la Kvelocity delle note presenti nel channel 0. Questa parametro è stato creato per permettere di aumentare l'intensità delle note del canale 0, che di solito sono le note della melodia del brano, rispetto alle note dell'accompagnamento che di solito si trovano negli altri canali.

Le informazioni inserite nella seconda scheda del form Opzioni invece, vengono estratte ed inserite nelle variabili corrispondenti solo nel caso in cui l'utente abbia premuto il tasto *Play with Expressiveness* dal menù del CaRo (le conseguenze della pressione di questo tasto vengono spiegate meglio più avanti). Le variabili sono le seguenti:

modelloKtempo è un array a livello globale di grandezza 5, che contiene i valori del parametro *Ktempo* per le 5 intenzioni espressive bright, hard, light, soft, e heavy. Tali valori sono presenti nella seconda scheda del form Opzioni;

modelloMvelocity è un array a livello globale che contiene i valori del parametro *Mvelocity* per le cinque intenzioni espressive;

modelloKvelocity è un array a livello globale che contiene i valori del parametro *Kvelocity* per le cinque intenzioni espressive;

modelloKlegato è un array a livello globale che contiene i valori del parametro *Klegato* per le cinque intenzioni espressive;

Se è stato premuto *Play Neutral* o *Play Mecchanically* queste ultime variabili vengono tutte settate a 1. Successivamente al set dei parametri è scandito per intero il contenitore contenente le varie note, quindi gli oggetti XMLPar, si cerca di identificare se esistono note con lo stesso pitch vicine, più precisamente all'interno di un raggio posto a 2000. Se è identificata una nota all'interno del raggio con lo stesso pitch è memorizzata la differenza tra i date delle note, nella posizione corrispondente alla nota, in un vettore chiamato *DurataMax* della stessa dimensione del contenitore delle note. Il vettore in questione è un array posizionale parallelo al contenitore delle note che serve per tenere traccia di eventuali vicinanze tra note

con lo stesso pitch, nel caso non venga trovata una nota è memorizzato nel vettore il valore -1. La situazione di avvicinamento troppo stretto si vuole evitare per problemi di esecuzione del player midi, gli eventi di attivazione e disattivazione della nota corrispondente possono entrare in collisione non rispettando il corretto ordine.
Prima di terminare la funzione è creato ed attivato un oggetto di tipo wxTimer con ciclicità pari al valore *Risoluzione* letto dal form Opzioni. Il timer in questione, passato un tempo pari al valore Risoluzione, va ad eseguire il codice contenuto all'interno del metodo *OnPerformanceTimer*.

OnPerformanceTimer

OnPerformanceTimer è una funzione chiamata ciclicamente dal timer attivato nel precedente metodo. Tale procedura serve per scandire progressivamente il vettore contenente i vari eventi musicali da eseguire di tipo XMPar, è il motore dell'applicazione. Si considera, da questo momento, un'esecuzione di tipo espressivo che è la più completa, per quella neutra si è visto che sufficiente impostare a 1 di alcuni parametri, che essendo oggetto di moltiplicazione risultano ininfluenti; per l'esecuzione meccanica non è applicata nessun tipo di modifica alle note.
L'argomentazione che segue è ripresa dalla tesi di Barichello Massimiliano[6] con opportune modifiche delle parti riprogettate.
L'interfaccia del CaRo(commentata nel dettaglio più avanti) fornisce all'utente cinque intenzioni espressive, nel codice sono detti aggettivi, le quali hanno delle coordinate ben precise nello spazio 2D, per essere utilizzate, vengono inserite in due vettori:

```
//coordinate dei 5 aggettivi
//(Bright-Brillante Hard-Duro Light-Leggero
Soft-Morbido Heavy-Pesante)
const float braniX[NumeroAggettivi]=
{0.9075,0.325,0.775,0.3375,0.13};
const float braniY[NumeroAggettivi]=
{0.53,0.857,0.234,0.065,0.53};
```

Ogni aggettivo ha delle costanti moltiplicative associate, che permettono di modificare i campi degli eventi di tipo nota, rendendola così più adatta all'intenzione espressiva. Muovendo il mouse sopra ad un aggettivo, l'utente può applicare al brano quella partico- lare intenzione espressiva, e può passare da un aggettivo

[6]http://tesi.cab.unipd.it/35067/1/tesiBarichello.pdf

ad un altro in maniera graduale (smooth morphing) producendo nel brano dei cambiamenti morbidi tra diverse intenzioni espressive. I parametri più importanti e molto più incisivi di altri per la riproduzione delle intenzioni espressive, come argomentato nel primo capitolo, sono:

Ktempo , che va ad agire sulla durata dell'evento nota, e anche sull'istante di inizio della nota successiva;

Mvelocity una costante moltiplicativa che viene applicata allo scarto tra la velocity dell'evento nota e le velocity media

Kvelocity che va ad agire sul campo velocity dell'evento nota, cioè sull'intensità di esecuzione sonora della nota;

Klegato che va ad agire sulla durata dell'evento nota.

Come detto sopra ogni intenzione espressiva ha un valore per questi quattro parametri. Possono capitare due situazioni:

1. se nell'interfaccia il mouse giace perfettamente sopra un aggettivo, vengono usati direttamente i quattro parametri corrispondenti all'aggettivo per applicarli ai campi dell'evento di tipo nota corrente;

2. se invece il mouse si trova in un punto casuale dello spazio 2D, diverso dalle cinque coordinate degli aggettivi, vengono pesate le distanze tra il puntatore del mouse ed ogni intenzione espressiva. Poi viene calcolata la media ponderata di ogni parametro sui cinque aggettivi, pesando in base al reciproco delle distanze. Qui di seguito è mostrato il codice relativo a questi calcoli.

```
for (int i=0; i<NumeroAggettivi; i++){
        Distanze[i]=(asseX-braniX[i]*Wlarghezza)^2+
                                (asseY-braniY[i]*Waltezza)^2;
        Pesi[i]=1.0/Distanze[i];
//meno distante è, e più peso c'è
        SommaPesi=SommaPesi+Pesi[i];
     }
     for (int i=0; i<NumeroAggettivi; i++){
        Ktempo=Ktempo+modelloKtempo[i]
                                          *Pesi[i]/SommaPesi;
        Mvelocity=Mvelocity+modelloMvelocity[i]
```

```
                                                *Pesi[i]/SommaPesi;
        Kvelocity=Kvelocity+modelloKvelocity[i]
                                                *Pesi[i]/SommaPesi;
        Klegato=Klegato+modelloKlegato[i]
                                                *Pesi[i]/SommaPesi;
}
```

Dove gli array che hanno la parola *modello* come prefisso, contengono i valori dei parametri delle cinque intenzioni espressive, che vengono in precedenza estratti dalla seconda scheda del form Opzioni.
La funzione *OnPerformanceTimer* viene chiamata ogni Risoluzione ms, e siccome le note del brano hanno delle durate che in generale non sono sincronizzabili con intervalli di Risoluzione ms, all'interno dell'OnPerformanceTimer sono necessari dei controlli per evitare che una nota del brano venga manipolata più di una volta o in tempi non adeguati. Per farlo vengono utilizzate le seguenti variabili:

InizioCiclo Viene inizializzata all'inizio della funzione tramite *MidiGetTime()*, e quindi contiene l'istante temporale di inizio dell'OnPerformanceTimer.

StopTime All'inizio della funzione *StopTime* contiene l'istante temporale di inizio del prossimo evento di tipo nota da manipolare. Quando *l'OnPerformanceTimer* finisce di analizzare una nota, *StopTime* viene impostato col valore di inizio dell'evento nota successivo.

L'intero corpo dell'OnPerformanceTimer viene eseguito solo se la funzione viene chiamata dopo l'istante temporale di inizio del prossimo evento nota da manipolare. Per far questo basta inserire questo controllo:

```
if (InizioCiclo>StopTime) { ...
```

Passato questo controllo, la funzione controlla la posizione del mouse e calcola i valori dei quattro parametri rispetto alle cinque intenzioni espressive dell'interfaccia di CaRo. Poi per evitare che una nota venga manipolata più di una volta, viene fatto un ulteriore controllo sul tempo

```
while (MidiGetTime()>StopTime) { ...
```

Prima di entrare nel *while*, *StopTime* indica il tempo di inizio (già oltrepassato) dell'evento nota che si sta manipolando. All'interno del *while* invece *StopTime* viene modificato con il tempo di inizio dell'evento nota successivo. In questo modo si entrerà nel while solo una volta, cioè solo quando la nota corrente verrà

manipolata per la prima volta; successivamente *StopTime* avrà un valore sicuramente superiore al *MidiGetTime()*. All'interno del while viene fatto un ulteriore controllo, necessario per evitare che il tempo di esecuzione di *OnPerformanceTimer* sia superiore al tempo necessario al Timer per scadere. Tale situazione viene indicata all'utente con un messaggio di saturazione nella *StatusBar*, che invita ad aumentare la *risoluzione* presente nel form Opzioni. Il codice che effettua questo controllo è il seguente:

```
if (MidiGetTime()-InizioCiclo>0.9*Risoluzione) {
    Saturazione=true;
    StatusBar1->SetStatusText
              (_T("Saturation! Increase resolution"));
    }
else Saturazione=false;
```

Tale soluzione risulta ottima in quanto aumentando la saturazione aumenta anche la durata del timer, e quindi la funzione ha più tempo per completare la sua esecuzione.
Il ciclo scandisce ogni nota del vettore contenente gli oggetti XMLPar e per ognuno di essi calcola i parametri, prima menzionati, per apportare le modifiche alle note e quindi all'esecuzione del brano. A seconda delle dinamiche espresse negli oggetti di tipo XMLPar sono svolte le seguenti operazioni:

Se l'oggetto indica che la nota corrispondente ha uno staccato nello spartito modificato con Finale, allora viene modificata la costante moltiplicativa che viene influenzata dallo staccato, con un peso pari alla costante indicata nella terza scheda del form Opzioni. In questo caso viene influenzata la variabile *klegato*:

```
Klegato = Klegato*parametriCstaccato[3];
```

Klegato andrà ad incidere sulla durata della nota;

Nel caso di un accento viene modificata la *Kvelocity*, che va ad influenzare la velocità di pressione del tasto (campo *velocity* dell'evento di tipo *typeNote*)

```
Kvelocity = Kvelocity*parametriCaccento[2];
```

In presenza di un respiro vengono modificate le seguenti costanti moltiplicative

```
Ktempo = Ktempo*parametriCrespiro[0];
Klegato =Klegato*parametriCrespiro[3];
```

che vanno ad agire entrambe sulla durata dell'evento nota, e il *Ktempo* anche sull'istante di inizio della nota successiva (tramite *OnsetEspressivo*), il quale viene calcolato in base alla differenza tra i date originali delle due note moltiplicata per il valore di *Ktempo*:

```
OnsetEspressivo=OnsetEsprPrec+Ktempo*
                    (OnsetNeutro-OnsetNeutroPrec);
```

Nel caso di tenuto viene modificato il valore del *Klegato*, e quindi della durata della nota.

```
Klegato = Klegato * parametriCtenuto[3];
```

In presenza di una delle cinque dinamiche, viene modificata la *Kvelocity*. Infatti una dinamica del tipo ff necessità una pressione del tasto (*velocity*) molto più potente rispetto ad una dinamica del tipo pp.

Nel caso di un pedale si deve controllare il tipo di pedale. In caso di pedaleON si deve creare un nuovo evento di tipo *typeCtrlChange* il quale, una volta mandato al Player, permette di applicare gli effetti del pedale del pianoforte alle note successive. Tale evento è composto da due campi: il campo 0 indica il tipo di controllo, nel nostro caso viene impostato a 64 per indicare il pedale (Damper Pedal), il campo 1 invece indica il valore di pressione del pedale (da 0 a 63 pedale off, da 64 a 127 pedale on). Inoltre l'evento viene aggiunto anche alla sequenza MidiShare *SeqEspressiva*, in modo che risulti presente sia nel Player e sia nella sequenza di tutti gli eventi del brano. Questo garantisce la presenza di tale evento anche nel caso di salvataggio del brano MIDI durante l'esecuzione di *Play with Expressiveness*. In caso di pedaleOFF invece, vengono eseguite le stesse cose, con la differenza che vengono impostati in maniera diverse i campi dell'evento *typeCtrlChange* che viene creato.

In presenza di legato, le operazioni da fare sono molto più complesse. Infatti, dobbiamo considerare nel loro insieme tutte le note che stanno sotto ad uno slur, e costruire una parabola. La parabola è del tipo

$$y = ax^2 + bx + c \, con \, a < 0$$

cioè una parabola rivolta verso il basso, e passante per due punti. Le coordinate x dei due punti vengono trovate andando a guardare il campo date delle

due note in cui inizia e finisce il legato. La coordinata y invece, che è uguale per entrambi i punti, viene calcolata in base al parametro DeltaLegato presente nel form Opzioni:

$$coordinata\ y = 1 - DeltaLegato$$

Il vertice della parabola ha per coordinata y il valore 1, mentre per coordinata x il valore che sta esattamente alla metà delle coordinate x dei due punti. Grazie al vertice e ai due punti è possibile calcolare l'equazione della parabola, e utilizzarla per trovare le costanti moltiplicative da applicare alle note del brano che sono coinvolte nello slur. Ci viene quindi in aiuto la classe *Parabola*, spiegata in precedenza, che attraverso i suoi metodi permette l'inserimento delle coordinate dei due punti e del delta, per ottenere le costanti che andranno a formare l'equazione della parabola. Infatti la costante moltiplicativa da applicare ad una nota con campo date pari a X1, non è altro che la coordinata y trovata sostituendo il valore X1 nell'equazione della parabola. Nella classe *Parabola* il metodo che implementa questa funzione è calcolaY. Nel nostro caso il legato va ad influire sulle costanti *Ktempo* e *Kvelocity*, quindi per ogni slur è necessario creare due parabole, ognuna per le due costanti moltiplicative. Attraverso i valori dei due parametri *DeltaLegato* del form Opzioni, possiamo trovare l'equazione della parabola relativa al *Ktempo* e l'equazione della parabola relativa alla *Kvelocity*. Dal punto di vista implementativo, le due parabole vengono create solo se l'oggetto XMLPar indica che la nota corrispondente nel Player ha un legato-start. Dalla nota corrente viene estratto il date, che sarà la coordinata x del punto a sinistra del vertice della parabola, e dall'oggetto XMLPar viene estratto il number del legato. Per poter costruire la parabola sono ora necessarie le informazioni della nota che si trova a fine dello slur. Per trovarla viene fatto un ciclo secondario che avanza nel vettore contenitore, e viene trovato l'oggetto contenen-te lo stesso number del legato e il parametro legato-stop. Allo stesso modo viene preso l'evento nota corrente, e vengono analizzati gli eventi successivi finché si trova l'evento *typeNote* che ha lo stesso ID dell'oggetto XML con il legato-stop appena trovato. In questo modo abbiamo la possibilità di estrarre il date dell'ultima nota sotto al legato (cioè la coordinata x del punto a destra del vertice della parabola), e di inserirlo nei due oggetti di tipo *Parabola*. Avendo i date delle due note agli estremi del legato, e il delta presente nel form Opzioni, è possibile calcolare per ogni parabola le tre costanti *a-b-c* che vanno a formare l'equazione, utilizzando la funzione *calcolaParabola* della classe. In questo modo i due oggetti di tipo *Parabola* sono ora utilizzabili per trovare le costanti moltiplicative da applicare agli eventi di tipo nota che sono influenzati dal legato.

Con il programma Finale è possibile aggiungere più di un legato, e CaRo gestisce correttamente anche il caso di più legato sovrapposti in una stessa nota. Infatti da una nota possono iniziare e terminare più di uno slur, e allo stesso tempo tale nota può essere influenzata da altri legato iniziati in note precedenti. In CaRo questa situazione viene gestita creando due vettori: *listaParaboleTempo* e *listaParaboleVelocity*.
Questi vettori dinamici conterranno in ogni istante i legato, e quindi le parabole che influenzano l'evento nota corrente. Quindi appena abbiamo un oggetto XMLPar che indica un legato-start vengono creati i due oggetti di tipo *Parabola*, vengono riempiti con tutte le informazioni necessarie per la creazione dell'equazione delle parabola (come visto in precedenza), e vengono aggiunti ai due vettori. Quando poi si arriverà ad un oggetto XMLPar contenente un legato-stop, vengono eliminati i due oggetti *Parabola* corrispondenti nei due vettori grazie alla corrispondenza tra il number dello slur nel file MusicXML e il number delle due parabole nei vettori.
Per ogni legato che influuenza la nota corrente vengono aggiornate le due costanti moltiplicative *Ktempo* e *Kvelocity* tramite la funzione *calcolaY*.

```
for(int i=0;i<listaParaboleTempo.size();i++){
Ktempo = Ktempo *
listaParaboleTempo[contaParabole]
.calcolaY(contenitore[currentNote].getBeat());
Kvelocity = Kvelocity *
listaParaboleVelocity[contaParabole]
.calcolaY(contenitore[currentNote].getBeat());
```

Si è vista come avviene la modifica dei vari parametri, ora verrà spiegato come i parametri influiscano realmente sui singoli eventi musicali.

Applicazione delle modifiche alle note

Come già accennato nella sezione precedente, verrà illustrato ora come vengono effettiva-mente applicate le costanti moltiplicative all'evento nota corrente. Come già spiegato tali costanti vanno ad agire su certi campi dell'evento *typeNote.* Le modifiche sugli eventi sono eseguite direttamente negli oggetti presenti nel vettore *contenitore*, prima della modifica è creata una copia di tale vettore per rendere possibile una nuova elaborazione delle informazioni indipendente dalla corrente senza il bisogno di ricaricare il file MusicXML.

La durata dell'evento nota corrente viene moltiplicata per *Klegato* e *Ktempo*, e poi viene preso il minimo tra il risultato di questa moltiplicazione e il valore massimo della durata per la nota corrente che è stato calcolato ed inserito nel vettore *DurataMax*.

La velocity dell'evento nota viene aggiornata nel seguente modo: viene preso il minimo tra il valore acquisito dallo spartito e il valore massimo della velocità *(KVMaxOut)*, e il valore massimo tra il risultato di quest'ultima operazione e il valore minimo della velocità *(KVMinOut)*.

Il valore date dell'evento nota (indica l'istante temporale della nota) viene impostato in base all'istante di pressione del tasto *Play with Expressiveness (IstanteInizio)*, alla *latenza* (di default vale 100ms, ed è estratta dal form Opzioni), e in base al valore di *OnsetEspressivo* che viene calcolato in precedenza.

```
contenitore[currentNote].setBeat(IstanteInizio+
                                 Latenza+OnsetEspressivo);
```

Se il *date* risultante della nota rimane inferiore al tempo corrente *(MidiGetTime())*, la nota non verrà mai processata dal Player. Per risolvere questo problema basta agire sulla *latenza*, chiedendo all'utente di aumentarla tramite il form Opzioni.

```
if (contenitore[currentNote].getBeat()<MidiGetTime())
{
    Saturazione=true;
    StatusBar1->SetStatusText
                (_T("Saturation! Increase resolution"));
}
```

In caso di saturazione l'evento nota non viene mandato ne al Player, ne inserito nella sequenza MidiShare *SeqEspressiva*, ma il valore della variabile Saturazione viene impostata di nuovo a false e si passa all'evento successivo. Se invece non c'è nessuna saturazione:

- viene creato un nuovo evento *tmpKeyOnOff* di tipo *typeKeyOn* nel quale vengono inseriti i valori espressivi appena trovati (date, velocity), e i valori del pitch, della porta e del channel dell'evento nota originale. Poi tale evento viene inviato al Player

```
MidiSend(ourRefNum,MidiCopyEv(tmpKeyOnOff));
```

- viene creato un nuovo evento che identifica la fine della nota appena creata (cioè un evento *typeKeyOff*). Per farlo viene modificato l'evento *typeKeyOn precedente*, visto che è già stato utilizzato, settando a 0 la *velocity*, e il *date* viene aumentato della durata che dovrebbe avere la nota espressiva

  ```
  tmpKeyOnOff->date=tmpKeyOnOff->date
                    +contenitore[currentNote].getDur();
  ```

Nel caso di esecuzione di tipo Meccanica i parametri acquisiti dal parsing non vengono modificati e vengono eseguiti solo questi due ultimi punti, invio puro degli eventi midi di attivazione e disattivazione delle note.

Al termine di un singolo giro è valutata la posizione di scansione del vettore contenitore, si possono presentare due casi:

- se non esiste un evento nota successivo a quello corrente, la *StatusBar* viene settata a Stop, viene bloccata la funzione di ciclicità del timer, è riportato il vettore *contenitore* allo stato originario per consentire una diversa performance; finita l'esecuzione è possibile salvare la performance in un file Midi.

- se invece esiste un evento di tipo nota successivo, allora vengono aggiornate le variabili *OnsetNeutro* - *OnsetEspressivo* - *StopTime*:

  ```
  currentNote++;
  OnsetNeutro=
              contenitore[currentNote].getBeat();
  OnsetEspressivo=OnsetEsprPrec+Ktempo
                  *(OnsetNeutro-OnsetNeutroPrec);
  StopTime=IstanteInizio+OnsetEspressivo;
  ```

  dove *contenitore[currentNote].getBeat()* è il date del nuovo evento nota prima delle modifiche espressive, e *IstanteInizio* indica l'istante temporale di pressione del tasto *Play with Expressiveness* dal menù Play. Dal momento che *OnsetEspressivo* è rispetto all'istante temporale di esecuzione della funzione *PlayExpressiva*, *StopTime* viene così impostato con il valore di inizio dell'evento nota successivo a quello appena analizzato da *OnPerformanceTimer*.

Come per la funzione *ParserMusicXML*, durante tutte le azioni svolte dalla funzione *OnPerformanceTimer* vengono messe in delle variabili delle informazioni che aiutano lo sviluppatore a capire se le operazioni svolte dalla funzione sono corrette o meno. Queste variabili sono delle stringhe globali il cui contenuto viene poi stampato in file di testo contenuti nella cartella del programma (InfoParse.txt per *ParserMusicXML* , e FileInfo.txt per *OnPerformanceTimer*).

Capitolo 3

CaRo 2.0: User Interface e funzionalità

Nel capitolo precedente è stata presentata l'applicazione dal punto di vista implementativo, sono state presentate alcune delle funzioni, quelli più complicate in termini tecnici. In questo capitolo è illustrata la user interface e spiegate tutte le funzioni offerte dal software CaRo 2.0.
All'apertura dell'applicazione il software si presenta come in *Figura 4.2*. Andiamo nell'ordine ad elencare le diverse operazioni che si possono eseguire.

3.1 Pannello Opzioni

Dal menù principale è possibile accedere al *Panello Opzioni* seguendo il percorso:

```
view->Options...
```

Il form Opzioni è composto da quattro schede:

- *MIDI Parameters:* la prima scheda (vedi *Figura 3.2*) contiene quattro caselle di testo che individuano la porta della connessione MidiShare, il valore del Tempo (o Beat Per Minute o Quarter note Per Minute), la latenza (parametro in ms utilizzato per spostare gli istanti temporali degli eventi), e la risoluzione (valore in ms di un timer che viene utilizzato durante l'esecuzione di un brano). All'apertura del programma Caro, i valori delle caselle di testo vengono settati rispettivamente a 0, 0, 100, 10. Il settaggio di questi valori viene implementato nel file *Options.cpp*.

Figura 3.1: *Interfaccia grafica CaRo*

- *Interface Parameters* nella seconda scheda (vedi *Figura 3.3*) troviamo tutti i valori che permettono a CaRo di rendere interattiva l'interfaccia per l'esecuzione espressiva del brano in real time.

- MusicXML Parameters: Nella terza scheda (vedi *Figura 3.4*) troviamo tutte le costanti moltiplicative relative ad ogni tipo di informazione che viene acquisita tramite l'uso del file MusicXML; infatti ognuna di queste informazioni va ad agire in alcuni dei parametri del modello utilizzato da CaRo, che permettono la modifica espressiva del brano. Nella seconda e nella terza scheda del form Opzioni, ci sono due pulsanti *Load* e *Save*, che permettono

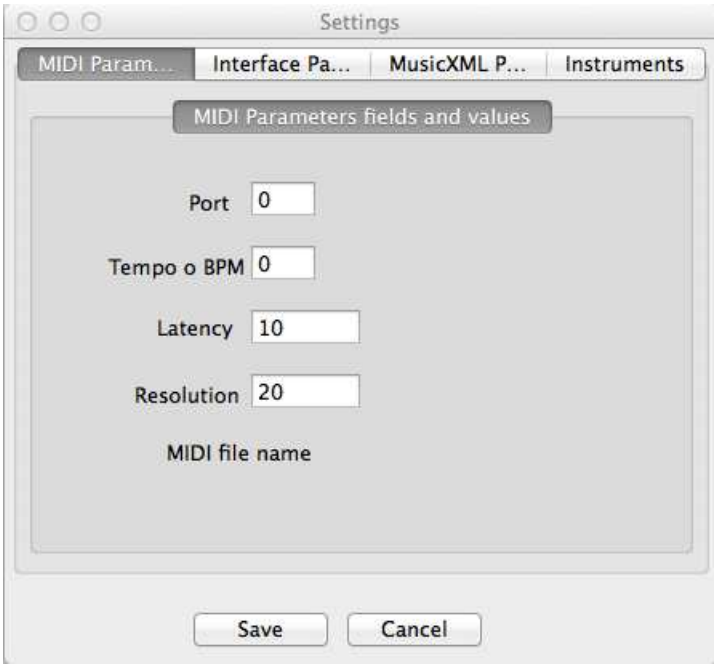

Figura 3.2: *Prima scheda opzioni*

di caricare e salvare i parametri delle scheda su file di testo. Queste due schede sono risultate molto utili nella fase di testing del programma CaRo, in quanto permettono la modifica delle costanti moltiplicative durante l'esecuzione del programma, e l'immediato ascolto dei cambiamenti effettuati. I valori di default sono caricati da un file presente dalla cartella del progetto.

- *Instruments:* nella quarta scheda (vedi *Figura 3.5*) è possibile selezionare lo strumento di esecuzione del brano (al momento CaRo non applica nessuna modifica a riguardo), e scegliere tramite una check button indicata con play only on channel 1 se il Player deve eseguire tutte le note del file con il pianoforte. Il Player ha di default quest'impostazione attivata, quindi verrà aggiunto un evento, che modifica il timbro, alla sequenza MidiShare solo se tale opzione non è selezionata. All'avvio del programma CaRo l'opzione play only on channel 1 non è selezionata.

Quando l'utente preme sui pulsanti *Load from file* o *Save to file* della seconda e terza scheda, vengono richiamate delle funzioni del form Opzioni. Tali funzioni sono implementate nel file *Options.cpp*. I metodi provvedono a caricare e salvare

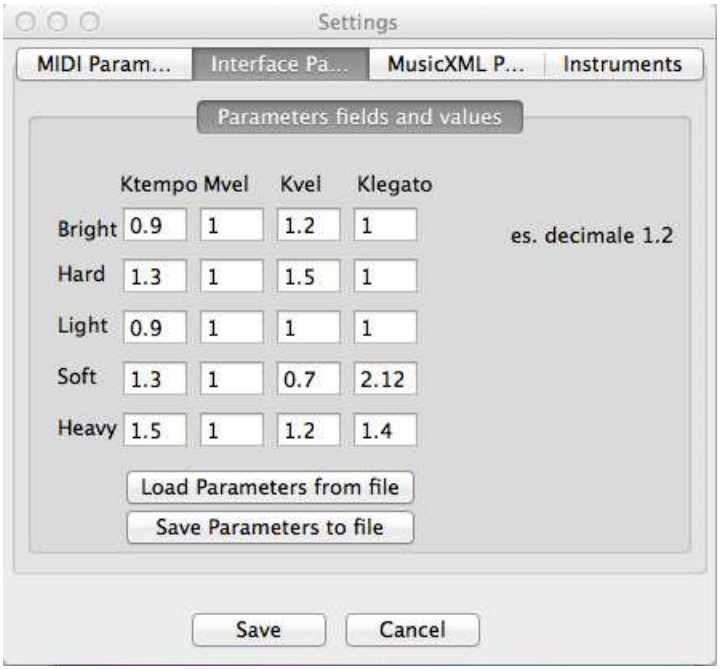

Figura 3.3: *Seconda scheda opzioni*

i file relativi ai parametri da e in un file a scelta dell'utente, selezionato dall'apposito form. Il pulsante *Save*, presente in tutti i pannelli, consente di salvare i parametri di tutte le schede nei relativi file di configurazione presenti nella cartella del progetto. I Parametri presenti nella seconda scheda sono salvati in un file chiamato *AdjectivesParameters.txt* presente nella cartella del progetto, se si desiderasse caricare un file che presenta valori diversi per questa scheda dovrà avere la seguente struttura:

```
0.9 Bright/Brillante - Ktempo
1 Bright/Brillante - Mvelocity
1.2 Bright/Brillante - Kvelocity
1 Bright/Brillante - Klegato
1.3 Hard/Duro        - Ktempo
1 Hard/Duro        - Mvelocity
1.5 Hard/Duro        - Kvelocity
1 Hard/Duro        - Klegato
0.9 Light/Leggero    - Ktempo
1 Light/Leggero    - Mvelocity
1 Light/Leggero    - Kvelocity
```

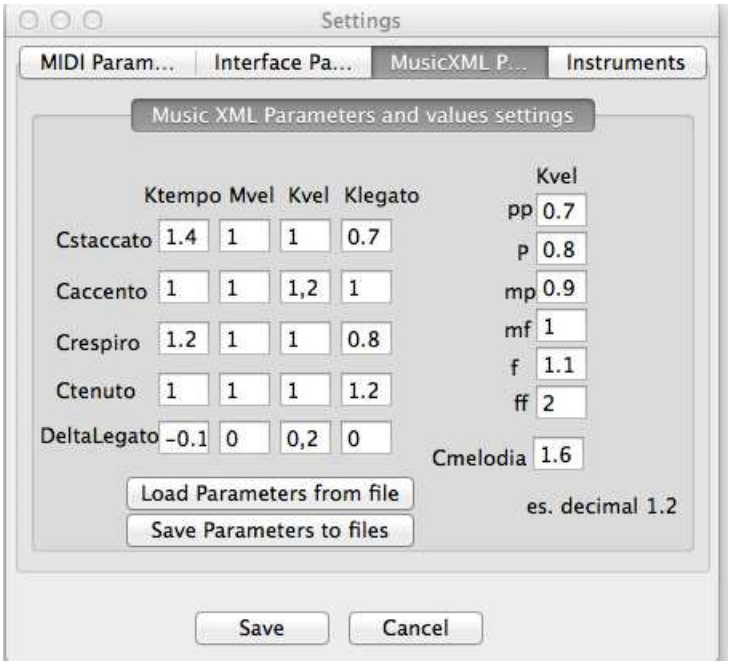

Figura 3.4: *Terza scheda opzioni*

```
1 Light/Leggero    - Klegato
1.3 Soft/Morbido     - Ktempo
1 Soft/Morbido     - Mvelocity
0.7 Soft/Morbido     - Kvelocity
2.12 Soft/Morbido     - Klegato
1.5 Heavy/Pesante    - Ktempo
1 Heavy/Pesante    - Mvelocity
1.2 Heavy/Pesante    - Kvelocity
1.4 Heavy/Pesante    - Klegato
```

Analogamente, per la terza scheda i parametri sono salvati in un file chiamato *MusicXML-parameters.txt* che riporta la seguente struttura:

```
1.4 CstaccatoKtempo
1 CstaccatoMvelocity
1 CstaccatoKvelocity
0.7 CstaccatoKlegato
1 CaccentoKtempo
1 CaccentoMvelocity
1,2 CaccentoKvelocity
```

Figura 3.5: *Quarta scheda opzioni*

```
1 CaccentoKlegato
1.2 CrespiroKtempo
1 CrespiroMvelocity
1 CrespiroKvelocity
0.8 CrespiroKlegato
1 CtenutoKtempo
1 CtenutoMvelocity
1 CtenutoKvelocity
1.2 CtenutoKlegato
-0.1 DeltaLegatoKtempo
0 DeltaLegatoMvelocity
0,2 DeltaLegatoKvelocity
0 DeltaLegatoKlegato
0.7 DynamicsKvelpp pp - pianissimo
0.8 DynamicsKvelp  p  - piano
0.9 DynamicsKvelmp mp - mezzo-piano
1 DynamicsKvelmf mf - mezzo-forte
2 DynamicsKvelf  f  - forte
1.1 DynamicsKvelff ff - fortissimo
```

```
1.6 Cmelodia
```

Per quanto riguarda la prima e quarta scheda i parametri sono salvati in un unico file chiamato *OtherParameters.txt* che presenta la seguente struttura:

```
0 Port
0 Tempo
10 Latency
20 Resolution
0 ChoiceInstruments
0 IsSoloPiano
```

Una volta regolati i vari parametri si può procedere con l'esplorazione delle altre funzioni dell'applicazione.

3.2 Caricamento di un file MusicXML

Per caricare all'interno del programma un file di tipo MusicXML ci sono due diverse possibilità:

- Dal menù seguendo il percorso

  ```
  File->Open MusicXML
  ```

- Dalla pratica toolbar dell'applicazione premendo il tasto riportato in *Figura 3.6*

Figura 3.6: *Open MusicXML*

Procedendo con una delle due procedure equivalenti si carica un file MusicXML e contestualmente il programma ne esegue il parsing. Un messaggio a schermo notifica il termine dell'operazione; un file, salvato nella cartella del programma (ParseInfo.txt), descrive i vari elementi trovati e caricati nel programma.

3.3 Esecuzione del brano

Una volta caricato il brano è pronto per essere eseguito, l'applicazione, infatti, ha già caricato e fatto il parsing della struttura del file. Esistono tre diversi tipi di possibile esecuzioni del brano:

- **Meccanica** Gli eventi musicali sono eseguiti senza segni di espressività, neanche quelli presenti nello spartito. L'esecuzione è possibile tramite il menù seguendo i comandi:

  ```
  Play->Play Mecchanical
  ```

  oppure tramite il tasto della toolbar che si può vedere *Figura 3.7*

  

  Figura 3.7: *Play Mecchanical*

- **Neutra** Gli eventi musicali sono eseguiti con solo i segni di espressività presenti nello spartito. L'esecuzione è possibile tramite il menù seguendo i comandi:

  ```
  Play->Play Neutral
  ```

  oppure tramite il tasto della toolbar che si può vedere *Figura 3.8*

  

  Figura 3.8: *Play Neutral*

- **Espressiva** Gli eventi musicali sono eseguiti tenendo conto delle indicazioni espressive contenute nello spartito e dell'input dell'utente fornito tramite lo spazio espressivo. L'esecuzione è possibile tramite il menù seguendo i comandi:

  ```
  Play->Play with expressiveness
  ```

  oppure tramite il tasto della toolbar che si può vedere *Figura 3.9*

Figura 3.9: *Play with expressiveness*

3.4 Play, Pausa, Stop riproduzione

Una volta avviata la performance di un brano è possibile mettere l'esecuzione in pausa grazie al tasto di *Figura 3.10* oppure dal menù, al percorso:

Figura 3.10: *Pausa*

`Play->Pause`

La riproduzione può essere ripresa in qualsiasi momento con il tasto *Play* mostrato in *Figura 3.11* oppure tramite il menù, seguendo il percorso:

Figura 3.11: *Play*

`Play->Play`

Se si desidera interrompere definitivamente l'esecuzione si usa il tasto *Stop* di *Figura 3.12* oppure tramite il menù seguendo il percorso:

`Play->Stop`

3.5 Salvataggio di una performance

Terminata un'esecuzione o dopo aver premuto il testo *Stop* è possibile procedere al salvataggio della performance appena eseguita in un file di tipo Midi. Tramite la pressione del tasto di *Figura 3.13* oppure tramite il menù, seguendo il percorso:

`File->Save`

Viene aperta una finestra di dialogo che fa scegliere all'utente il percorso in cui salvare il file.

Figura 3.12: *Stop*

Figura 3.13: *Save*

3.6 Caricamento ed esecuzione di un file Midi

L'applicazione CaRo consente l'esecuzione di un brano in formato Midi. Per procedere al caricamento di tale file si preme il tasto mostrato in *Figura 3.14* oppure tramite il menù,

Figura 3.14: *Open file Midi*

seguendo il percorso:

```
File->Open MIDI...
```

si procede all'esecuzione dello stesso utilizzando il menù al percorso:

```
Play->Play Midi Mecchanical
```

Il file Midi verrà eseguito senza apportare modifiche espressive, se il file è riferito ad una esecuzione espressiva verrà eseguito nello stato in cui si trova e quindi in maniera espressiva.

3.7 Panello configurazioni

Tramite il percorso del menù

```
View->Configure Tools...
```

si accede ad un pannello di configurazione, come si può vedere da *Figura 3.15*; dalla finestra è possibile cambiare l'immagine di sfondo dell'applicazione. La modifica, al momento, non ha molto senso poiché non c'è la possibilità di definire le nuove posizioni assunte dagli aggettivi nella nuova immagine di sfondo. Alla successiva esecuzione del programma è caricata sempre la solita immagine di default dell'applicazione, vista in precedenza in *Figura 4.2*.

Figura 3.15: *Configure Tools*

3.8 Sviluppi futuri

L'espressività da dare ad una esecuzione è impartita dall'utente attraverso un piano, facendo uso del mouse. Una miglioria da apportare al programma è cambiare la modalità con cui viene data l'intenzione espressiva all'applicazione, scegliere una modo che sia meno restrittivo e più naturale per l'utente, per esempio:

- tramite il controller della wii (Nintendo);
- riconoscimento delle gesture tramite la Kinect (Microsoft);
- identificazione di determinate posizioni del corpo nello spazio fisico tramite webcam.

Al momento attuale il software consente di cambiare l'immagine di sfondo ma non di ridefinire le posizioni dei vari aggettivi, è necessario, quindi, caricare la nuova immagine con le stesse dimensioni e le stesse posizioni degli aggettivi della precedente. Un miglioramento al programma consentirebbe di caricare una qualsiasi immagine di sfondo e

ridefinire le nuove posizioni degli aggettivi.
Altro sviluppo interessante potrebbe essere quello di portare l'applicazione CaRo su mobile, utilizzando il touch oppure l'accelerometro del dispositivo per fornire l'intenzione espressiva.

Capitolo 4

Test applicazione CaRo 2.0

Durante lo sviluppo del progetto, per i test in itinere, è stato utilizzato il sintetizzatore interno al computer. Tale oggetto è in grado di riprodurre gli eventi midi inviati dall'applicazione CaRo. La qualità di riproduzione del suono è appena discreta, sufficiente per capire se l'esecuzione, dal punto di vista strutturale è in linea con lo spartito e le aspettative iniziali. Il programma CaRo, come spiegato in precedenza, esegue un brano in maniera espressiva a seconda dell'espressività impartita dall'utente attraverso un piano bidimensionale. La resa sonora in esecuzione è di fondamentale importanza e influenza l'azione che l'utente fornisce per ottenere l'effetto espressivo desiderato, diverso è il rendering sonoro diverso sarà il movimento sul piano espressivo. Per ottenere una ottima qualità del suono si collega l'applicazione CaRo, quindi il computer in cui è eseguita, ad un pianoforte Disklavier Yamaha, tale strumento è un normale pianoforte acustico dotato di attuatori nei tasti e nei pedali, tramite un apposito driver sono inviati gli eventi midi al pianoforte che è in grado di eseguirli. I test che si presentano nel prossimo paragrafo sono stati svolti utilizzando tale sistema. Test simili riferiti alla precedente versione dell'applicazione sono visionabili in Canazza et al. (2015).

4.1 Risultati Test

In questo paragrafo si riportano i risultati ottenuti dai test di alcune performance svolte con l'applicazione CaRo collegato al Disklavier, con l'intento di mostrare come il sistema lavora. Per svolgere tali test è stato scelto un estratto di *Per Elisa* di *L. van Beethoven* (vedi *Figura 4.1*), il brano si presta all'annotazione di numerosi segni espressivi e, allo stesso tempo, è scritto in maniera relativamente semplice, portando ad una facilità di visualizzazione e comprensione dei risultati. Mantenendo la stessa partitura sono state svolte sette performance di tipo differente: cinque performance sono state eseguite mantenendo sempre la stessa espressività (bright, hard, heavy, light, soft), il mouse durante la performance

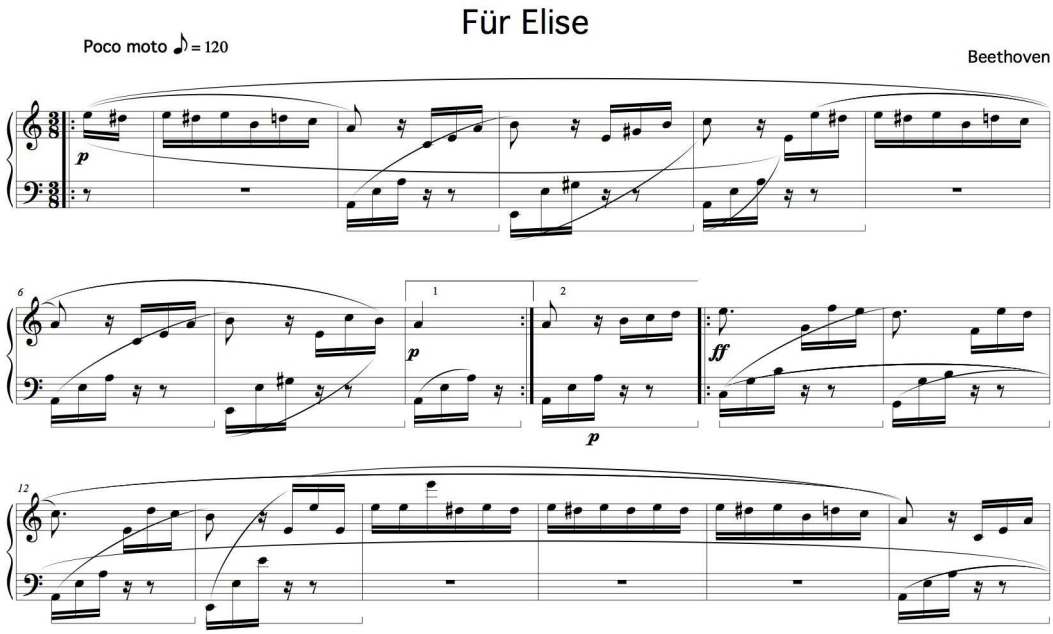

Figura 4.1: *Estratto di Per Elisa usato per effettuare i test*

restava fisso sopra l'etichetta corrispondente; una performance è stata eseguita eseguendo una traiettoria con il mouse nel piano bidimensionale come mostrato in *Figura 4.2*. Infine si esegue il brano in maniera meccanica, senza applicare modifiche di tipo espressivo o trasformazioni alle varie note, questo tipo di esecuzione è usata come esecuzione di base, utile ai fine del confronto con gli altri tipi.
Il sistema esegue le differenti performance tenendo conto della sequenza delle note scritte nello spartito, le intenzioni espressive sono calcolate in relazione alla posizione del mouse nel control space.
Si riporta come primo caso i grafici riferiti all'esecuzione di tipo meccanico. Sarà interessante poter confrontare gli altri tipi di esecuzione con questo caso di base. In *Figura 4.3* si riporta il piano roll dell'esecuzione di tipo meccanico. Un piano-roll è un metodo per rappresentare degli eventi di tipo musicale in modo da avere una visione di tipo grafico di alcuni dei più importanti parametri. Il grafico è di tipo bidimensionale: nell'asse verticale sono riportate le informazioni riguardanti le note mentre nell'asse orizzontale è riportato il tempo in secondi. Nel momento in cui una nota è suonata una nuova linea orizzontale è disegnata nel piano roll, la lunghezza della linea ne rappresenta la durata, le due estremità ne descrivono il tempo di inizio e fine. L'altezza a cui è riportata una singola sbarretta nel grafico superiore è in corrispondenza all'altezza della nota mentre nel grafico infe-

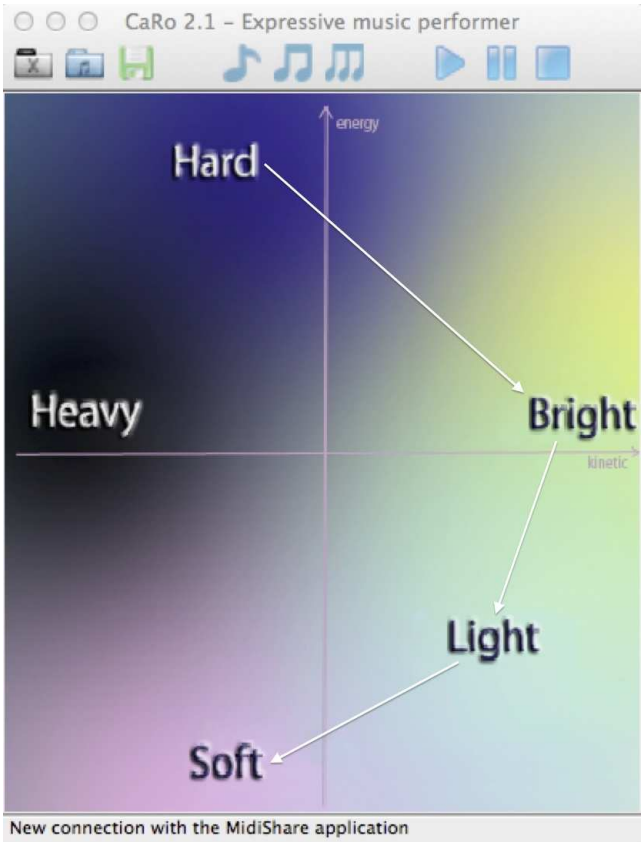

Figura 4.2: *Interfaccia dell'applicazione CaRo. Le frecce indicano la traiettoria eseguita durante la performance.*

riore in corrispondenza alla *key velocity* di quella determinata nota. Per quanto riguarda l'esecuzione di tipo meccanico non è apportata nessun tipo di modifica, la key velocity delle varie note risulta essere costante in quanto le dinamiche incontrate durante la lettura dello spartito non sono considerate; le durate delle varie note non sono alterate rispetto allo spartito originale.
Si riportano ora i piano roll relativi alle esecuzioni mono espressive, nell'ordine: heavy *Figura 4.4*, hard *Figura 4.5*, bright *Figura 4.6*, light *Figura 4.7*, soft *Figura 4.8*. In tutti i grafici si nota una discontinuità della *key velocity* dovuto al cambiamento di dinamica nello spartito da *p* a *ff* come si può vedere da *Figura 4.9*, nonostante il cambio imposto dall'indicazione riportata sullo spartito i valori iniziali e finali e, quindi, la differenza tra gli stessi è differente a seconda dell'espressività selezionata; valori di *key velocity* più bassi si osservano per la performance di tipo *soft* fino ad arrivare ai valori più alti per la

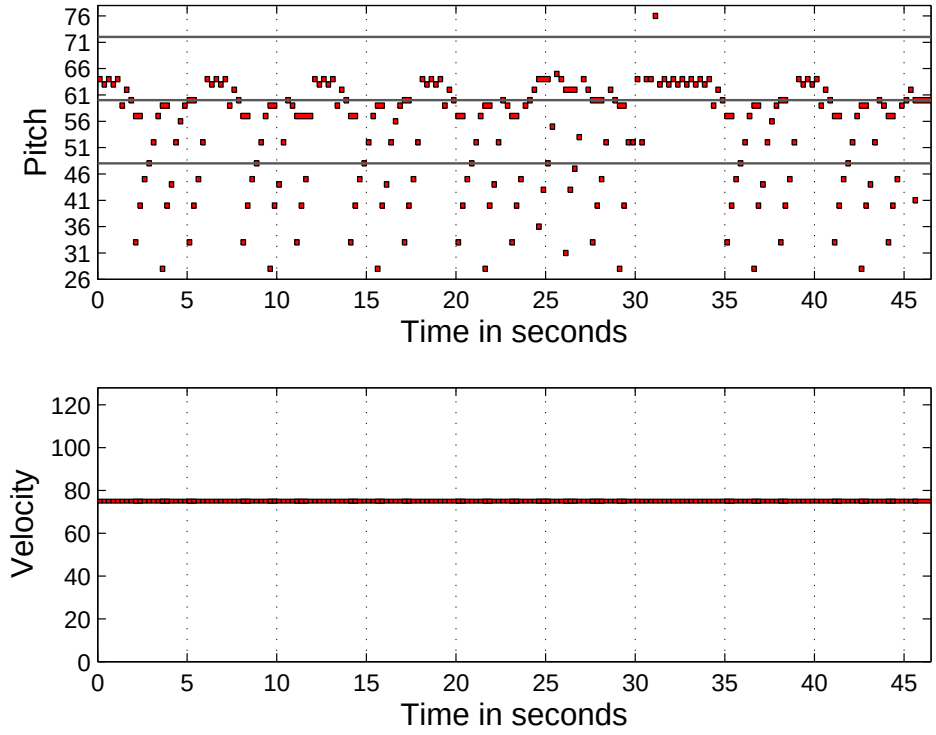

Figura 4.3: *Piano roll della performance di tipo meccanico*

performance di tipo *hard*. L'andamento della *keyvelocity*, per i diversi tipi di performance, si può apprezzare ancora di più dal grafico di *Figura 4.10*; in ordinata è rappresentato il valore della *key velocity* mentre in ascissa la progressione delle note. Oltre alla modifica alla *key velocity*, una determinata resa espressiva, influenza la durata delle note, dal piano roll si riesce ad apprezzare la diversa lunghezza dei vari trattini, i quali rappresentano le durate delle note. Nell'esecuzione di tipo *heavy* le note hanno durata maggiore rispetto a espressività più brillanti quali *bright* oppure *light*. Insieme alla durata delle note sono modificati anche gli istanti temporali in cui iniziano i vari eventi musicali, si riportano in tabella 4.1 i tempi in cui iniziano le prime dieci note nell'espressività di tipo *heavy* e *soft*, è riportato anche quello relativo alla performance meccanica come caso di confronto. I tempi di inizio nell'espressività di tipo *heavy* risultano successivi rispetto a quelli della performance di tipo *soft* e ulteriormente allogati rispetto all'esecuzione meccanica, fenomeno ancora più evidente se si usa la performance di tipo *bright* come confronto la quale riporta tempi precedenti rispetto all'esecuzione meccanica.

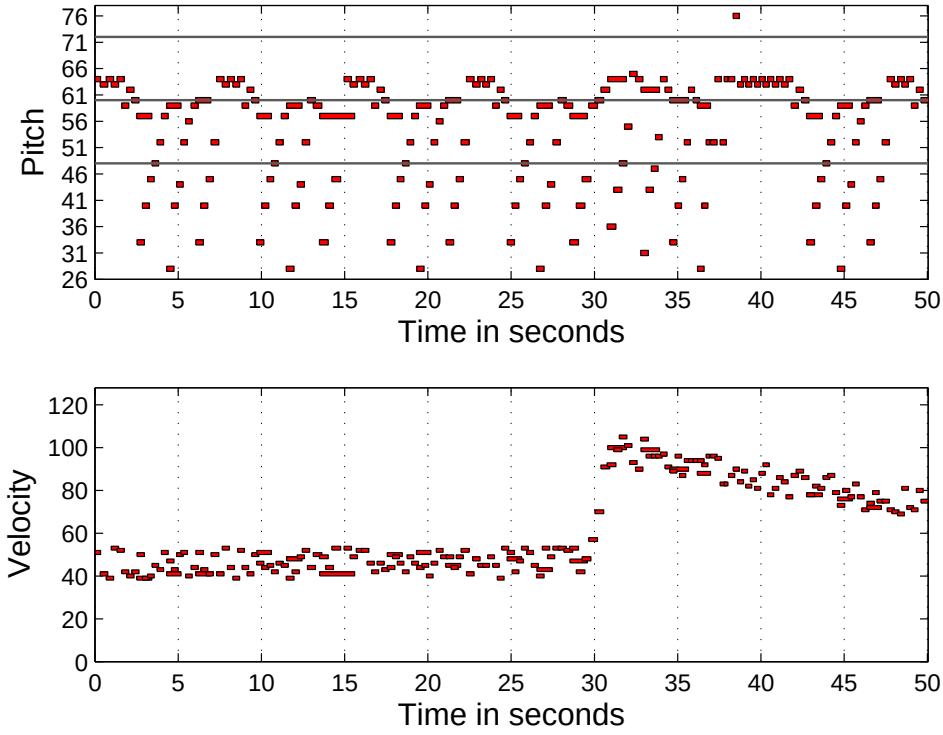

Figura 4.4: *Piano roll della performance heavy*

In *Figura 4.11* è riportato il piano roll della performance che è ottenuta compiendo con il mouse una traiettoria descritta all'inizio del paragrafo. Nel grafico che rappresenta la *key velocity* si riesce ad apprezzare quattro macro aree, una per ogni tipologia di espressività. Si osserva che anche la durata subisce delle variazioni sostanziali, all'inizio c'è un allungamento della durata nominale per poi passare ad una durata sempre più breve, in corrispondenza di *bright* e *light*, alla fine una distensione con l'espressività di tipo *soft*.
In tutte le performance di tipo espressivo si tengono in considerazione le legature inserite nello spartito per dare maggiore dinamismo al brano, come spiegato nel capitolo secondo. Tali segni espressivi influenzano in maniera significativa la *key velocity*, creando un effetto di fraseggio musicale ben definito. In *Figura 4.12* si esegue uno zoom sul grafico che rappresenta la *key velocity*, si nota come la velocità varia con andamento a parabola seguendo la struttura delle legature inserite nello spartito (vedi *Figura 4.13*).
I numerosi test presentati in questo paragrafo sono concordi alle aspettative iniziali, il sistema risponde in maniera opportuna al cambiare dell'espressività desiderata dall'utente,

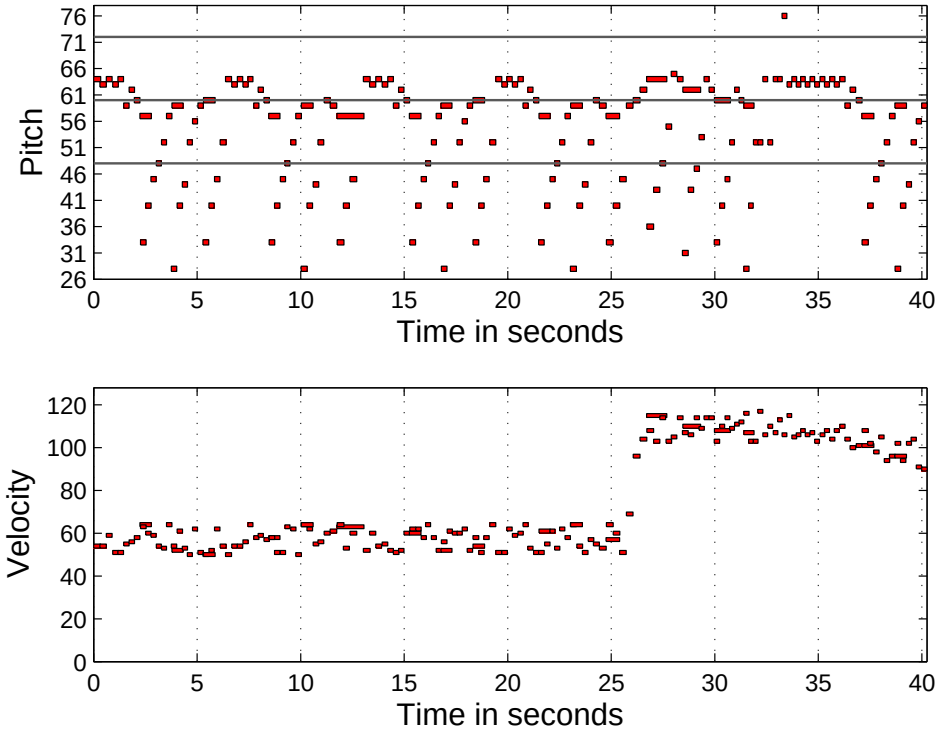

Figura 4.5: *Piano roll della performance hard*

impartita tramite il piano espressivo.
Sotto il profilo tecnico l'applicazione soddisfa appieno i requisiti, i test appena presentati ne sono la prova, resta da appurare, in maniera più dettagliata, se sotto il profilo espressivo e quindi psicologico il programma fornisce l'effetto desiderato dall'utilizzatore finale. Tale analisi rimane una questione aperta e quindi di maggior approfondimento, come spiegato brevemente alla fine del primo capitolo.

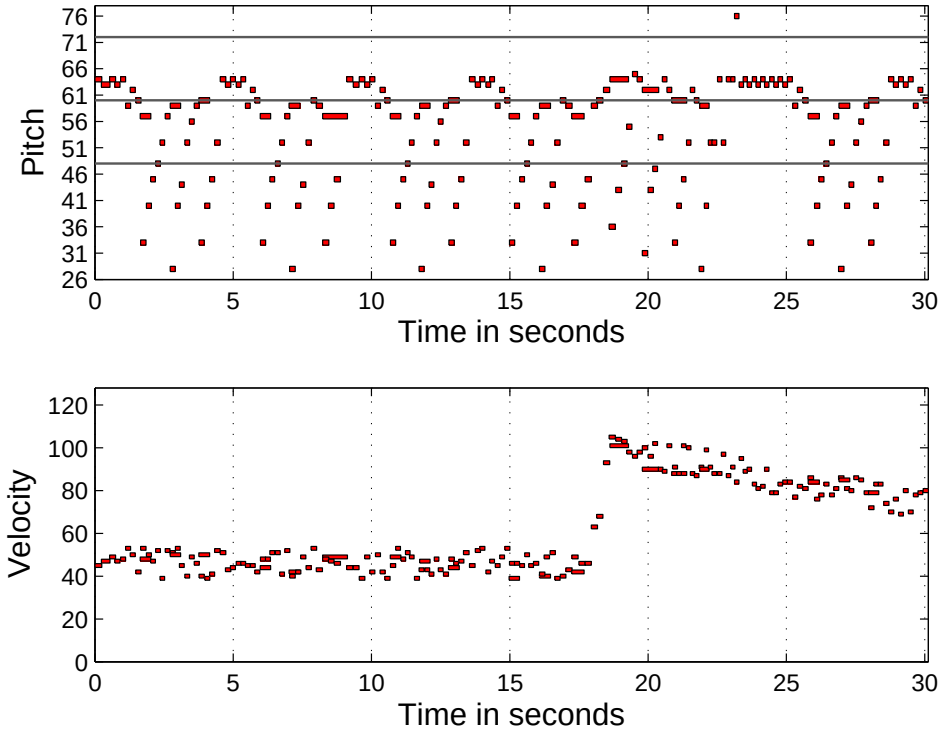

Figura 4.6: *Piano roll della performance bright*

# note number	Onset mecchanical	Onset heavy	Onset soft
1	0	0	0
2	0,5	0,648	0,592
3	1	1,348	1,124
4	1,5	1,964	1,706
5	2	2,65	2,216
6	2,5	3,22	2,754
7	3	3,84	3,25
8	3,5	4,434	3,824
9	4	5,04	3,844
10	4	5,052	4,354

Tabella 4.1: *Onset di inizio per le espressività heavy e soft rispetto al caso meccanico*

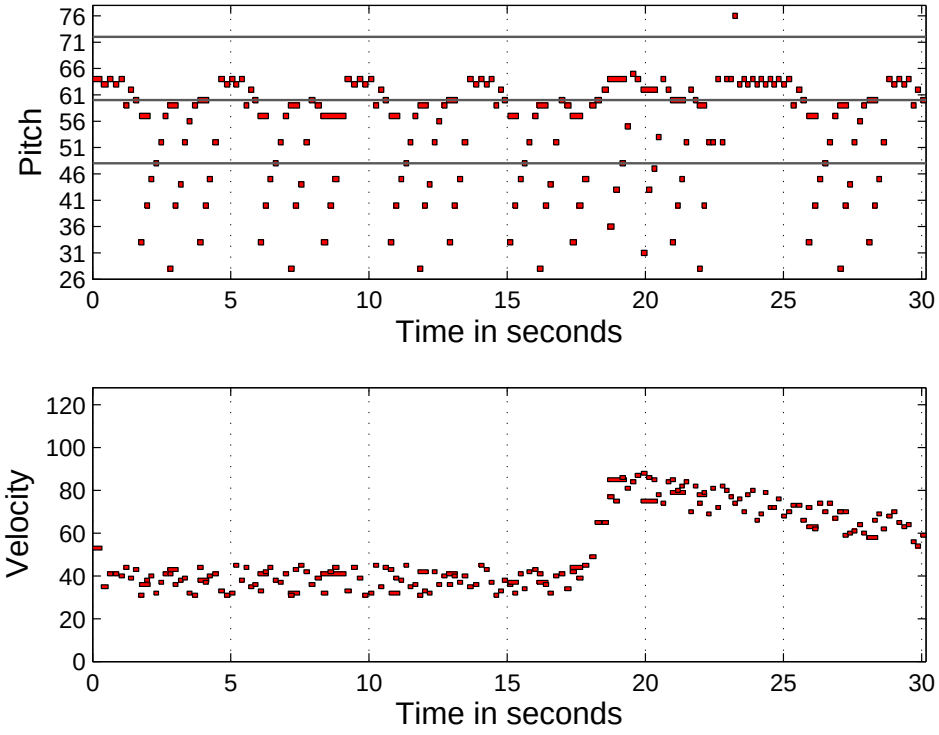

Figura 4.7: *Piano roll della performance light*

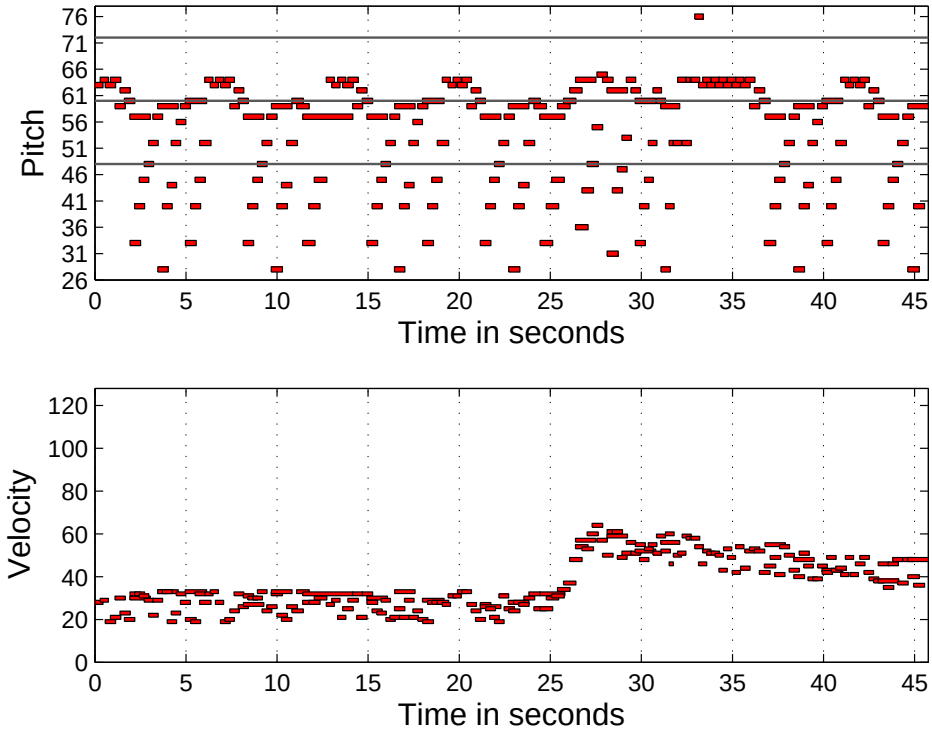

Figura 4.8: *Piano roll della performance soft*

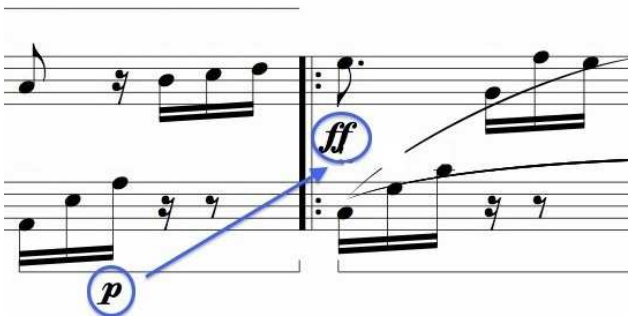

Figura 4.9: *Estratto di partitura*

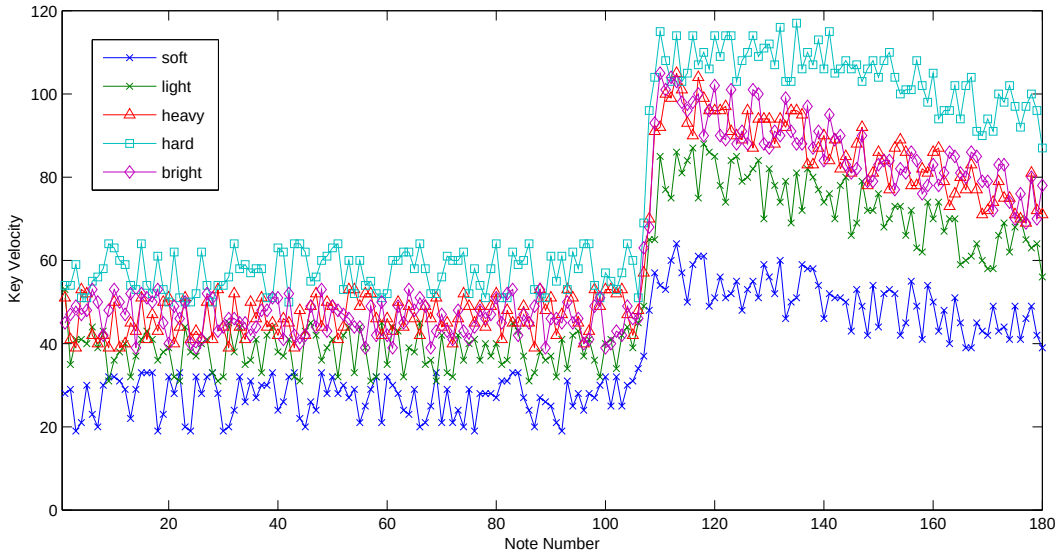

Figura 4.10: *Variazione della key velocity per le varie performance*

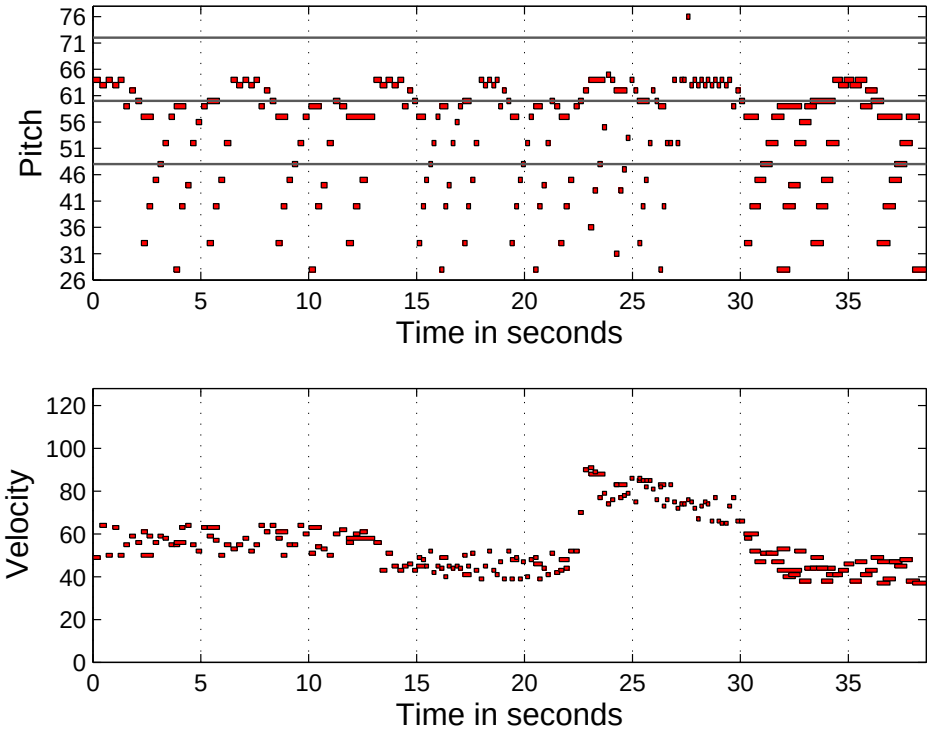

Figura 4.11: *Piano roll della performance compiendo una traiettoria*

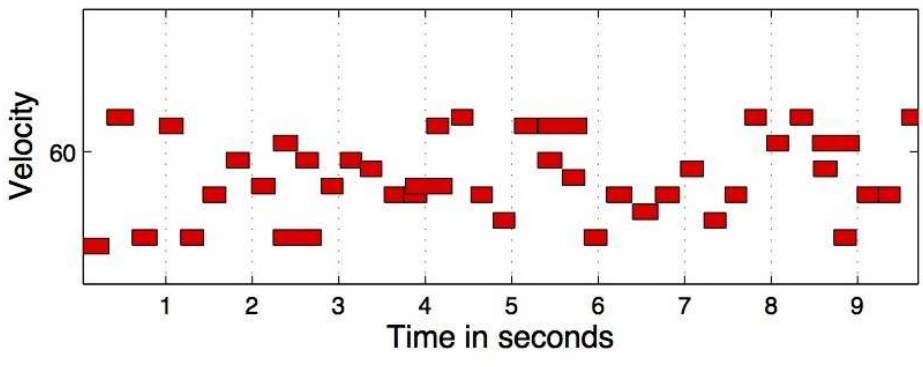

Figura 4.12: *Zoom della key velocity per una performance espressiva*

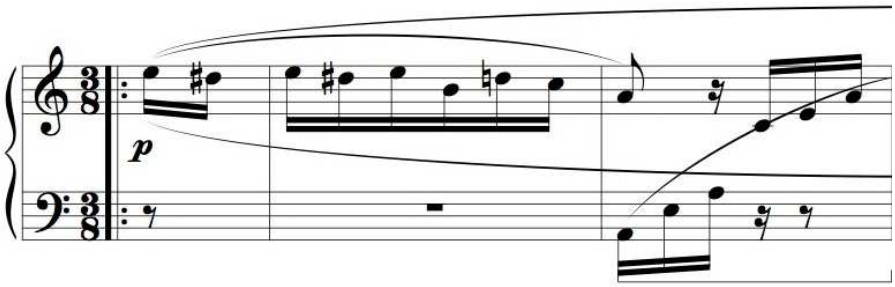

Figura 4.13: *Estratto di partitura che evidenzia le legature presenti*

Capitolo 5

Considerazioni finali

In questa tesi si è presentato un'applicazione informatica che esegue un brano in maniera espressiva utilizzando l'intenzione dell'utente.
Si sono descritti diversi metodi per la rappresentazione dell'espressività e si è cercato di capire la correlazione che sussiste tra variazione di determinate caratteristiche musicali ed emozioni. Un piano bidimensionale, solitamente utilizzato per la rappresentazione dell'espressività, è stato usato per impartire l'intenzione espressiva.
L'espressività è frutto di numerose varianti tecniche, strutturali, di esecuzione ecc, che vanno a coinvolgere la psicologia dell'individuo, e quindi la soggettività, con limitazioni, inoltre, di tipo linguistico e culturale; è impossibile, quindi, riuscire a creare un modello universale di espressività. In questa tesi sono presentati solo alcuni dei risultati relativi all'espressività musicale. Il campo di ricerca in questione è ancora molto giovane, e le possibili strade ancora da esplorare sono molteplici. Nonostante questo, si sono potuti apprezzare i primi risultati positivi dai test di laboratorio e dalle valutazioni date dal pubblico.
L'obiettivo dell'esecuzione automatica espressiva della musica non è di sottrarre agli esseri umani questo compito, bensì cercare di migliorare l'interazione con le macchine, utilizzando tecniche che risultano più naturali e meno sfrustanti, in modo particolare per utenti diversamente abili o comunque per persone inesperte in ambito informatico. Il metodo con cui l'utente interagisce con l'applicazione può essere di diversa natura, sono da preferirsi tecniche di riconoscimento dell'espressività poco o per nulla invasive. Dotare l'applicazione di questa possibilità è il prossimo obiettivo.

Bibliografia

Laura-Lee Balkwill, William Forde Thomson, and Rie Matsunaga. Recognition of emotion in japanese, western, and hindustani music by japanese listerners. *Japanese Psychological Research*, 46(4):337–349, 2004.

Lisa Feldman Barett. Discrete emotion or dimension? the role of valence focus and arousal focus. *Psychology Press Ltd*, 12(4):579–599, 1998.

Sergio Canazza, Giovanni De Poli, and Antonio Roda. Emotional response to major mode musical pieces: Score-dependent percepetual and acoustic analysis. Technical report, Universita di Padova, 2011.

Sergio Canazza, Giovanni De Poli, and Antonio Rodà. How do people asses computer generated expressive music performances. *Sound and Music Compuing Conference*, pages 353–359, 2013.

Sergio Canazza, Giovanni De Poli, and Antonio Rodà. Caro 2.0: An interactive system for expressive music rendering. *Hindawi Publishing Corporation*, (850474):13, 2015.

Antonio DeLisa. La musica di edgar varèse. *In Poesia- Filosofia delle poetiche e dei linguaggi*, 2005.

Tuomas Eerola, Rafael Ferrer, and Vinoo Alluri. Timbre and affect dimensions: Evidence from affect and similarity ratings and acoustic correlates of isolated instruments sounds. *An Interdisciplinary Journal*, 30(1):49–70, September 2012.

P. R. Farnsworth. The social phychology of music. *State univesity press*, 1969.

Johnny R.J. Fontaine, Klaus R. Scherer, Etienne B. Roesch, and Phoebe C. Ellsworth. The world of emotions is not two-dimensional. *Association for Pysccholological Science Journal*, 18(12):1050–1057, 2007.

Kate Hevner. Experimental studies of the elements of expression in music. *American Journal of Phycology*, 48:246–268, 1936.

Peter J. Rentfrow and Samuel D. Gosling. The do re mi's of everyday life: The structure and personality correlates of music preferences. *Journal of Personality and Social Psychology*, 84(6):1236–1256, 2003.

Bruno Haskins Repp. Probing the cognitive representation of musical time: Structural constraints on the perception of timing perturbations. *Journal of Acustical Society of America (JASA)*, 44:241–281, 1992.

Antonio Roda, Emery Schubert, Giovanni De Poli, and Sergio Canazza. Toward a musical turing test for automatic music performance. *11th International Symposium on Computer Music Multidisciplinary Research (CMMR) Plymouth, UK*, 2015.

J. A. Russel. A circumplex model of affects. *Social Psychol*, 39(6):1161–1178, 1980.

Ulrich Schimmack and Rainer Reisenzenin. Experiencing activation: Energetic arousal and tense arousal are not mixtures of valence and activation. *American Psychological Association Journal*, 2(4):412–417, 2002.

Emery Shubert. Update of the hevner adjective checklist. *Pecepetual and Motor Skills*, 96:1117–1122, 2003.

Emery Shubert, Giovanni De Poli, Antonio Rodà, and Sergio Canazza. The role of individual difference in judging expressiveness of computer-assisted music performaces byexperts. *ACM Transactions on Applied Perception*, 11(4), 2014a.

Emery Shubert, Giovanni De Poli, Antonio Rodà, and Sergio Canazza. Music systemisers and music empathisers - do they rate expressiveness of computer generated performances the same? *Sound and Music Compuing Conference*, 2014b.

J. A. Sloboda and P. N. Juslin. Psychological perspectives on music and emotion. *Oxford University Press*, 2001.

Sandrine Vieillard, Isabelle Peretz, Nathalie Gosselin, Stéphanie Khalfa, Lise Gagnon, and Bernard Bouchard. Happy, sad, scary and peaceful musical excerpts for research on emotions. *Psychology Press Ltd*, 22(4):720–752, 2008.

Yi Hsuan Yang and Homer H. Chen. Prediction of the distribution of perveived music emotions using discrete samples. *IEEE Computer Society*, 19(7):2184–2196, 2011.

Yi Hsuan Yang and Homer H. Chen. Machine recognition of music emotion: A review. *ACM Transactions on Intelligent Systems and Technology*, 3(3):40:1–40:30, 2012.

Zhihong Zeng, Maja Pantic, Glenn I. Roisman, and Thomas S. Huang. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *IEEE Computer Society*, 31(1):39–58, 2009.