Università degli Studi di Milano

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE e NATURALI

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE DELLA COMUNICAZIONE MUSICALE

Determinazione Automatica di Chroma Vector su brani in formato IEEE 1599

Kevin Andreoni

Tesi di Laurea

Relatore:

Prof. Luca Andrea Ludovico

Correlatore:

Dott. Adriano Baratè

Anno Accademico 2008/2009

Indice

CAP. 1 Introduzione

1.1 Contesto e obiettivi

Il formato standard IEEE 1599, sviluppato dai ricercatori del LIM (Laboratorio di Informatica Musicale) presso l'Università degli Studi di Milano, è in grado di rappresentare ogni livello di astrazione della musica in un unico file .xml: non nasce con l'intento di sostituire consolidati formati di file, ma con il fine di creare uno strumento a cui tutte le rappresentazioni digitali (partiture, testi, audio) della musica potessero far riferimento, costruendone un'indicizzazione spazio/temporale.

La caratteristica chroma, intesa in ambito musicale (come definito in [1]), è un versatile livello intermedio che associa una scala di colori alle diverse densità di note presenti in un istante di un brano musicale. Essa è definita attraverso un grafico avente per ascissa l'asse temporale, e per ordinata la suddivisione dei pitch di una scala musicale, privati della differenza di ottava.

Lo scopo di questa dissertazione è quello di integrare, attraverso un software, le potenzialità dei diagrammi costruiti mediante la componente chroma con il formato multi-strato di xml musicale IEEE 1599.

Nella fase preliminare del periodo di tirocinio sono stati raccolti diversi articoli scientifici con argomento fondante lo studio di grafici chroma (*chromagram*), per garantire alla ricerca una panoramica generale sulle procedure e sui possibili problemi riscontrabili nella creazione di chromagram. In tutte queste tesi il grafico è costruito per brani della cultura occidentale, avente come riferimento la scala temperata, pertanto le divisioni delle altezze sono le medesime 12 divisioni di semitono della scala, partendo dal DO e proseguendo con incrementi di mezzo tono fino alla nota SI. L'aspetto temporale del grafico è illustrato mediante la creazione di vettori, la quale lunghezza è proporzionale alla durata dell'evento musicale che rappresentano. Tali vettori verranno successivamente colorati in dipendenza della quantità di una determinata nota in un instante di tempo, effettuando una scalatura dal colore nullo (assenza di note) al colore pieno (massima concentrazione di note).
Le variabili utili per la crezione di chromagram sono pertanto pitch e durata di ogni evento che costituisce la tessitura di un brano musicale. Questo ha permesso l'estrazione di grafici chroma sia da partiture scritte, dove gli eventi sono descritti in linguaggio notazionale; che da brani audio, i cui eventi sono notevolmente meno distinguibili rispetto alla partitura d'origine, ma che sono comunque estrapolabili con analisi spettrale e particolari algotirmi statistici. Nella seguente tesi verrà sviluppato un algoritmo per il primo tipo di estrazione enunciata, ossia quello da partitura scritta.

La seconda fase del tirocinio si è concentrata sull'implementazione di un software che avesse come obiettivo la determinazione automatica di chromagram partendo da partiture digitali codificate in formato IEEE 1599. L'applicativo non è progettato per essere standalone, ma è integrato nel Framework-MX sviluppato per l'editing e l'utilizzo di file IEEE 1599. L'ambiente di sviluppo per la programmazione sarà il medesimo del Framework-MX: Microsoft Visual Studio con codice scritto in linguaggio C#.
La rappresentazione in un formato di xml musicale ben strutturato di una partitura, come può essere quello IEEE 1599, è il mezzo più fedele per la sua traduzione in dominio digitale: operando una mappatura di ogni simbolo notazionale presente nella partitura d'origine, incapsulata in tag xml, vengono aboliti gli errori inerenti alla semantica globale del brano. Questo aspetto è innovativo rispetto alle ricerche analoghe [3] che operano la traduzione esclusivamente con software OMR (Optical Music Recognition) che, non godendo della strutturazione di un file di xml musicale, approssima le ambiguità grafiche della partitura non attribuendo i corretti valori notazionali corrispondenti.

L’utilizzo pratico dei grafici chromagram è rivolto all’ambito MIR (Music Information Retrieval), difatti in letteratura si possono trovare numerosi applicativi che sfruttano le potenzialità della caratteristica chroma per sviluppare degli algoritmi di audio matching, key extraction, sincronizzazione audio/partitura e quant’altro.

La presente tesi ha come obiettivo l’integrazione della caratteristica chroma con il framework di lavoro del formato IEEE 1599, e si presta come punto di partenza per lo sviluppo di applicativi MIR che sfruttino la corrispondenza spazio/temporale dei flussi di informazione musicale ottenuta dalla strutturazione dei file IEEE 1599, con la versatilità dei grafici chroma vector.

CAP. 2 La chroma

2.1 Concetto di chroma

Negli anni ’60 Shepard, realizzando un’analisi sulla percezione del pitch [1], intuì che poteva essere interpretato come due differenti attributi: l’altezza di tono (espresso in Hertz) e la **chroma**. L’altezza di tono descrive il generale aumento di tono di un suono, in corrispondenza dell’aumentare della sua frequenza. La chroma, d’altro canto, ha la versatile caratteristica di essere per natura periodica in ogni ottava [2], pertanto due toni separati da un numero intero di ottave condividono lo stesso valore di chroma.

In definitiva, la componente chroma identifica su 12 diversi livelli i corrispettivi livelli di altezza di tono delle note [DO, DO#, RE, ... , SI] della scala temperata, evidenziando, in un unico vettore, la presenza di energia della medesima nota, eliminando gli intervalli di ottava. Sono state scelte le discretizzazioni di tono della scala temperata in quanto essa è la più consueta nei brani popular della cultura occidentale [2], ma questa scelta è da considerarsi univocamente relativa a questa ricerca dato che la chroma, scelta a priori la suddivisione da applicare, è derivabile da qualsiasi tipo di scala musicale.

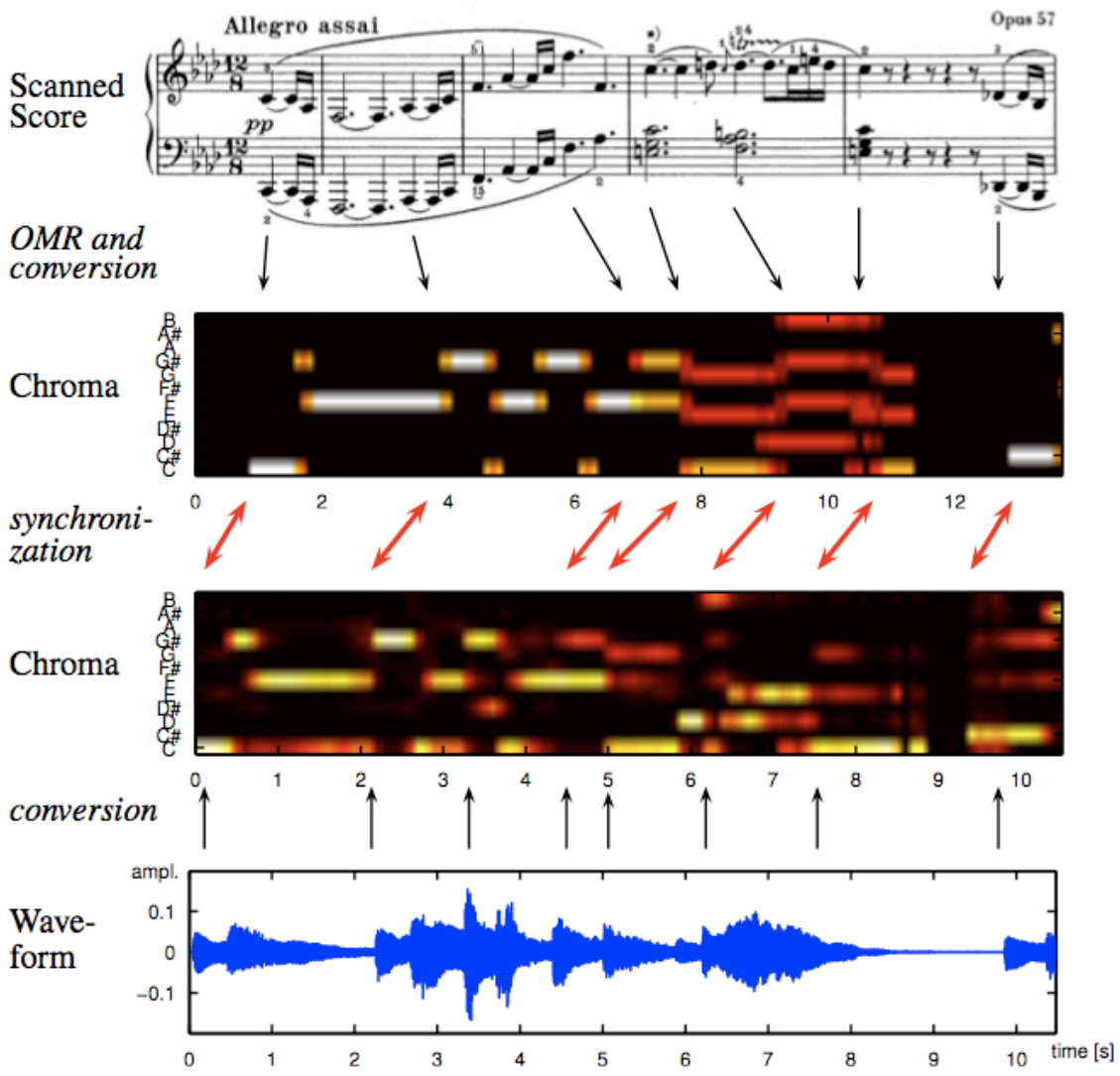

Fig. 1 - Confronto di chromagram estratto da partitura e chromagram estratto da audio, con le relative sincronizzazioni delle rappresentazioni grafiche.

Il grafico che illustra questa interpretazione del pitch è definito **chromagram** (ossia un insieme di **chroma vector**) [3]. L'asse delle y è suddiviso nei 12 livelli di altezza, mentre sull'asse delle x è indicato il dominio temporale. Il risultato della costruzione del chromagram sarà un insieme di vettori colorati proporzionalmente a seconda di quanta energia è presente in un determinato istante di tempo. Le variabili che andranno a realizzare i vettori del chromagram sono dunque le altezze discrete delle note e le loro relative durate, quindi sono caratteristiche estrapolabili da entrambe le rappresentazioni musicali attuali: quella scritta, sottoforma di partitura notazionale, e quella audio, sottoforma di vibrazioni dell'aria. Dato che questi valori possono essere veicolati da diversi mezzi, le procedure per l'estrazione avranno degli algoritmi e dei problemi differenti.

I valori delle altezze:

- *partitura*: sono evidenziati dalla posizione verticale delle note, rispetto ai righi del pentagramma, e dalle relative chiavi (di violino, di basso) e alterazioni in chiave corrispondenti.

- *audio*: sono deducibili dall'analisi spettrale del relativo brano, quindi valori espressi in Hertz. Come riferimento standard per la scala temperata, si utilizza il LA della terza ottava del pianoforte con frequenza 440Hz, le note superiori e inferiori verranno calcolate proporzionalmente.

I valori delle durate:

-*partitura*: seguono la suddivisone, espressa dai simboli notazionali (croma, semi croma, minima, semi minima, etc.), della battuta. La durata complessiva della battuta è scritta all'inizio del pentagramma in forma frazionaria (4/4, 3/4, 7/8). Le durate degli eventi saranno pertanto delle frazioni che sommate restituiranno il valore della durata della battuta.

-*audio*: l'unità di misura è quella dei secondi e i valori sono ricavabili dall'analisi temporale dello spettro. La durata degli eventi audio è un fenomeno molto poco musicale, e pertanto fa affidamento alla relativa esecuzione del brano in esame, non alle durate simboliche della musica scritta nella relativa partitura.

I due differenti algoritmi per l'estrazione di chroma vector da partitura musicale scritta e da brani audio verranno trattati nei paragrafi successivi.

Il concetto di chroma è strettamente legato all'aspetto musicale, piuttosto che a quello fisico, di interpretazione della percezione del pitch, ed è proprio questo che rende i chromagram degli strumenti molto validi per eseguire algoritmi di Music Information Retrieval quali audio matching o sincronizzazione audio-partitura. Per effettuare queste operazioni sarà necessario l'impiego di entrambi i chromagram (quello ricavato dalla partitura e quello ricavato dall'audio) che, dopo un adeguato confronto, riusciranno a garantire l'univocità del brano musicale sottoposto.
Nel paragrafo 2.4 verranno presentati gli algoritmi MIR che usufruiscono dei chroma vector trovati fin ora in letteratura.

2.2 Estrazione di chromagram da partitura

Per la creazione di chromagram da partitura è necessario un livello intermedio [3] che traduca i simboli notazionali in corrispondenza dei loro effettivi valori di altezza e durata, digitalizzando, al meglio delle possibilità, ogni aspetto di semantica musicale della partitura d'origine. Questo livello intermedio, nella maggior parte degli studi, è ottenuto da un software di Optical Music Recognition (OMR), mentre l'obiettivo primo di questa tesi è di ottenere questa traduzione attraverso un file IEEE 1599 fortemente strutturato, capace quindi di ottenere un risultato molto più preciso, eliminado gli errori tipici di una digitalizzazione OMR. Il formato standard IEEE 1599 di xml musicale risulta essere un buon compromesso per eseguire questa operazione, confermandone peraltro la versatilità di utilizzo e ampliando le potenzialità del Framework-MX sviluppato nel L.I.M. (Laboratorio di Informatica Musicale). In seguito verranno presentate entrambe le procedure per l'estrazione della chroma da partitura.

2.2.1 Digitalizzazione di partiture attraverso Optical Music Recognition

Un software di Optical Music Recognition (OMR) riceve in input una partitura digitale, ne fa una mappatura di tutti i simboli (segnature di tempo, armatura di chiave, pause, note), localizzandoli mediante le coordinate (x,y) dei pixel dell'immagine e infine restituisce come output la digitalizzazione della partitura in formato binario (.niff, ormai obsoleto) oppure direttamente in un tipo di file proprietario per software di editing di partitura (.mus per Finale, .viv per VivaldiScan).

Program	*Publisher/type*	*Input*	*Output*	*Platforms*	*Web site*
Capella-Scan 6.1	Capella Software	bmp, gif, pdf, png, PS, tif, scanner	Capella, MIDI, MusicXML	Windows 95 or above	www.capella-software.com/capscan.htm
MIDI-Connections Scan 1.3	CAS	tif, scanner	MIDI	Windows 95 or above	www.midi-connections.com/Product_Scan.htm
MP Scan 2	Braeburn Software	bmp, scanner	Music Publisher	Windows 95 or above	www.braeburn.co.uk/mpsinfo.htm
NoteScan	AMNS	tif	Nightingale	Mac OS 9 & X	www.ngale.com
OMeR (Optical Music easy Reader) 2.1	Myriad	gif, jpg, pdf, png, tif, etc. (via QuickTime)	Melody/Harmony Assistant	Windows 95 or above, Mac OS 8.6 - X	www.myriad-online.com/en/products/omer.htm
PhotoScore 4.2.0	Neuratron	bmp, pdf, scanner	MIDI, MusicXML, NIFF	Windows, Mac OS	www.neuratron.com
ScoreMaker FX Pro	Kawai	??	MusicXML, other(s)	Windows 98 or above	www.kawai.co.jp/cmusic/products/scomwin
Scorscan 1.3	npcImaging	tif, scanner	SCORE	Windows 95 & 98	www.npcimaging.com/scscinfo/scscinfo.html
SharpEye 2.68	Visiv	bmp, tif, scanner	MIDI, MusicXML, NIFF	Windows 95 or above	www.visiv.co.uk
SmartScore Pro 5.3.1	Musitek	tif	Finale, MIDI, NIFF	Windows, Mac OS 9(?) & X	www.musitek.com/smartscre.html
Vivaldi Scan	goVivaldi	bmp, tif	Vivaldi, XML, MIDI	Windows 95 or above, Mac OS 7 - 9.2.x(?)	www.vivaldistudio.com/ENG/VivaldiScan.asp

Fig. 2 - Tabella comparativa dei principali software OMR.

Il grado di bontà dei risultati del software OMR dipende fortemente dalla qualità dell'immagine originale della partitura e dalla complessità dei simboli notazionali contenuti in essa. Dato che nella tradizionale notazione sono presenti simboli che veicolano informazioni inerenti l'intero brano (segnature di chiave, segnature di tempo, staff) il software OMR potrebbe restituire dei risultati compromettenti per la semantica globale: se non viene riconosciuta un'alterazione di chiave all'inizio del pentagramma verranno modificate tutte le note in corrispondenza dell'alterazione mancante, cambiando completamente la tonalità del brano. Un altro eventuale errore potrebbe essere riscontrato nella ricognizione della durata di un evento, specialmente per brani con segnature di tempo irregolari (5/4 , 7/8) oppure con gruppi irregolari di note (terzine), che potrebbero falsare la durata effettiva delle singole battute.

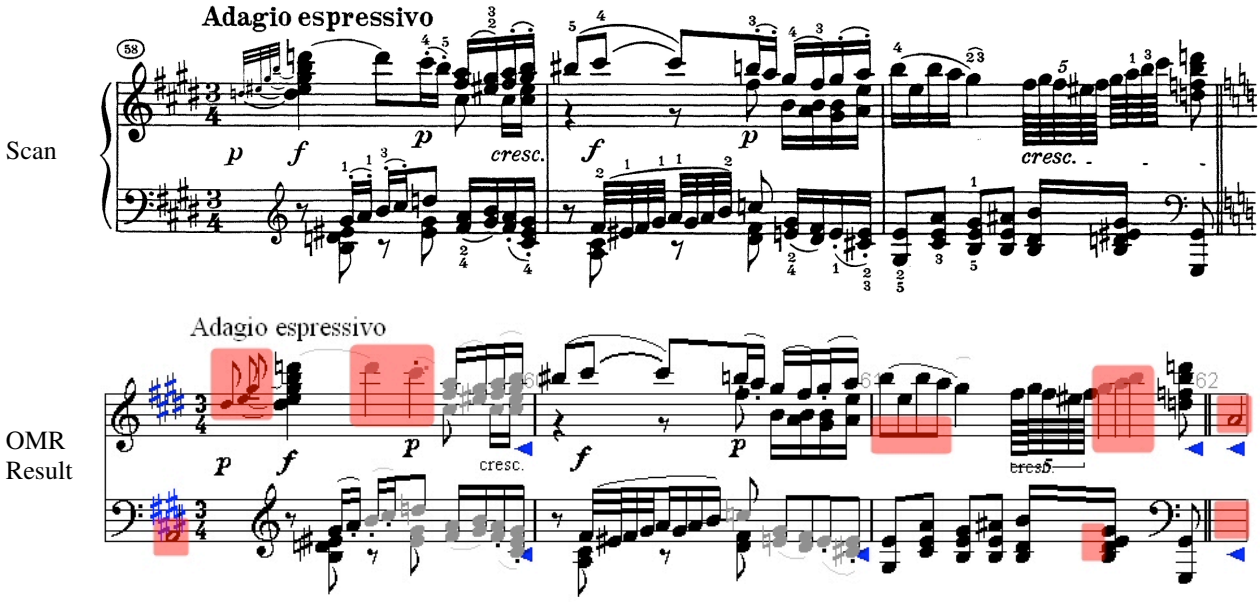

Fig. 3 - Risultato di una scansione OMR. Gli errori sono evidenziati in rosso, la freccia blu indica un errore nella durata totale della battuta.

Quindi per avere un file OMR efficace sarebbe opportuno (e in certe situazioni strettamente necessario) operare la mappatura con diversi prodotti OMR e successivamente verificare e confrontare la qualità dei singoli output.

Gli applicativi MIR che sfruttano questo tipo di digitalizzazione [3 5] dovranno pertanto essere in grado di gestire gli errori effettuando approssimazioni, che comunque soddisfino l'efficacia generale del software.

2.2.2 Digitalizzazione di partiture utilizzata in IEEE 1599

L'alternativa che si vuole proporre permette di svolgere al meglio la funzione di mappatura della partitura rispetto a un software OMR e consiste nell'utilizzare un linguaggio di xml musicale capace, dopo un adeguata strutturazione, di gerarchizzare una partitura grazie all'utilizzo di tag xml. L'utilizzo di questo linguaggio per la creazione del livello intermedio tra partitura cartacea e partitura digitale è in grado di non alterare la semantica originale, risultando il più fedele metodo per la digitalizzazione di partiture.

Per questo elaborato si è scelto il formato IEEE 1599 (trattato in dettaglio nel capitolo 3), che è uno standard che incapsula su più livelli i diversi contenuti multimediali (audio/ video), simbolici (notazione) e grafici (partitura) di un brano musicale, etichettati opportunamente all'interno di un file .xml. Nello specifico caso di questo formato, la creazione del file .xml è frutto dell'esportazione della partitura già digitalizzata tramite un plug-in del software per editing notazione Finale, che converte automaticamente la partitura .mus in un file di xml musicale che rispetta le regole del DTD del formato IEEE 1599.

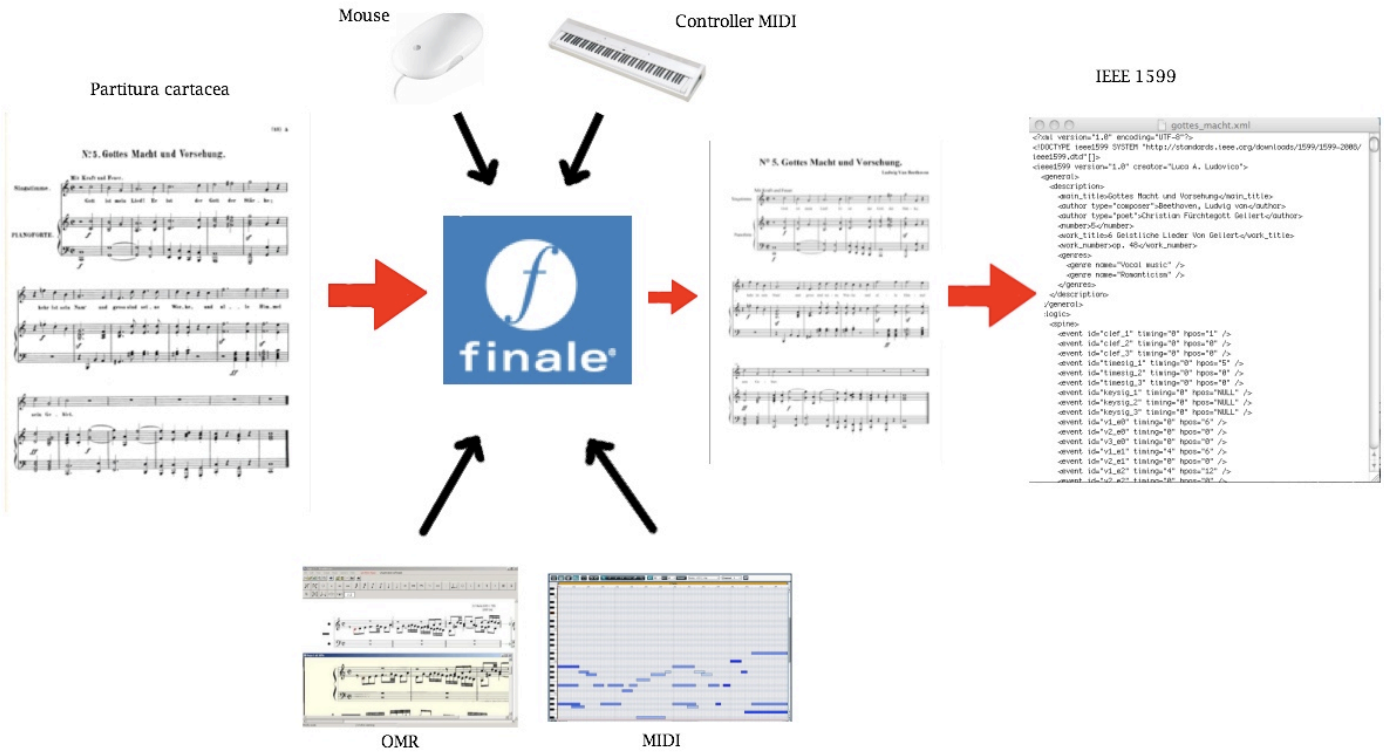

Fig. 4 - Schema a blocchi della digitalizzazione operata in IEEE 1599

Il processo di inserimento della simbologia notazionale all'interno della partitura in Finale varia in dipendenza della tipologia di brano preso in esame, cercando di riuscire a garantire la corrispondenza perfetta delle due partiture. Il metodo sicuramente più affidabile, ma in contemporanea il più dispendioso in termini di tempo, è l'inserimento tramite mouse (o controller MIDI) di ogni singolo simbolo; per riuscire ad automatizzare questo passaggio si è pensato di partire da una base approssimata già editabile in Finale, come l'importazione del corrispettivo file MIDI o della scansione con software OMR. In questo modo il file .xml che verrà creato sarà la miglior rappresentazione in termini grafici ma soprattutto semantici della partitura cartacea di partenza.
Garantire questa univocità tra partitura scritta e partitura digitale aumenta notevolmente il grado di precisione della conversione effettuata con OMR, e di conseguenza il grafico chromagram non avrà delle ambiguità e evidenzierà con maggior dettaglio ogni aspetto rilevabile dalla notazione musicale simbolica in dominio digitale.

2.3 Estrazione dei chroma vector da audio

Per la creazione di chromagram dalla clip audio il procedimento è computazionalmente più complesso ed è stato al centro di numerose ricerche, infatti in letteratura possiamo trovarne diversi algoritmi [2 4 6], la maggior parte dei quali arriva al risultato con approssimazioni statistiche, ma senza intaccarne la globale efficacia.
L'estrazione del grafico può essere vista a grandi linee come la suddivisione dello spettro secondo le bande frequenziali scandite dalle altezze delle note (nel nostro caso il riferimento è LA 440 Hz), e la successiva normalizzazione dei vettori appena costruiti nelle 12 rispettive divisioni del chromagram, eliminando i salti di ottava. Una volta discretizzato l'asse y, i tempi dei singoli eventi che saranno i valori dell'asse x saranno calcolati sommando gli intervalli di finestratura spettrale in cui è presente l'energia relativa alla banda frequenziale della nota evidenziata nel vettore.
Questa dissertazione non ha come scopo la creazione di chromagram da brani audio, ma è stato scelto di illustrarne un metodo per dare completezza alla ricerca, in quanto per la maggior parte delle applicazioni MIR *chroma-based* sono utilizzati in combinazione entrambi i diagrammi chroma visti in precedenza.

2.3.1 Algoritmo per l'estrazione

La procedura che viene presentata è quella di Muller [4] in quanto possiede un alto grado di robustezza alla variazione di parametri quali timbro, dinamica, articolazioni e leggere deviazioni di tempo (come ad esempio i trilli). Per semplificare la trattazione la procedura verrà divisa in due fasi: nella prima è stata utilizzata una piccola finestra di analisi spettrale per identificare le 12 diverse distribuzioni di energia da attribuire ai 12 livelli di altezza del chromagram, successivamente è stata usata una finestratura maggiore per compiere delle analisi statistiche, i cui valori saranno attribuiti a una soglia che avrà il compito di normalizzare il chromagram ottenuto.

Il primo step viene così diviso:

1) Il segnale audio viene scomposto in 88 bande frequenziali corrispondenti al valore delle note comprese tra A0 e C8 (il range di una tastiera di pianoforte tradizionale). Per la scomposizione è stato utilizzato un banco di filtri (chiamato pitch bank filter [5]) costituito da dei filtri ellittici che si prestano in modo ottimale a questo tipo di elaborazione.

2) Viene applicata la convoluzione tra le 88 sottobande frequenziali e la STMSP (short-time mean-square power) con una finestratura rettangolare di 200ms in overlap per metà con quella precedente.

3) Vengono normalizzate le 88 finestrature negli appositi 12 livelli del chromagram, ripetendo la finestratura STMSP su tutte le sottobande relative ad ogni altezza. Per esempio per la creazione del chroma vector per la nota C, verranno unite le finestrature delle sottobande relative a C1,C2,C3,...,C8.

4) Infine, verrà ripetuto lo step **3)** per ognuno dei 12 livelli, così da ottenere un vettore con 12 dimensioni corrispondenti alle relative finestre d'analisi.

Il segnale audio d'ingresso è stato dunque convertito in una sequenza di vettori che contengono il valore della chroma per ognuno dei 12 livelli relativi a una finestratura di 200ms, quindi ottenendo 10 vettori per secondo. Questo è molto utile per l'elevata sensibilità ai cambi localizzati di distribuzione di energia, come le piccole variazioni di articolazioni o di tempo.

Il secondo compito dell'algoritmo è quello di garantire una corretta rappresentazione della distribuzione di energia anche su finestrature di maggior grandezza. Per svolgere questo compito è necessario quantizzare ogni vettore, discretizzando con numeri interi i valori di ampiezze delle distribuzioni di energia. Successivamente è possibile effettuare la convoluzione con ogni sequenza di vettori interi e, in questo specifico algoritmo, è stata utilizzata una finestratura di Hann di grandezza 41.

Il risultato di quest'ultima finestratura è una sequenza di 12 chroma vectors interi non negativi, che rappresentano una sorta di pesatura statistica della distribuzione di energia su 41secondi di segnale. Questa normalizzazione della distribuzione di energia è definita CENS features [5] (Chroma Energy distribution Normalized Statistics).

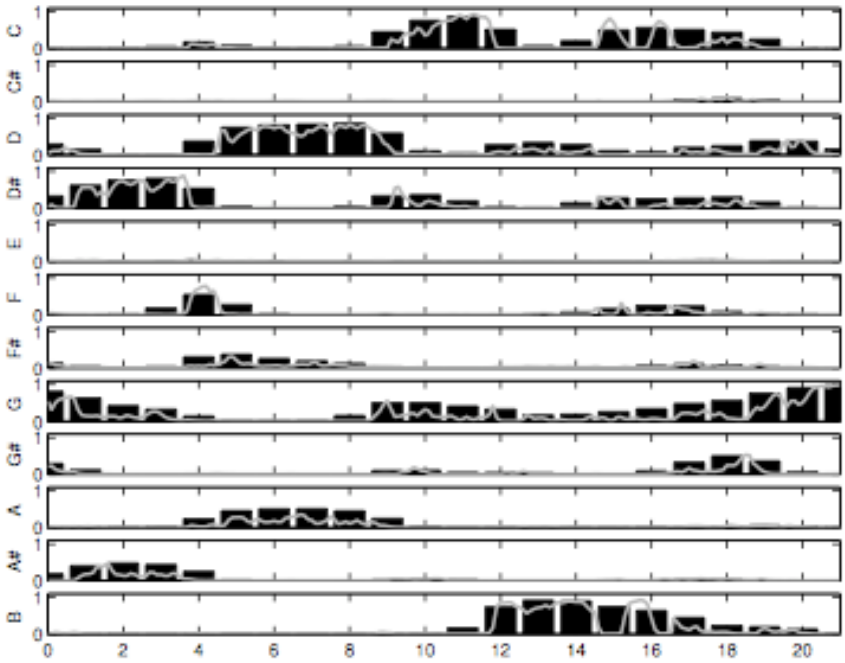

Fig. 5 - CENS features estratto dai primi 21 secondi della Quinta sinfonia di Beethoven.

L'algoritmo CENS appena illustrato è eseguibile su file audio non compressi (.wav .aif), quindi senza perdita di informazione. Esistono altri algoritmi che invece sfruttano la compressione .mp3 del brano in input utilizzando le sottobande del filtro polifasico per analizzare le frequenze indicanti le note, evitando di utilizzare il pitch bank filter.

2.4 Utilizzi in ambito MIR

Data la grande capacità di far corrispondere biunivocamente un brano audio con la sua relativa partitura, i diagrammi chroma vector sono stati largamente utilizzati per progettare dei nuovi applicativi di **Music Information Retrieval**, ossia quei software che offrono la possibilità di recuperare informazioni su di un brano musicale non basandosi solamente sui metatags indicati, ma utilizzando le caratteristiche intrinseche della musica, quali altezza, timbro, ritmo, etc. , per riuscire a svolgere algoritmi di ricerca e di analisi.

Le tipologie di software MIR che utilizzano il chromagram sono, per la maggior parte, quelli che sfruttano la potenzialità della corrispondenza biunivoca per effettuare audio matching. Con il termine audio matching viene inteso quel processo per cui automaticamente viene riconosciuta una musica in un sistema di ricerca elettronico, e un validissimo esempio di matching realizzato grazie all'uso di chroma vector su 12 dimensioni è quello dell'Università di Bonn, "Audio matching via chroma-based statistical features", dove viene svolta la procedura illustrata nel punto 2.3 confrontando statisticamente i chromagram di volta in volta ottenuti. Questo confronto statistico è ampliabile ad altri algoritmi di MIR come l'identificazione di canzoni cover, utilizzando come campione di partenza il chromagram della versione originale del brano [5].

Un'ulteriore applicazione trovata in rete [6] opera creando dei brevi campioni rappresentativi (audio thumbnails) del chorus o del ritornello di brani musicali contenuti in un database, identificando le ripetizioni e i pattern ritmici. Sfruttando i diagrammi chroma, viene data l'opportunità di effettuare un browsing veloce ed efficiente basato sulla preview dei ritornelli delle canzoni contenute.

Esistono altri algoritmi, non prettamente di audio matching, che utilizzano i chromagram:

I. Quello "Lyrics-based audio retrieval" [7] che permette di effettuare matching tra il testo e il relativo brano audio. Questo algoritmo sfrutta il collegamento sincrono tra testo e melodia tipico del MIDI (utilizzato nei file .kar, basi musicali MIDI per karaoke), mettendolo in relazione col relativo file audio attravesto la creazione di chroma vectors.
II. L'algoritmo presente nell'articolo "Musical Key extrction from audio" [2] che dato in input un brano musicale, ne calcola il chromagram, e effettua un'analisi sulle altezze e sulle relative alterazioni, restituendo la tonalità dell'audio relativo.

III. La procedura per ottenere la sincronizzazione automatica di audio e partitura [3]. Questo algoritmo effettua una sincronizzazione tra una partitura cartacea digitalizzata tramite software OMR e un file audio non compresso, procedendo per battute o per fraseggi significativi ma non per singoli simboli musicali, in quanto la digitalizzazione può presentare errori che potrebbero falsare l'effettiva sincronia audio-visiva che si vuole ottenere.

Questo elenco di algoritmi MIR mette in evidenza la versatilità dei diagrammi chroma vectors, che possono essere un valido e potente livello intermedio con la capacità di far corrispondere e di mettere in relazione le diverse rappresentazioni analogiche e digitali della musica.

CAP.3 IEEE 1599

3.1 Quadro generale del formato IEEE 1599

Il formato di xml musicale IEEE 1599, più comunemente conosciuto sotto l'acronimo di **MX** (ossia il nome del relativo framework di sviluppo), è divenuto standard internazionale nel 2008 ed è nato con lo scopo di rappresentare ogni livello di astrazione della musica in un unico file .xml, in quanto esistono numerosissimi formati per la rappresentazione dei diversi aspetti musicali, ma ancora non era presente una struttura che li contenesse e li mettesse in relazione spazio/temporale.

I livelli di astrazione considerati nel formato sono: le rappresentazioni multimediali (audio/ video) dell'opera, la partitura notazionale simbolica, la rappresentazione sintetica (es. MIDI, Csound) e la segmentazione tramite, ad esempio, Reti di Petri. Le molteplici caratteristiche vantaggiose dei file xml come la strutturazione gerarchica, l'intellegibilità, l'estendibilità e l'interscambiabilità in rete hanno trovato un'ottima applicazione in campo musicale, infatti in IEEE 1599 l'informazione è codificata su sei diversi strati (layers), ognuno dei quali fortemente relazionato con gli altri, necessitando dunque di una forte strutturazione capace di gestire i diversi contenuti riguardanti l'opera.

Gli strati in cui viene descritta interamente l'informazione musicale di un opera sono:

General: viene considerata l'opera nella sua interezza, raggruppandone, se presenti, le diverse istanze della stessa. Qui sono indicati, nell'elemento *description,* i metatags tipici quali titolo, autore, genere, etc. .

Music logic: è il cuore dell'organizzazione logica e strutturale dei file MX ed è composto a sua volta da 3 sotto strati.

Spine: implementa l'integrazione spazio/temporale che relaziona i diversi layers. Nel paragrafo seguente ne verrà data un panoramica dettagliata, in quanto è stato al centro delle problematiche relative al software Chroma Vectors.

Los (Logically Organized Symbols): racchiude il contenuto informativo simbolico del brano inteso come insieme di simboli notazionali scritti in ordine di battute.

Layout: aspetto grafico di impaginazione.

Structural: è il risultato della segmentazione dell'intero brano, ossia il frutto di un analisi musicologica.

Notational: si riferisce alle istanze visive dell'opera. Le rappresentazioni possono essere di tipo notazionale (NIFF) oppure grafico (pdf, jpeg) e identificano la rappresentazione spaziale dello spine, che ne permette la localizzazione.

Performance: incapsula una produzione musicale sintentica dell'opera. In questo layer sono presenti i parametri delle note da eseguire (sottoforma di file MIDI) e i suoni che essi devono avere (vengono utilizzati linguaggi per la sintesi del suono come Csound), creando delle diverse rappresentazioni denominate *performance_instance.*

Audio: è lo strato più basso dell'astrazione musicale, ed è rappresentato dalla traccia audio del brano. Sono ammessi formati sia compressi che non compressi, in quanto l'effettivo utilizzo è legato alla temporizzazione degli eventi musicali, e non alla qualità dell'informazione veicolata.

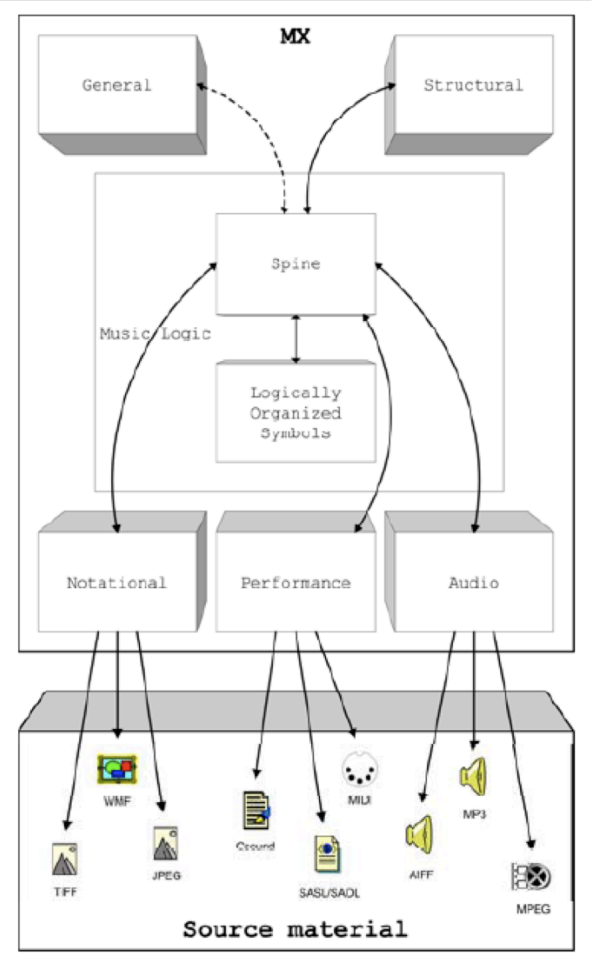

Fig. 6 - Schema a blocchi del formato IEEE 1599

Come evidenziato dalle relazioni in figura x.x, lo **spine** è il fulcro che garantisce l'interazione fra i diversi layers Notational, Performance, Structural e Audio con il layer Music Logic, permettendo a ogni evento considerato nel file MX di avere un riferimento spaziale e temporale in ogni modalità di astrazione musicale intesa nei diversi livelli.
In definitiva è possibile riassumere gli scopi del formato IEEE 1599 in un'ottimizzazione delle informazioni musicali, operando attraverso:

***Strutturazione*:** divisione multistrato delle informazioni, con relative segmentazioni di audio e partiture.

***Interazione* :** grazie alla presenza dello spine multistrato è possibile far interagire i vari livelli.

***Indicizzazione*:** fornire uno strumento valido per la ricerca di autori, brani, melodie etc. con la maggior completezza di informazioni legate all'opera, grazie alla creazione di appositi indici univochi relativi agli eventi.

3.2 Analisi dello spine

Il sub-layer indicato con il nome di spine rappresenta la struttura logica che implementa la correlazione temporale (attraverso l'attributo *timing*) e quella spaziale (attraverso l'attributo *hpos*) fra i diversi tipi di informazioni contenute nei livelli Notational (partitura), Performance (produzione sintetica) e Audio (traccia audio) che compongono il file in formato IEEE 1599.

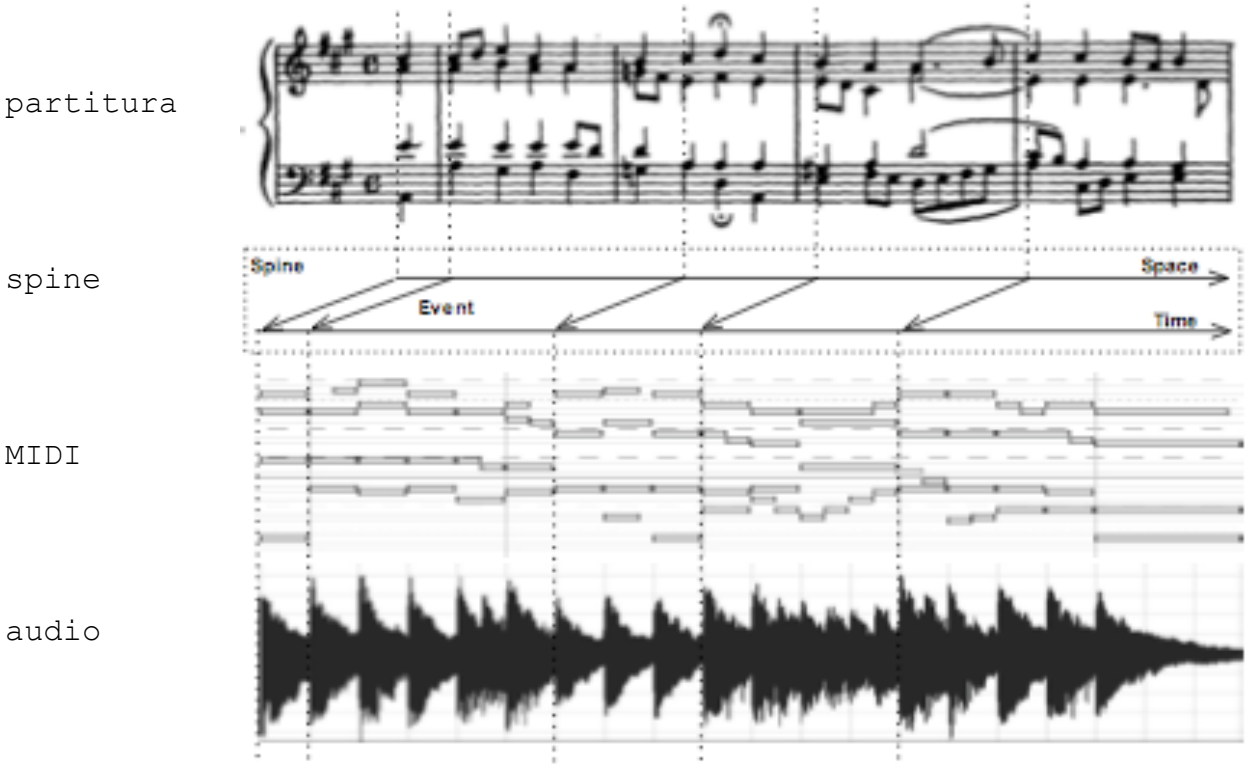

Fig. 7 - Sincronizzazione audio/MIDI/partitura in corrispondenza dello spine

Per effettuare questa sincronia fra informazioni musicali contenute in supporti digitali differenti, è necessaria una mappatura completa di ogni singolo simbolo si voglia tener conto nel completamento del file IEEE 1599. Pertanto i figli del tag xml <*spine*> saranno una lista strettamente ordinata di tutti gli eventi corrispondenti a tutti i simboli presenti nella partitura d'origine (chiavi, segnatura di tempo, alterazioni, accordi, pause, etc.. Va precisato che le note singole vengono codificate come accordo a una dimensione). Per garantire la completa indicizzazione e sincronizzazione di ogni evento su ogni livello, gli attributi contenuti in ogni elemento (*event*) saranno: un identificatore univoco (*id*), un attributo per la temporizzazione (*timing*) e un attributo per la spazializzazione grafica (*hpos*).

```
...
<spine>
    <event id="clef_1" timing="0" hpos="1" />
    <event id="clef_2" timing="0" hpos="1" />
    <event id="timesig_1" timing="0" hpos="5" />
    <event id="timesig_2" timing="0" hpos="0" />
    <event id="keysig_2" timing="0" hpos="NULL" />
    <event id="keysig_3" timing="0" hpos="NULL" />
    <event id="v1_e0" timing="0" hpos="6" />
    <event id="v2_e0" timing="0" hpos="0" />
```

Fig. 8 - Porzione di codice relativo allo spine

3.2.1 Id

L'attributo **id** assegna un identificatore univoco all'evento: ha il compito di tenere traccia di ogni simbolo presente sulla partitura all'interno dell'intero file .xml, pertanto questo attributo è contenuto anche nei nodi dei layers riferiti nello spine (all'interno dell'attributo *event_ref*). Per mantenere la validità del file non sono ammessi valori duplicati dell'attributo *id,* in quanto ogni *id* deve essere riconducibile ad un solo elemento *event*. La scelta dei valori è arbitraria ma per garantire una buona strutturazione è auspicabile una nomenclatura che permetta una decifrazione mnemonica degli eventi indicizzati nell'*id*: come nell'esempio in figura x.x, si può notare che per i simboli delle chiavi (di violino, di basso) è stato scelto il valore *clef_x,* dove *x* ne indica la numerazione in ordine di apparizione sulla partitura; discorsi analoghi possono essere fatti per gli altri eventi.

3.2.2 Timing

L'attributo **timing** costituisce una temporizzazione virtuale ed è necessario per mantenere la sincronia fra gli eventi disposti sui diversi layers. I valori dell'attributo sono espressi in **VTU** (Virtual Timing Units) i quali rappresentano una sorta di unità di misura virtuale a cui viene assegnata una porzione di tempo proporzionata alla durata dell'evento corrispondente. In altre parole, i valori sono dei numeri interi, solitamente divisibili per molti numeri (potenze di 2), che segmenteranno il dominio temporale degli eventi, in modo tale da riuscire a ricostruirne il brano nella sua interezza. L'effettiva assegnazione dei valori dell'attributo avviene considerando gli eventi simultanei verticalmente in partitura come contemporanei (come effettivamente si opera leggendo un pentagramma), quindi senza incremento di VTU (il valore rimane a zero); mentre la durata sarà espressa, sempre in VTU, come valore dell'attributo *timing* dell'evento immediatamente precedente, indicando l'intervallo di tempo virtuale trascorso fra un evento e il suo successivo.

Per ottimizzare l'assegnazione dei valori di timing, viene scelto un valore soglia di VTU da attribuire alla durata della battuta, cioè quella riportata nella segnatura di tempo relativa (nel file IEEE 1599 questo valore è contenuto nell'attributo *vtu_amount* dell'elemento *time_indication*, nel <*los*>). Una volta definito questo valore, la somma delle durate in VTU di ogni evento contenuto nella medesima battuta dovrà corrispondere al valore *vtu_amount*, altrimenti verrà falsata la temporizzazione.
Ricapitolando con un esempio pratico:

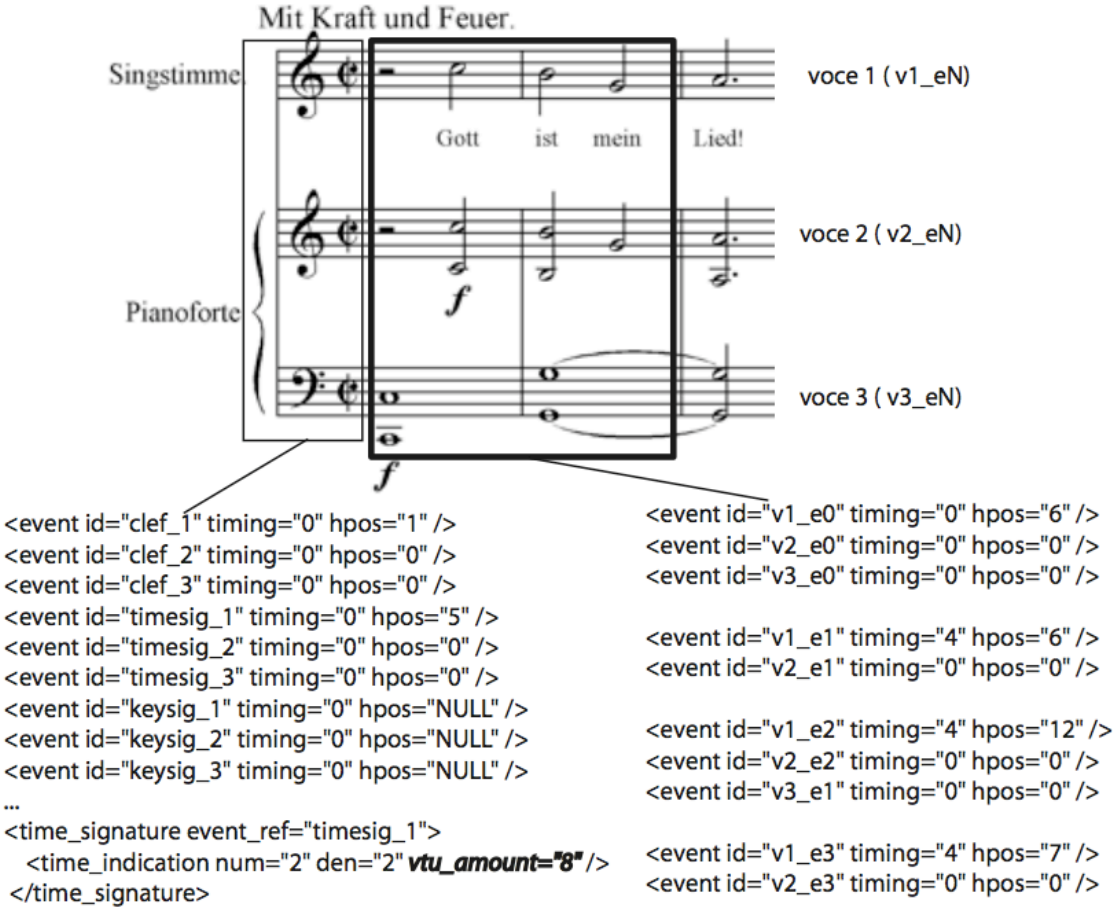

Fig. 9 - Empio di temporizzazione ottenuta tramite timing e vtu_amount

In figura si può notare che il valore dell'intera battuta è di 8 VTU e che l' evento *v1_e1* è quello immediatamente successivo, in ordine temporale, agli eventi precedenti (in questo caso contemporanei); e dal valore dell'attributo *timing="4"* si capisce che quell'evento si trova ad una durata equivalente a metà battuta rispetto al suo precedente. Il simbolo dell'evento è infatti una pausa della durata di 2/4. In modo analogo vengono trattate le durate dei simboli seguenti, presenti in partitura.

Dato il valore di *vtu_amount* è calcolabile quindi la durata in VTU di ogni figura notazionale, tenendo in considerazione il valore del *time_signature* come durata limite di ogni battuta e la possibile presenza di figure di gruppi irregolari, come le terzine.

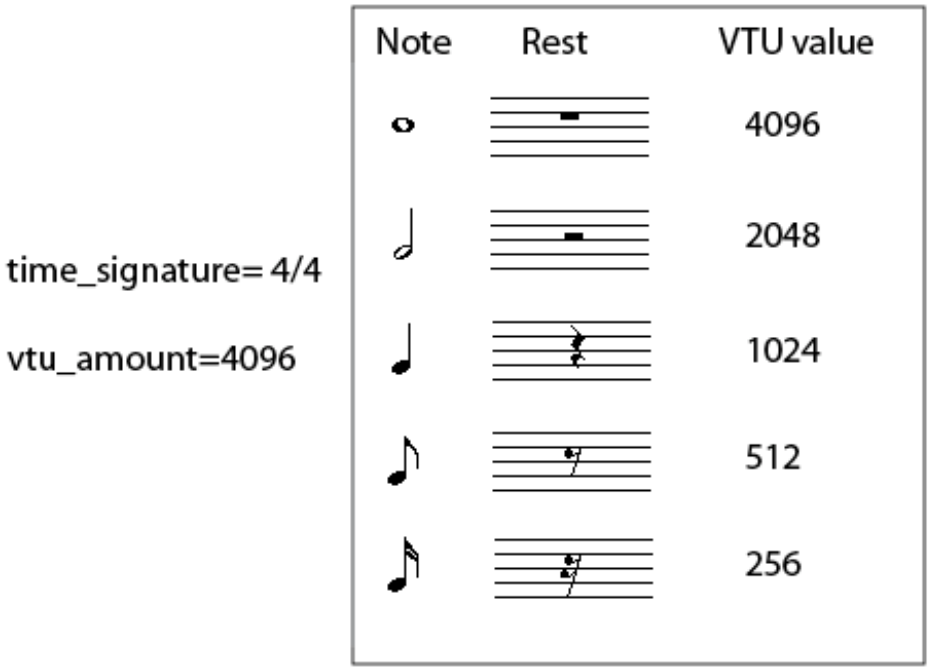

Fig. 10 - Esempio di assegnazione dei valori di VTU per i diversi simboli musicali

Nell'esempio appena illustrato si può notare come i numeri delle potenze di due si prestino ottimamente al proporzionamento dei VTU, in dipendenza del dimezzarsi della durata.
Intuitivamente si può desumere che la durata di una nota che presenta un punto di valore (in MX denominato *augmentation_dots*) espressa in VTU, risulterà la durata della nota più la sua metà. Per quanto riguarda i gruppi irregolari, invece, viene individuato il numero di simboli che devono essere eseguiti, prevalentemente scritto sopra alle legature del gruppo (in MX denominato *tuplets_ratio*), e la durata effettiva che l'evento irregolare dovrà avere. Quindi si divide la durata totale in VTU per il numero dei simboli, ottenendo il valore in VTU di ogni singolo simbolo del gruppo.
Grazie a questa modalità di assegnazione di valori all'attributo *timing*, si riesce a garantire una temporizzazione virtuale robusta e di grande precisione, facilmente correlabile alle scansioni temporali degli altri file musicali, indicizzati negli appositi layers del file IEEE 1599.

3.2.3 Hpos

L'attributo **hpos** fornisce una spazializzazione orizzontale virtuale dell'aspetto grafico della partitura. La partitura viene concepita come nella rappresentazione *ScoreView* di *Finale*, ossia scritta tutta di seguito, senza ritorni a capo. Analogamente al *timing*, i valori dell'*hpos* sono proporzionati alle durate degli eventi, definendo l'intervallo di spazio grafico che intercorre fra un evento e il suo successivo. Se l'evento nello spine non ha un riferimento grafico significativo, il valore dell'attributo è NULL.

Hpos serve per sincronizzare i diversi aspetti grafici delle rappresentazioni dell'opera: come si evince dalla figura x.x, vengono correlati visivamente i pentagrammi della partitura, la notazione del sequencer MIDI e la forma d'onda del brano audio, permettendo il confronto delle diverse rappresentazioni anche in fase di riproduzione.

3.3 Los e gerarchizzazione delle partiture

Il Logical Oranized Symbols è il sub-layer di Music Logic in cui è contenuta l'informazione simbolica dell'intera partitura, considerando separatamente ogni elemento che la costituisce (chiavi, alterazioni, agogiche, note, pause e quant'altro). Per effettuare una strutturazione delle informazioni veicolate dalla partitura, senza comprometterne la giusta esecuzione e la semantica globale, è necessario adottare una gerarchizzazione delle informazioni relative ai singoli simboli costituenti. La suddivisione utilizzata nel formato IEEE 1599 risulta:

- una partitura è formata da più accollature
- un'accollatura è formata da più pentagrammi
- un pentagramma contiene o può contenere più parti
- una parte è formata o può essere formata da più voci
- una voce viene considerata per intero, battuta per battuta
- una battuta contiene accordi e/o pause
- un accordo è formato da una o più note

Va precisato che la gerarchia *pentagramma* > *parte* > *voce* creata implicitamente dallo schema precedente, è valida per molti casi di partiture classiche, ma in letteratura non mancano delle eccezioni che violano questa relazione. Basti pensare a una partitura più complessa dove più *parti* corrispondono ad un singolo *pentagramma*; una *parte* costituita

da due o più *voci* (ovviamente sul medesimo *pentagramma*); una variazione di *pentagramma* da parte di una *voce*. In ogni caso queste anomalie sono puramente di livello grafico nella partitura originale, ma possono essere espresse ugualmente nel Los senza errori.

Gli eventi che costituiscono l'effettiva tessitura musicale, ossia *accordi* e *pause*, sono codificati all'interno di ogni singola *battuta*, e vengono descritti seguendo l'ordine di apparizione delle *parti* di riferimento. Nel <los> appariranno ordinatamente prima tutte le *battute* che costituiscono la *parte 1* (che conterranno, sempre ordinatamente, la *voce 1,* la *voce 2, etc.*), poi tutte le *battute* relative alla *parte 2*, e così via. Così operando si può notare che i nodi figli che compongono il <los> non sono ordinati secondo lo scandire della partitura (come, ad esempio, viene codificato lo spine), ma apportano al file di xml musicale una forte strutturazione fedele alla partitura d'origine.

3.4 Definizione della gerarchia in IEEE 1599

Viene presentato l'elenco degli elementi e dei relativi attributi identificati nel DTD d'origine del formato IEEE 1599 per la porzione di codice relativa alla gerarchizzazione sopra citata, contenuta all'interno dei tags <los> ... </los>:

```
...
<staff_list>
      <staff id="staff_1">
        <clef shape="G" event_ref="clef_1" staff_step="2" />
        <time_signature event_ref="timesig_1">
          <time_indication num="2" den="2" vtu_amount="8" />
        </time_signature>
        <key_signature event_ref="keysig_1">
          <flat_num number="0" />
        </key_signature>
      </staff>
      <staff id="staff_2">
        <clef shape="G" event_ref="clef_2" staff_step="2" />
        <time_signature event_ref="timesig_2">
          <time_indication num="2" den="2" vtu_amount="8" />
        </time_signature>
        <key_signature event_ref="keysig_2">
          <flat_num number="0" />
        </key_signature>
      </staff>
      ...
</staff_list>
...
```

Fig 11 - Porzione di codice xml relativa all'accollatura

<staff_list> : indica l'accollatura, ossia l'elenco dei pentagrammi che la compongono. I nodi figli sono obbligatoriamente degli <*staff*>, e deve esserne presente almeno uno per ogni <*staff_list*> per garantire la validità del DTD.

<staff> : indica il pentagramma. Possiede l'attributo *id* che ne indicizza la posizione rispetto alla relativa <*staff_list*>. Gli elementi figli sono quei simboli relativi ad ogni pentagramma: *clef, time_signature, key_signature, barline, tablature_tuning.*

```
<!ELEMENT staff (clef | ( key_signature | custom_key_signature) | time_signature | barline | tablature_tuning)*>
<!ATTLIST staff
        id ID #REQUIRED
        line_number CDATA "5"
        ossia (yes | no) "no"
        tablature (none | french | german | italian) #IMPLIED>
```

Fig. 12 - Porzione di codice relativa all'elemento staff, estratta dal DTD del formato IEEE 1599

Ogni pentagramma può contenere tali informazioni ripetute più volte all'interno del medesimo *staff* in quanto è ammissibile il cambio di chiave, segnatura di tempo e di tonalità durante l'esecuzione dello stesso. I riferimenti nello spine di tali elementi ne individueranno l'effettiva corretta posizione spazio-temporale nella partitura. Gli ultimi due elementi dell'elenco non sono sempre presenti negli esempi di file IEEE 1599, in quanto non sono fondamentali, non sono indicati come richiesti all'interno del relativo DTD.

<clef> : è l'elemento relativo alla chiave in cui dovrà essere eseguito il pentagramma. Il tag identifica un elemento vuoto, ossia che si apre e si chiude, senza contenere ulteriori elementi figli. I suoi attributi sono:

- *shape (o type)* : rappresenta il tipo di chiave (violino o sol, basso o fa, etc.)
- *staff_step* : definisce la posizione della chiave sul pentagramma
- *event_ref* : puntatore dell'evento nello spine, relativo al valore di *id* indicatovi
- *octave_num* : numero intero che indica se alzare o abbassare di n° ottave la tonalità della chiave

```
<!ELEMENT clef EMPTY>
<!ATTLIST clef
        shape (G | F | C | gregorian_F | gregorian_C | percussion | doubleG | tabguitar) #REQUIRED
        staff_step CDATA #REQUIRED
        octave_num (0 | 8 | -8 | 15 | -15) "0"
        %event_ref;>
```

Fig. 13 - Porzione di codice relativa all'elemento clef, estratta dal DTD del formato IEEE 1599

<time_signature> : nominato precedentemente, è l'elemento in cui è memorizzato il valore della segnatura di tempo, solitamente sotto forma di frazione (4/4, 3/4, 7/8, etc.), relativo allo *staff* padre in cui è contenuto. Necessità di indicizzazione, infatti il suo attributo *event_ref* ha come valore l'*id* del evento puntato nello spine.

```
<!ELEMENT time_signature (time_indication)+>
<!ATTLIST time_signature
        visible (yes | no) "yes"
        %event_ref;>
```

Fig. 14 - Porzione di codice relativa all'elemento time_signature, estratta dal DTD del formato IEEE 1599

All'interno del suo unico nodo figlio, *time_indication*, sono presenti gli attributi relativi a numeratore e denominatore della frazione e il valore della durata, determinata da una temporizzazione virtuale, espressa in VTU. I tre valori sono di tipo intero (CDATA in xml).

```
<!ELEMENT time_indication EMPTY>
<!ATTLIST time_indication
        num CDATA #REQUIRED
        den CDATA #IMPLIED
        abbreviation (yes | no) "no"
        vtu_amount CDATA #IMPLIED>
```

Fig. 15 - Porzione di codice relativa all'elemento time_indication, estratta dal DTD del formato IEEE 1599

<key_signature> : indica l'armatura di chiave del pentagramma, ossia il numero delle alterazioni in chiave, estrapolandone intrinsecamente la tonalità. Se infatti si considera una tonalità di Do maggiore (o la relativa La minore), l'elemento *key_signature* presenterà i valori a zero (quindi con la possibilità di venire anche omesso), in corrispondenza della mancanza di alterazioni in chiave per la relativa tonalità.

```
<!ELEMENT key_signature (sharp_num | flat_num) >
<!ATTLIST key_signature
        %spine_ref;>
```

Fig. 16 - Porzione di codice relativa all'elemento key_signature, estratta dal DTD del formato IEEE 1599

Dal DTD si evince che anche per questo elemento è richiesto l'attributo *event_ref*, per il puntamento allo spine. I nodi figli saranno: *sharp_num* il numero di diesis e *flat_*num il numero dei bemolle che sono presenti nell'armatura di chiave.

```
<!ELEMENT sharp_num EMPTY>
<!ATTLIST sharp_num
        number (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7) #REQUIRED>

<!ELEMENT flat_num EMPTY>
<!ATTLIST flat_num
        number (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7) #REQUIRED>
```

Fig. 17 - Porzione di codice relativa agli elementi sharp_num e flat_num, estratta dal DTD del formato IEEE 1599

Proseguendo nell'analisi del Los, ora viene chiuso il tag *</staff_list>* e inizia la lista effettiva degli eventi musicali, incapsulati nelle battute (*measure*). In questa parte di strutturazione i blocchi maggiori sono costituiti dalle parti (*part*), che a loro volta conterranno la lista delle voci che la costituiscono (*voice_list*) e quindi le singole battute disposte in ordine crescente. Infine, i nodi foglia saranno le pause (*rest*) e le singole note (notehead*)* componenti l'accordo (*chord*).

```
...
</staff>
    </staff_list>
    <part id="soprano">
      <voice_list>
        <voice_item id="soprano_voice1" staff_ref="staff_1" />
      </voice_list>
      <measure number="1">
        <voice voice_item_ref="soprano_voice1">
          <rest event_ref="v1_e0" staff_ref="staff_1">
            <duration num="1" den="2" />
          </rest>
          <chord event_ref="v1_e1">
            <duration num="1" den="2" />
            <notehead staff_ref="staff_1">
              <pitch step="C" octave="6" />
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="2">
      ...
```

Fig. 18 - Porzione di codice relativa alla chiusura del tag staff_list e all'introduzione della prima part

<part> : in questo elemento saranno presenti i simboli relativi all'effettiva parte di uno strumento, che può essere composta da una o più voci. I nodi derivati da questo elemento saranno: *voice_list* che conterrà l'elenco delle voci; e gli elementi *measure* che conterranno il contenuto delle battute. Tra gli attributi troviamo l'*id* che è sempre presente (#REQUIRED) e altri attributi con il compito di segnalare una trasposizione di pitch, di alterazioni o di ottava (non sono dati necessariamente richiesti, #IMPLIED).

```
<!ELEMENT part (voice_list, measure+)>
<!ATTLIST part
        id ID #REQUIRED
        performers_number CDATA "unknown"
   transposition_pitch (A | B | C | D | E | F | G) #IMPLIED
   transposition_accidental %accidental; #IMPLIED
        octave_offset CDATA #IMPLIED>
```

Fig. 19 - Porzione di codice relativa all'elemento part, estratta dal DTD del formato IEEE 1599

<voice_list> : è la lista delle voci (*voice_item*), ed è stato creato per garantire una forte strutturazione, in quanto il suo compito è solamente quello di contenitore.

<voice_item> : sono gli elementi che corrispondono alle voci, quindi, escludendo le eccezioni, sarà relativo ad un singolo pentagramma.

```
<!ELEMENT voice_item EMPTY>
<!ATTLIST voice_item
        id ID #REQUIRED
        staff_ref IDREF #REQUIRED
        notation_style (normal | rhythmic | slash | blank) #IMPLIED>
```

Fig. 20 - Porzione di codice relativa all'elemento voice_item.

L'attributo *id* è utilizzato come puntatore all'interno della *measure* per indicizzare gli eventi musicali in corrispondenza della voce a cui appartengono. L'attributo *staff_ref* è il puntatore al pentagramma relativo, e grazie a ciò anche l'elemento *voice_item* a un relativo riscontro nella scansione temporale definita dallo *spine*.

<measure> : in questi tags inizia effettivamente l'implementazione degli eventi musicali, all'interno della gerarchia del file IEEE 1599. Dato che gli eventi vengono descritti, parte per parte, battuta per battuta, questo elemento necessità di un attributo (*number*) che conterrà la numerazione delle battute per ogni voce presente.

```
<!ELEMENT measure (voice+ | multiple_rest | measure_repeat?)>
<!ATTLIST measure
        number CDATA #REQUIRED
        id ID #IMPLIED
        show_number (yes | no) #IMPLIED
        numbering_style (arabic_numbers | roman_numbers | small_letters | capital_letters) #IMPLIED>
```

Fig. 21 - Porzione di codice relativa all'elemento measure, estratta dal DTD del formato IEEE 1599

L'elemento figlio principale è *voice*, che possiede un putatore al relativo *voice_item*.

<voice> : elemento che identifica la voce all'interno della battuta, compito svolto dall'attributo *voice_item_ref*. I nodi figli sono il cuore dell'informazione simbolica espressa in partitura, ossia gli elementi *chord* e *rest*.

```
<!ELEMENT voice (chord | rest | tablature_symbol | gregorian_symbol)+>
<!ATTLIST voice
        voice_item_ref IDREF #REQUIRED
        ossia (yes | no) "no">
```

Fig. 22 - Porzione di codice relativa all'elemento voice, estratta dal DTD del formato IEEE 1599

Operando un filtraggio sull'elemento *voice* sarà quindi possibile ottenere l'elenco di tutti gli eventi musicali ordinati secondo la loro apparizione nelle battute.

<chord> : è l'elemento composto dalle note (almeno una nota deve essere obbligatoriamente implementata) e ne contiene la durata (*duration*), la presenza di punti di valore (*augmentation_dots*) e le articolazioni (*articulation*).

```
<!ELEMENT chord (duration, augmentation_dots?, (notehead+ | repetition), articulation?)>
<!ATTLIST chord
        id ID #IMPLIED
        %event_ref;
        stem_direction (up | down | none) #IMPLIED
        beam_before (yes | no) "no"
        beam_after (yes | no) "no"
        cue (yes | no) "no"
        tremolo_lines (no | 1 | 2 | 3 | 4 | 5 | 6) #IMPLIED>
```

Fig. 23 - Porzione di codice relativa all'elemento chord, estratta dal DTD del formato IEEE 1599

Possiede diversi attributi che gestiscono la semantica e la rappresentazione grafica dell'accordo. L'unico attributo fondamentale è l' *event_ref* che indicizza l'evento nello spine, e ne garantisce la corretta temporizzazione virtuale effettuando la corrispondenza con l'attributo *timing*.

<rest> : è un elemento di pari livello di *chord*, e indica le pause presenti nel pentagramma. Anch'esso presenta gli attributi *duration* e *augmentation_dots* che ne descrivono la durata, che saranno correlati a loro volta con il *timing* dell'evento puntato nello spine attraverso l'attributo *event_ref*.

```
<!ELEMENT rest (duration, augmentation_dots?)>
<!ATTLIST rest
        id CDATA #IMPLIED
        %event_ref;
        staff_ref IDREF #IMPLIED
        hidden (no | yes) #IMPLIED>
```

Fig. 24 - Porzione di codice relativa all'elemento rest, estratta dal DTD del formato IEEE 1599

<duration> : è la durata dell'evento (sia *chord* che *rest*) espressa come frazione di numeri interi. Può avere il nodo figlio *tuplet_ratio* precedentemente citato, se la durata è relativa a un gruppo irregolare, in quanto l'insieme delle durate espresse in frazione non sono più l'effettivo intervallo temporale in fase di esecuzione.

```
<!ELEMENT duration (tuplet_ratio?)>
<!ATTLIST duration
        num CDATA #REQUIRED
        den CDATA #REQUIRED>
```

Fig. 25 - Porzione di codice relativa all'elemento duration, estratta dal DTD del formato IEEE 1599

<notehead> : è un elemento figlio di *chord* ed è annidiato all'interno di *duration* perchè tutti gli elementi componenti un accordo possiedono la medesima durata. *Notehead* contiene tutti quei valori propriamente della nota come l'altezza (*pitch*), la presenza di alterazioni, la diteggiatura da seguire (*fingering*).

```
<!ELEMENT notehead (pitch, printed_accidentals?, tie?, fingering?)>
<!ATTLIST notehead
  id ID #IMPLIED
  staff_ref IDREF #IMPLIED
  style (normal | harmonic | unpitched | cymbal | parenthesis | circled | squared) #IMPLIED>
```

Fig. 26 - Porzione di codice relativa all'elemento notehead, estratta dal DTD del formato IEEE 1599

<pitch> : rappresenta l'ultimo nodo per la strutturazione di ogni evento *chord* e possiede gli attributi che identificano completamente una nota. L'attributo *step* ne indica la altezza nell'ottava (espressa con la notazione angolosassone di pitch), l'attributo *octave* ne indica l'ottava relativa (espresso come numero intero) e l'attributo *actual_accidentals* che indica se la nota presenta alterazioni (i valori sono di tipo *accidental* e possono essere *sharp, flat, double_sharp, double_flat, natural).*

```
<!ELEMENT pitch EMPTY>
<!ATTLIST pitch
        step (A | B | C | D | E | F | G | none) #REQUIRED
        octave CDATA #REQUIRED
        actual_accidental %accidental; #IMPLIED>
```

Fig. 27 - Porzione di codice relativa all'elemento pitch, estratta dal DTD del formato IEEE 1599

Possiamo desumere che questa gerarchia così ben strutturata sia una fedele digitalizzazione dell'informazione veicolata da una partitura cartacea, capace di gestire diverse tipologie di partiture: da quelle orchestrali molto complesse a quelle più elementari, come può essere uno spartito per pianoforte e voce.
Ogni evento è descritto nell'interezza delle sue caratteristiche semantiche, e l'indentazione creata dal file .xml ne favorisce un utilizzo versatile in fase di programmazione.
Nel capitolo seguente verrà presentata l'analisi del software Chroma Vector e l'uso effettivo dello *spine* e di questi elementi presenti nel *<los>,* per il calcolo del grafico chromagram.

CAP. 4 Chroma Vector

4.1 Introduzione al software

Il software per l'estrazione di chroma vector che verrà presentato è integrato all'interno del Framework-MX contenente diversi plug-in da applicare ai file in formato .xml IEEE 1599, sviluppato nel laboratorio del LIM (Laboratorio di Informatica Musicale), nel dipartimento di Informatica. Il codice è scritto in linguaggio C#, e come piattaforma di sviluppo si è scelto di utilizzare Visual Studio (sistema operativo Windows XP). Data l' attuale presenza di molti plug-in già implementati in linguaggio C#, la scelta di utilizzare questo ambiente di sviluppo non è stata cambiata in fase di progettazione, per mantenere omogeneità fra i diversi software (Easy Midi, Event Inspector, Audio Mapper, ...) che permettono di gestire le caratteristiche dei file IEEE 1599 in un unico software di lavoro, Framework-MX.

4.2 Analisi ad alto livello: comportamento del sistema

L'implementazione del plug-in che verrà illustrata ha come obiettivo la determinazione automatica, attraverso il calcolo del grafico, della caratteristica di *chroma*, utilizzando come partitura d'origine un file in formato IEEE 1599. Il grafico viene generato a runtime dopo aver premuto il bottone Start!, con la possibilità di prendere in esame l'intero brano o solo parti delimitate da un range di battute.

INPUT **OUTPUT**

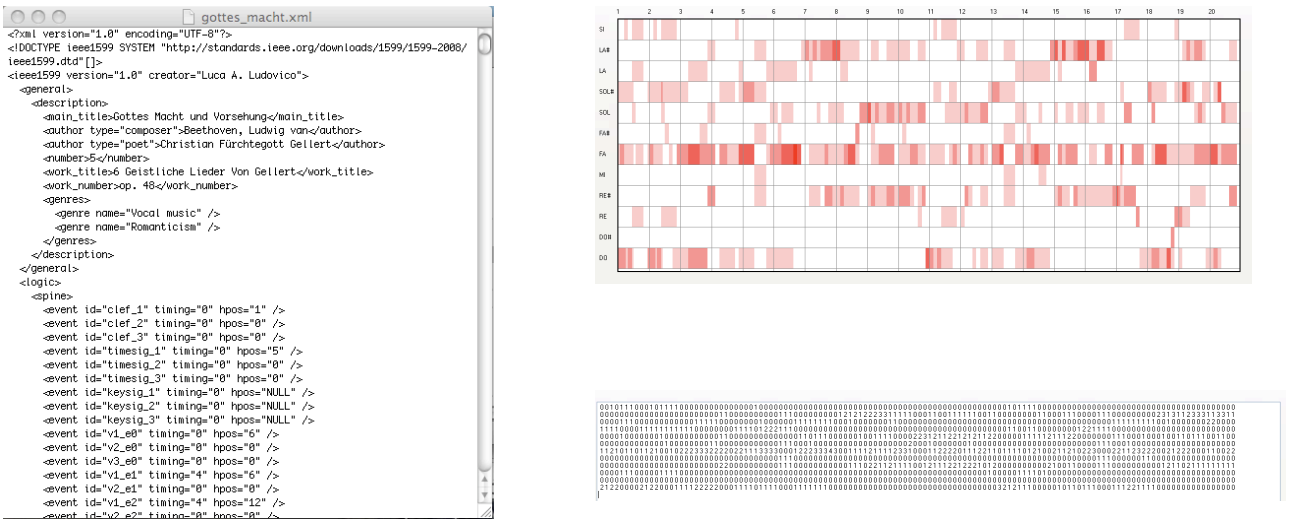

Fig. 28 - Input e output del software Chroma Vector

In input il software richiederà un file conforme al DTD del relativo formato IEEE 1599. Il Framework-MX permette di aprire un file .xml, dando la possibilità di selezionare se operare o meno la validazione rispetto al DTD originario, verificando l'attendibilità del file.
In output Chroma Vector offre la possibilità di visualizzare due schermate: la prima (quella selezionata di default) è definita Chroma View e indica il grafico che evince la caratteristica energetica su diversi livelli, la chroma appunto; la seconda, invece, è una matrice di numeri interi avente 12 righe corrispondenti ai livelli di altezza delle note e *x* colonne in dipendenza dei valori temporali degli eventi contenuti nelle battute selezionate. Questa visualizzazione a griglia permette di incapsulare in un unica variabile, facilmente gestibile, gli indici che generano la colorazione (quindi la densità di nota) di ogni singolo vettore. Si è scelto di aggiungere questa visualizzazione con la possibilità di effettuare copia/incolla, per degli ulteriori sviluppi con software di calcolo o sintesi quali, ad esempio, MatLab.

Il *modus operandi* del programma può essere sintetizzato sommariamente in questi punti:

1. Apertura del file .xml in formato IEEE 1599 e del plug-in Chroma Vector

2. Scelta del range di battute, dei colori da attribuire ai vettori e del bpm per l'approssimazione temporale in minuti e secondi.

3. Click sul bottone Start!, esecuzione dell'algoritmo per l'estrazione della chroma

4. L'algoritmo si preoccupa di:
    (a) Decifrare il tempo in chiave e il suo relativo valore in VTU, contenuti nell'elemento *Time_Signature.*
    (b) Analisi delle durate di tutti gli eventi compresi nel range di battute. La durata che corrisponderà all'evento più breve diventerà l'unità di misura grafica da attribuire al valore espresso in VTU.
    (c) Analisi delle altezze e delle alterazioni delle note. E' necessario che le note enarmoniche (ossia di ugual valore semantico, ma con diversa denominazione) vengano considerate all'interno dei 12 valori di discretizzazione (divisione di semitoni crescente, quindi con l'aggiunta di diesis).

5. Compilazione della matrice di interi GridVectors contenente i contatori incrementali che identificano la presenza sincrona di note con la medesima altezza.

6. Implementazione dei rettangoli colorati costituenti i vettori. La scala della colorazione è proporzionata al numero di note contemporanee, ed è illustrata graficamente nella sezione *Legend.*

7. Di default è selezionata la visualizzazione del ChromaGram, altrimenti può essere selezionata l'interfaccia a Grid, dove una *TextBox* conterrà la matrice di interi GridVectorsGridVectors.

Nel prossimo paragrafo verrà presentata una descrizione dettagliata di questi step, con le implementazioni di codice C# relative.

4.3 Analisi a basso livello: implementazione del codice

Tra le librerie di funzioni da includere al progetto *ChromaVector.cs* oltre a quelle generiche contenenti le direttive agli oggetti di frequente utilizzo in eseguibili codificati in C#,vi sono la *System.Xml* che fornisce un supporto di elaborazione per interrogazioni a file .xml, mentre per la creazione del file IEEE 1599 su cui utilizzare le suddette interrogazioni, è inclusa la libreria proprietaria del framework: *IEEE1599StandardLibrary*. Per l'integrazione del plug-in all'interno dell'eseguibile del Framework-MX è necessario includere la libreria *IEEE1599Framework* che ne permette la gestione del file e il posizionamento nel menù generale. Per la creazione dei vettori grafici, illustrati nel chromagram da dei rettangoli, è necessaria l'importazione della *System.Drawing* e *System.Drawing.Drawing2D* per la creazione della griglia fra le linee delimitanti note e battute.

Il controllo utilizzato per contenere la creazione del grafico è un *pictureBox*, utilizzato peraltro come controllo per la legenda della scala cromatica. Per la modifica dei colori dei vettori e del background vi sono i controlli *ColorDialog*, che permettono di gestire il cambio di valore di una variabile *Color* aprendo una form di Windows per la modifica dei colori. Gli ulteriori controlli utilizzati sono i classici, utilizzati nella quasi totalità delle applicazioni odierne.

Viene presentata l'analisi dell'implementazione così come la progettazione logica ha portato alla stesura dell'intero codice, ricalcando la procedura svolta dall'algoritmo per lo sviluppo del chromagram.

4.3.1 Analisi delle battute e delimitazione del range

In fase di inizializzazione del plug-in viene generato un *ArrayList* che conterrà l'elenco completo del numero di battute componenti l'intero brano. Questo elenco verrà stampato in due *comboBox* sulle quali verrà verificata l'attendibilità del range selezionato in fase di esecuzione tramite click sul bottone Start!.

```
...
  XmlNodeList xList = document.Xml.SelectNodes("/ieee1599/logic/los/part/measure");
  ArrayList Measure = new ArrayList();

            foreach (XmlNode xN in xList)
               if(xN.Attributes["number"].Value!=null)
                  Measure.Add(Int32.Parse(xN.Attributes["number"].Value));

            Measure.Sort();
...
```

Fig. 29 - Analisi attributo measure

Tutte le eventuali operazioni da effettuare saranno relative solamente agli eventi inclusi nell'intervallo selezionato, quindi la richiesta *Xml.SelectNodes* andrà a operare controllando il numero di battuta corrente, che corrisponde all'indice del ciclo.

```
...
  StartMeasure = Int32.Parse(comboBoxMeasureStart.SelectedItem.ToString());
  EndMeasure = Int32.Parse(comboBoxMeasureEnd.SelectedItem.ToString());

  if (StartMeasure > EndMeasure)
        MessageBox.Show("Error, select right measure!");

  for (int countMeasure = StartMeasure; countMeasure <= EndMeasure; countMeasure++)
        {
         XmlNodeList xList = xmlDocument.Xml.SelectNodes("/ieee1599/logic/los/part/ measure[@number='"
+ countMeasure.ToString() + "']/voice/*");

            foreach (XmlNode xN in xList)
            {
...
```

Fig. 30 - Interrogazione xml per il range di battute

4.3.2 Time_signature e vtu_amount

In seguito il plug-in svolge l'analisi degli attributi *Time_indication* e *vtu_amount* per decretare il valore della segnatura di tempo e della relativa durata in termini di VTU. Nel caso in cui questo valore varia tra una battuta e la sua successiva (musicalmente si ha un cambio di tempo) appare una *MessageBox* che avverte che il range di visualizzazione sarà forzato al valore dell'ultima battuta con la medesima segnatura di tempo iniziale. Questo caso particolare viene gestito in questo modo per un'implementazione meno complessa; ciò non toglie il suo possibile perfezionamento in uno sviluppo futuro.

```
...
  foreach (XmlNode xNt in xTimeSignature)
          {
            Time.num = Int32.Parse(xNt.Attributes["num"].Value.ToString());
            Time.den = Int32.Parse(xNt.Attributes["den"].Value.ToString());

             ...

             if ((TimeChange.num!=0)&&((TimeChange.num != Time.num) || (TimeChange.den != Time.den)))
             {
               EndMeasure = countMeasure - 1;
               comboBoxMeasureEnd.SelectedItem = EndMeasure;
               MessageBox.Show("The Time Signature was changed at measure" + EndMeasure.ToString());
              }
...
```

Fig. 31 - Controllo sul cambio di tempo.

Dagli esempi di file in formato IEEE 1599 testati con questo software è stata verificata la possibile assenza dell'attributo *vtu_amount*, pertanto è stato necessario_introdurre un *InputBox* dalla quale è possibile inserire questo valore manualmente. Data la mancanza di questo controllo, o di controlli equivalenti, di default è stato aggiunto al progetto il riferimento *Microsoft.VisualBasic* che ne permette l'integrazione in C#, evitando la creazione di una form ad hoc per l'inserimento dati.

```
...
  if (xNt.Attributes.Count.Equals(3))
        VTU = Int32.Parse(xNt.Attributes["vtu_amount"].Value.ToString());

  else if (VTU==0)
        {
            String Prompt = "Insert right value (default value is 4096):";
            String Title = "Attribute vtu_amount don't found";
            String Default = "4096";
            Int32 XPos = ((SystemInformation.WorkingArea.Width / 2) - 200);
            Int32 YPos = ((SystemInformation.WorkingArea.Height / 2) - 100);

           String Result = Microsoft.VisualBasic.Interaction.InputBox(Prompt, Title, Default, XPos, YPos);

           VTU = Int32.Parse(Result);
        }
...
```

Fig. 32 - InputBox per inserimento manuale di vtu_amount

4.3.3 Suddivisione virtuale del tempo

Una volta assegnato il valore della temporizzazione virtuale di ogni singola battuta, è stato pensato di attribuire un'unità di misura dinamica per la visualizzazione grafica della lunghezza dei vettori. Questo valore è equivalente alla durata minima, espressa in VTU, dell'evento più breve all'interno del range selezionato. Ricapitolando quanto visto nel paragrafo 3.2.2 relativo all'attributo *timing*, il valore di VTU espresso al suo interno è relativo alla durata dell'evento immediatamente precedente nella disposizione in partitura, perciò è necessario calcolare il valore effettivo della durata in VTU dell'evento, per poi operare una ricerca sul minore.

In definitiva il valore *vtu_amount* della battuta viene diviso per l'intero *VTUminSlice* (l'unità di misura di suddivisione) restituendo il numero di slice componenti una battuta. Questo numero moltiplicato per il numero totale di battute considerate equivarrà alla variabile *nSlice*, ossia il numero totale di divisioni di tempo utilizzando il *VTUminSlice* corrente.

```
...
  XmlNodeList xChord = xmlDocument.Xml.SelectNodes("/ieee1599/logic/los/part/measure/voice/
chord[@event_ref='" + Event_ref + "']/duration");
        foreach (XmlNode xC in xChord)
                {
                   float o = 0; //augmentation_dots in decimale
                   int num = Int32.Parse(xC.Attributes["num"].Value.ToString());
                   int den = Int32.Parse(xC.Attributes["den"].Value.ToString());
                   float dur = ((float)num / (float)den);

                   if (xC.ParentNode.ChildNodes.Item(1).Name.Equals("augmentation_dots"))
                   {
                      float aux = dur;
                      int Point_ref = Int32.Parse(xC.ParentNode.ChildNodes.Item(1).Attributes
                      ["number"].Value.ToString());

                      for (int w = 0; w < Point_ref; w++)
                      {
                         aux = aux / 2;
                         den = den * 2;
                         o += aux;
                      }
                   }

                   if (xC.ChildNodes.Count != 0)
                   {
                     terzinaCount++;
                   }
                   else terzinaCount = 0;

                   dur = (float)num /(float) den;

                   xVTU = ((dur+o)/((float)minSlice.num / (float)minSlice.den)) * VTUminSlice;
...
```

Fig. 33 - Calcolo della durata di eventi di tipo Chord

Come si può evincere dal codice in figura, per il calcolo della durata in VTU viene applicata una proporzione fra la durata espressa in frazione (trasformata nel *float* equivalente) e il valore dell'unità di misura *VTUminSlice*. Un'operazione analoga viene svolta per gli eventi di tipo *Rest*.

4.3.4 Localizzazione delle altezze e degli enarmonici

Sono stati ottenuti i valori costituenti l'asse temporale del grafico, ora è necessario implementare il posizionamento dei vettori all'interno dei 12 livelli relativi alle altezze. Si ricorda che la codifica delle note, in IEEE 1599, è svolta considerando elementi di tipo *chord* anche quando è presente una nota singola. Pertanto è stato creata una variabile di tipo *ArrayList* a cui verrà aggiunto, di volta in volta, il valore dell'attributo *step* relativo all'elemento *pitch*. E' stato usato un contenitore di dimensione variabile, in quanto non è previsto un attributo che indichi di quante *notehead* è composto l'accordo.

Ad *Accordo* non viene aggiunta la stringa corrispondente al nome della nota, bensì viene operata una numerazione ordinata dei 12 livelli di altezza, partendo dall'alto: il livello corrispondente al SI è il valore 1, quello relativo al DO è il 12. Il valore 0 è assegnato alla variabile *Accordo* nel caso in cui l'evento considerato è una pausa.

```
...
                    switch (nh.Step)
                    {
                        case 'A':
                            {
                                switch (nh.RealAccidentals)
                                {
                                    case AccidentalType.natural:
                                        {
                                            Accordo.Add(3);
                                        } break;
                                    case AccidentalType.none:
                                        {
                                            Accordo.Add(3);
                                        } break;
                                    case AccidentalType.double_flat:
                                        {
                                            Accordo.Add(5);
                                        } break;
                                    case AccidentalType.double_sharp:
                                        {
                                            Accordo.Add(1);
                                        } break;
                                    case AccidentalType.flat:
                                        {
                                            Accordo.Add(4);
                                        } break;
                                    case AccidentalType.sharp:
                                        {
                                            Accordo.Add(2);
                                        } break;
                                    default: {
                                                } break;
                                }

                            } break;
                        case 'B':
                            {
...
```

Fig. 34 - Compilazione della variabile Accordo con enarmonici della nota LA

In musica le altezze delle note possono presentare diverse denominazioni, pur essendo relative al medesimo valore semantico: queste note vengono definite enarmonici. Nell'algoritmo è necessario prevedere una traduzione delle altezze nell'ordine illustrato nel plug-in, ossia con incrementi di semitono, in modo da far corrispondere adeguatamente tutte le note al relativo livello di altezza nel chromagram.

4.3.5 Compilazione variabile GridVectors

L'algoritmo prevede la creazione di una matrice di interi, enunciata dalla variabile GridVectors, che ha come scopo la mappatura degli indici che verranno letti per l'implementazione grafica del chromagram. Il dimensionamento di questa matrice dovrà essere dinamico per ogni volta che verrà premuto il bottone di esecuzione: il numero di righe, ossia quello dei livelli di altezza, varrà evidentemente 12; mentre il numero di colonne, ossia il numero totale delle divisioni relative all'unità di misura *VTUminSlice*, è calcolato a runtime come spiegato nel paragrafo sulla suddivisione temporale (4.3.3), ed è contenuto nella variabile *nSlice*. La matrice sarà inizializzata con tutti 0, che verranno incrementati indicando il numero di note sincrone con la medesima energia.

```
...
  nSlice = (int)(VTU / VTUminSlice) * (EndMeasure - StartMeasure + 1);
  GridVectors = new int[nSlice, 12];

  for (int i = 0; i < nSlice; i++)
      for (int j = 0; j < 12; j++)
          GridVectors[i, j] = 0;
...
```

Fig. 35 - Inizializzazione GridVectors

La compilazione di questa matrice avviene effettuando un ciclo sulla richiesta xml che interroga la porzione di codice contenuta nei tags *<spine>*, selezionando solo gli eventi compresi nell'intervallo di battute.

Per ogni evento bisogna effettuare un controllo sull'attributo *timing,* che possiede valore 0 se l'evento è contemporaneo al precedente, oppure un valore diverso da 0 che corrisponde alla distanza in VTU dall'evento appena trascorso. Questo controllo permette la localizzazione, nella matrice, del punto di partenza da cui verrà espletata la durata dell'evento, sottoforma di multiplo dell'unità di misura. Per garantire ciò, sono stati creati due indici: *xGriglia* che ha il compito di tenere traccia del cambio di *timing*, posizionando il puntatore al corretto valore della matrice relativo all'inizio dell'evento; *xCount*, invece, viene incrementato proporzionalmente all'unità di misura, fino al numero di slice che completano la durata dell'evento.

Il posizionamento nella matrice al corretto livello di altezza, viene gestito dal valore di ogni intero contenuto nella variabile *Accordo*, implementata come visto nel paragrafo precedente.

```
...
  for (int t = (int)VTUminSlice; t <= (int)xVTU; t += (int)VTUminSlice)
     {
        if (timing == 0)
           {
             terzina = 0;
              if (obj != 0)
                       {
                            if ((Xgriglia+Xcount) < nSlice)
                              {
                                    GridVectors[Xgriglia+Xcount, obj-1] += 1;
                              }
                       }
           }
        else
           {
             if (timing < VTUminSlice)
                       {
                             if (terzina==0)
                                   {
                                      first = (int)((timing*terzinaCount+1)/VTUminSlice);
                                      Xgriglia += first;
                                      terzina++;
                                   }
                       }
              else if (first == 0)
                       {
                             first = (int)(timing / VTUminSlice);
                             Xgriglia += first;
                        }

              if (obj != 0)
                        {
                              if ((Xgriglia + Xcount) < nSlice)
                                {
                                   GridVectors[Xgriglia+Xcount, obj-1] += 1;
                                }

                        }
           }
   Xcount++;
   }

...
```

Fig. 36 - Assegnazione valori incrementali a GridVectors

Una volta eseguito il posizionamento dell'evento all'interno della variabilie GridVectors, viene utilizzato un contatore incrementale per salvare nella matrice la presenza sincrona della medesima altezza nel medesimo istante di tempo.
La matrice completata avrà una dimensione *nSlice* x 12, e i propri elementi saranno degli interi che indicheranno la caratteristica di chroma relativa a ogni altezza.

4.3.6 Implementazione grafica del chromagram

Dalla lettura sequenziale della GridVectors verranno implementati i vettori costituenti il chromagram. Tali vettori graficamente vengono definiti tramite dei *Rectangle*, oggetti grafici della libreria *System.Drawing*, i quali verranno colorati proporzionalmente all'energia relativa di ogni livello di altezza.

Si otterrà una scalatura di colore tra 0, che equivale all'assenza di nota, e il numero massimo di note sincrone di ugual altezza. Questo numero è il valore massimo contenuto nella matrice di interi.

Graficamente la scalatura cromatica è implementata sfruttando la componte alfa di trasparenza del colore, in C# editabile con la funzione *Color.FromArgb()*.

```
...
  for (int i = 0; i < 12; i++)
      {
      for (int j = 0; j < nSlice; j++)
          {
            if (GridVectors.Length != 0)
                {
                   aux = j + 1;

                   //elimina pixel bianchi
                   VTUrett.Width = (int)(larghezza*aux)-(int)(larghezza*j);
                   VTUrett.X = (int)(larghezza * j) + (int)contorno.X+ 1;

                    Color myColor1;

                    myColor1 = Color.FromArgb(GridVectors[j, i] * (int)(255 / value), myColor);
                    Brush myBrush = new SolidBrush(myColor1);

                    g.FillRectangle(myBrush, VTUrett.X, VTUrett.Y, VTUrett.Width, VTUrett.Height);

 ...
```

Fig. 37 - Implementazione grafica dei vettori.

Le dimensioni dei rettangoli, che corrispondono ai vettori, sono dinamiche in dipendenza della grandezza dell'intervallo di battute: il valore dell'altezza è identico alle divisioni relative ai livelli delle note, mentre il valore della lunghezza è proporzionale allo spazio orizzontale utilizzabile in *pictureBox*. Un ulteriore sviluppo potrebbe essere quello di integrare una *ScrollBar* orizzontale in modo da permettere all'utente di selezionare dinamicamente la lunghezza dei vettori.

Per migliorare la visualizzazione del chromagram, si è scelto di tracciare delle linee verticali in corrispondenza dell'inizio di ogni battuta, e delle linee orizzontali per suddividere i livelli di altezza.

4.3.7 Localizzazione con mouse

Una feature integrata in questo software permette di localizzare tramite puntamento del mouse, la posizione temporale di un determinato vettore del chromagram. Il valore del tempo indicato nella *label* è approssimato, in quanto è necessario inserire manualmente il corretto numero di bpm (battiti per minuto) del brano caricato nel Framework-MX.

```
...
  secTot = (float)((float)60 / (int)(bpm * ((float)0.25 / ((float)minSlice.num / (float)minSlice.den))));

   Xpic=(float)((e.X /((contorno.Width-1)/nSlice)) * secTot);
   msec = (int)(((float)Xpic - (int)Xpic)*1000);
   min = (int)((int)Xpic / 60);

    if(min!=0)
        sec = (int)(((float)Xpic - (int)Xpic) + ((int)((int)Xpic % 60)));
    else
        sec = (int)(Xpic - (((float)Xpic - (int)Xpic)));

    label9.Text = " "+min.ToString()+" : "+sec.ToString()+" : "+msec.ToString();
...
```

Fig. 38 - Divisione dell'approssimazione dei secondi

Per facilitarne la lettura è stata implementata anche una *label* in cui verrà scritto il nome della nota che si sta puntando tramite mouse.

```
...
  if ((e.Y >= Ypic) && (e.Y < (Ypic + altezzaRighe)))
            label5.Text = ""+ nomiNote[0];
  else if ((e.Y >= (Ypic + altezzaRighe)) && (e.Y < (Ypic + (altezzaRighe * 2))))
            label5.Text = "" + nomiNote[1];
  else if ((e.Y >= (Ypic + (altezzaRighe * 2))) && (e.Y < (Ypic + (altezzaRighe * 3))))
            label5.Text = "" + nomiNote[2];
  else if ((e.Y >= (Ypic + (altezzaRighe * 3))) && (e.Y < (Ypic + (altezzaRighe * 4))))
            label5.Text = "" + nomiNote[3];
  else if ((e.Y >= (Ypic + (altezzaRighe * 4))) && (e.Y < (Ypic + (altezzaRighe * 5))))
            label5.Text = "" + nomiNote[4];

  ...

  else if ((e.Y >= (Ypic + (altezzaRighe * 10))) && (e.Y < (Ypic + (altezzaRighe * 11))))
            label5.Text = "" + nomiNote[10];
  else if ((e.Y >= (Ypic + (altezzaRighe * 11))) && (e.Y < (Ypic + (altezzaRighe * 12))))
            label5.Text = "" + nomiNote[11];
  else
        {
            label5.Text = "";
            label9.Text = "";
        }
...
```

Fig. 39 - Compilazione Label relativa alle note

4.4 Interfaccia utente

In questo paragrafo verrà trattata la spiegazione dell'intefaccia grafica del software Chroma Vector. Nella parte superiore della schermata è situato il menù contenente le varie opzioni di visualizzazione.

Menu:

- *Select Measure*

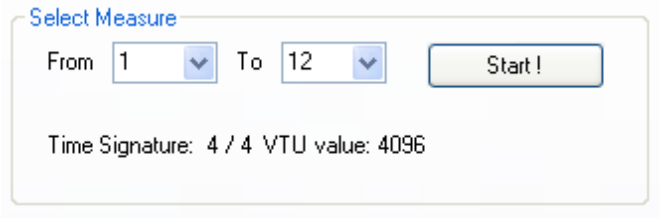

Fig. 40 - GroupBox relativo alla selezione delle battute

In questo *groupBox* è possibile selezionare il numero della battuta di inizio e quello della battuta finale del brano caricato nel framework. La *label* sottostante, una volta in esecuzione, viene completata indicando il *Time_Signature* e il relativo valore di VTU per battuta. Il bottone Start! è il controllo per mandare in esecuzione il calcolo del grafico.

- *Graphic Tool*

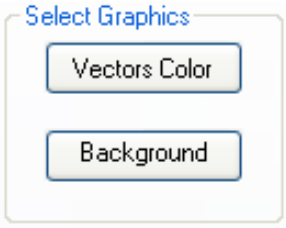

Fig. 41 - GroupBox relativo al cambio dei colori

In questo *groupBox* sono presenti due bottoni. Il primo permette di selezionare il colore dei vettori, mentre il secondo il colore di contrasto del background. Per la scelta dei colori è stato utilizzato l'oggetto *ColorDialog*.

- *Select View*

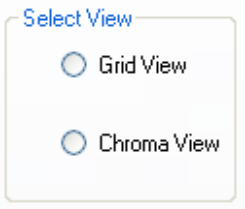

Fig. 42 - GroupBox relativo alla selezione della visualizzazione

In questo *groupBox* sono presenti solamente due *radioButton* che permettono di alternare le due visualizzazioni della chroma possibili: quella Grid, ossia quella dove gli eventi sono espressi in una *textBox* sotto forma di una matrice di interi; quella Chroma, che ritorna alla visualizzazione di default, ossia quella del chromagram.

- *Audio Tool*

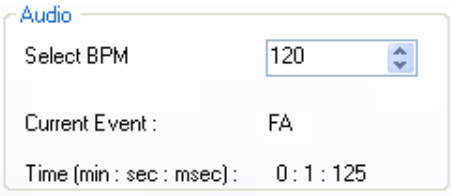

Fig. 43 - GroupBox relativo alla selezione per la features Audio

In questo *groupBox* è presente un selettore di BPM, scalato da 40 a 200 valori interi, che viene utilizzato per un calcolo approssimativo della durata del brano. In questo parte del menù sono presenti anche due *label* che si aggiorneranno in automatico con il metodo *MouseMove* dell'oggetto *picturBoxChroma* stampando la nota dell'evento corrente e il relativo valore temporale espresso in minuti : secondi : millisecondi , calcolati grazie al valore della segnatura di tempo e i BPM selezionati.

- *Color Legend*

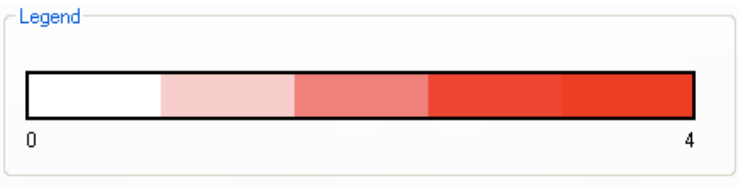

Fig. 44 - GroupBox relativo alla selezione per la features Audio

In questa parte del menù è illustrata la legenda dei colori a cui il chromagram fa riferimento, indicando con zero l'assenza di nota (colore di background) e con il numero massimo di eventi contemporanei la fine della scalatura del colore dei vettori, diviso proporzionalmente agendo sulla componente alfa di trasparenza del colore.

Nella parte inferiore della schermata sono situate alternativamente le due modalità di visualizzazione:

- *Chroma View*

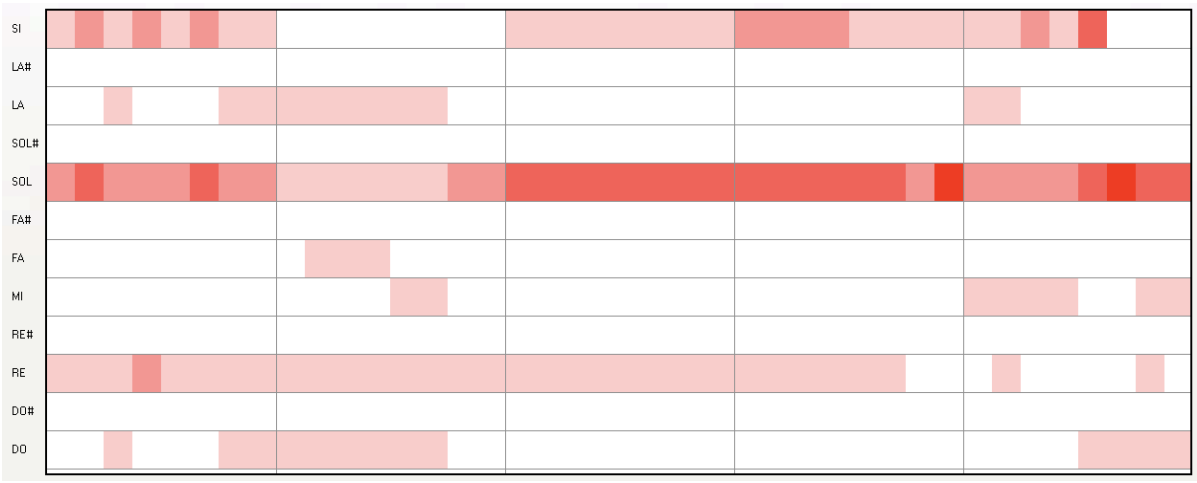

Fig. 45 - Esempio di Chroma View

E' implementata dalla *pictureBoxChroma* ed è dove sarà visualizzato l'effettivo output del software. All'interno vi sono illustrati i vettori, graficamente sottoforma di rettangoli, diversamente colorati in corrispondenza della quantità di medesime note sincrone eseguite nella partitura d'origine.

- Grid View

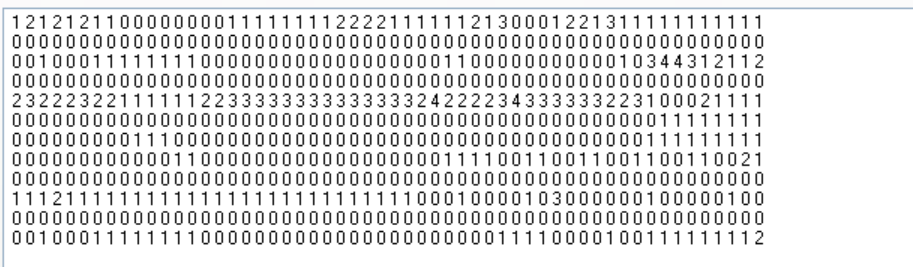

Fig. 46 - Esempio di Grid View

La visualizzazione della griglia è stata sviluppata in una *TextBox* automaticamente ridimensionata, che conterrà una matrice di 12 x N, dove 12 saranno le righe corrispondenti alle altezze delle note, e N sarà il numero di slice di durata minima (unità di misura corrispondente ai VTU) per le colonne della matrice.

CAP. 5 Conclusioni

5.1 Testing e valutazioni finali

Per conferire completezza alla ricerca, vengono presentati una serie di test effettuati utilizzando il plug-in Chroma Vector, con diversi file .xml codificati in formato IEEE 1599. Come precisato in precedenza, per un ottimale visualizzazione del chromagram è auspicabile un intervallo di battute tale da permettere di distinguere le colorazioni dei vari eventi. Il numero massimo di battute visualizzabili varia in dipendenza dell'informazione musicale effettivamente presente in partitura: se il brano considerato è un lied per piano e voce i pentagrammi presenti saranno al massimo tre, quindi difficilmente si verificheranno degli errori grafici e si potrà selezionare il brano nella sua interezza; d'altro canto una partitura per orchestra sinfonica necessita, in fase di selezione dell'intervallo, di operare per tentativi. Un altro fattore che condiziona la qualità del chromagram è la durata minima (utilizzata come unità di misura grafica) di un evento con valore 1/32 o inferiore: questo porterebbe alla creazione di un numero molto elevato di slice componenti i vettori, e, su un ampio intervallo di battute, potrebbe causare incomprensioni.

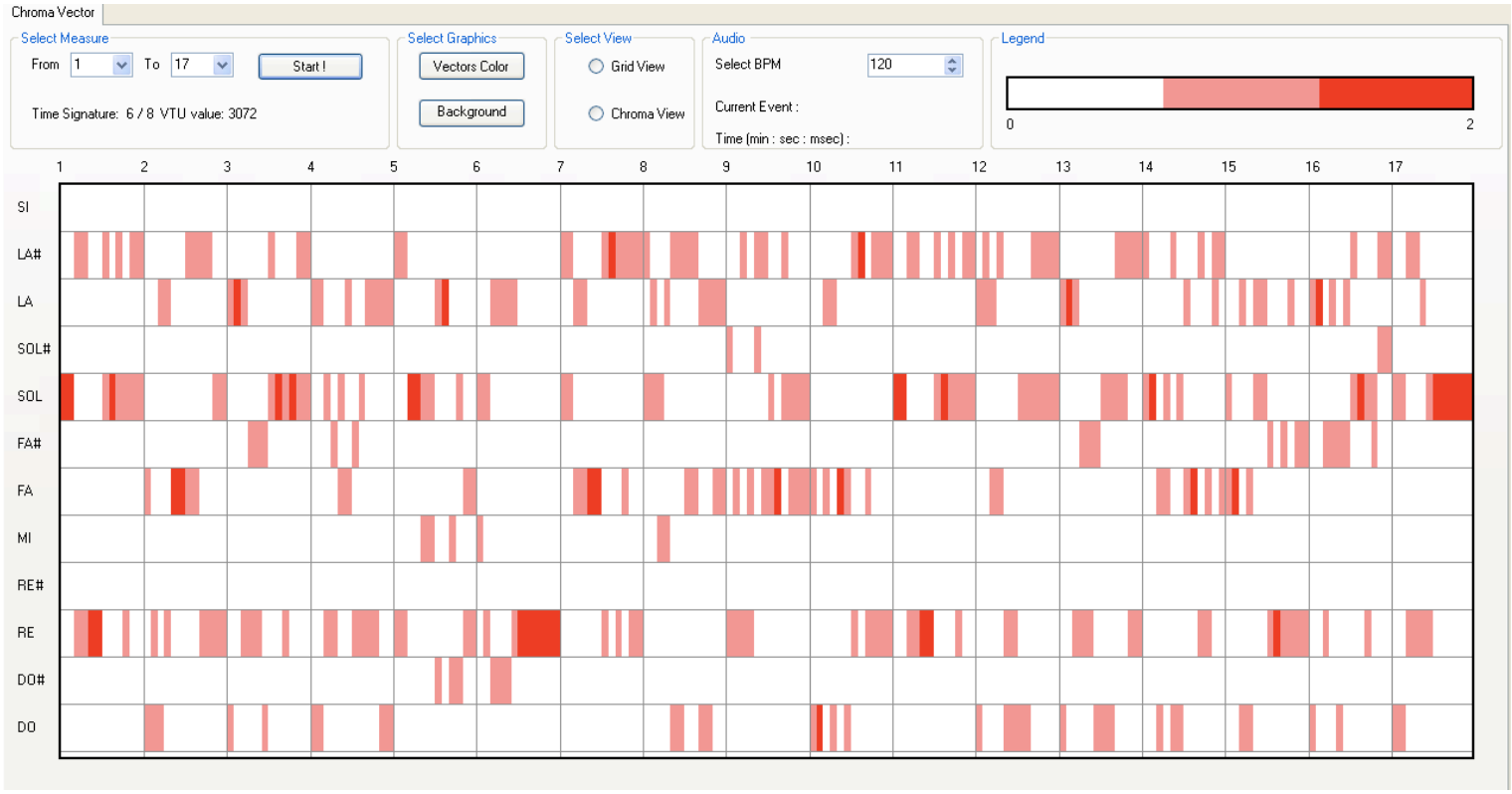

Fig. 47 - Test del software effettuato con menuet.xml ricavato dal brano di Hendel

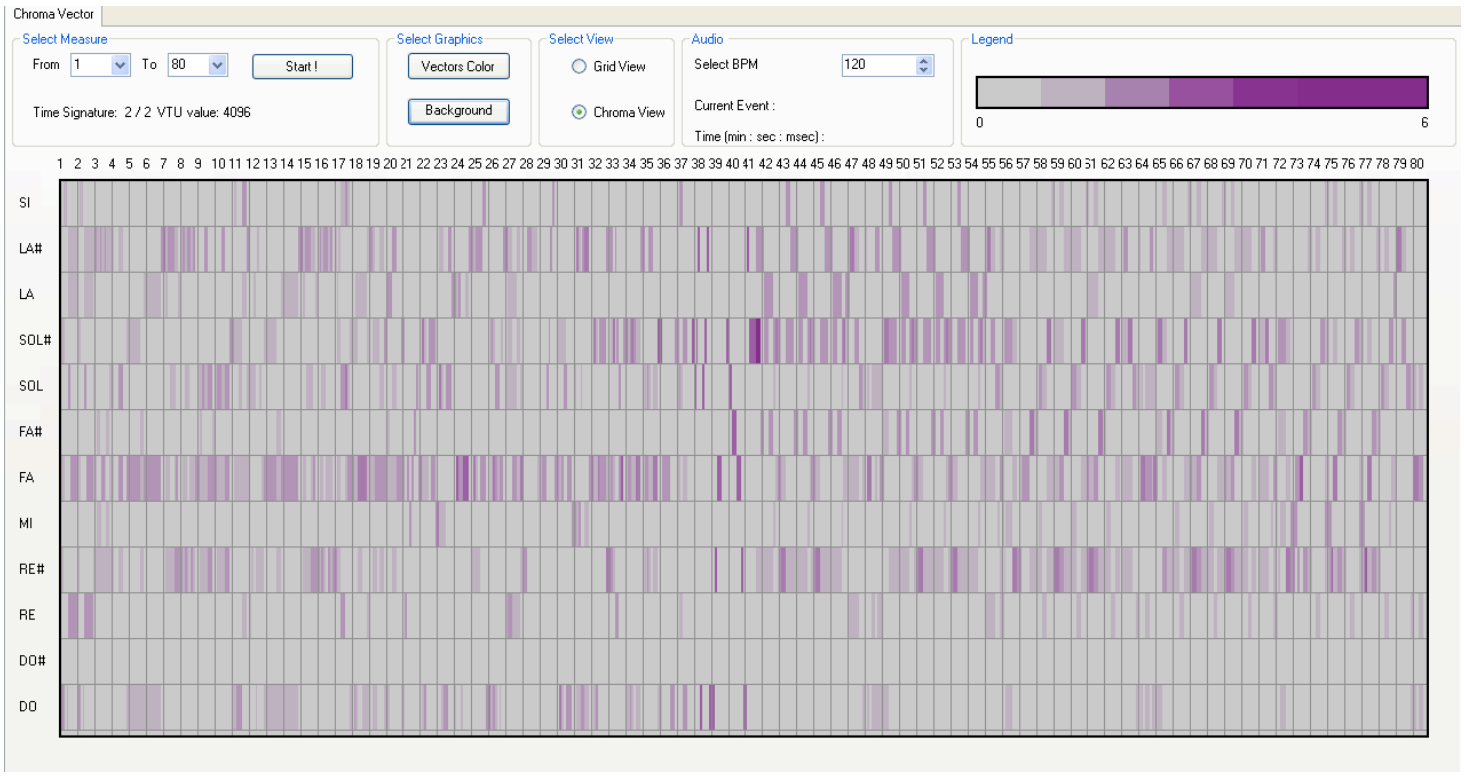

Fig. 48 - Test del software effettuato con il brano “King porter stomp” di jazz

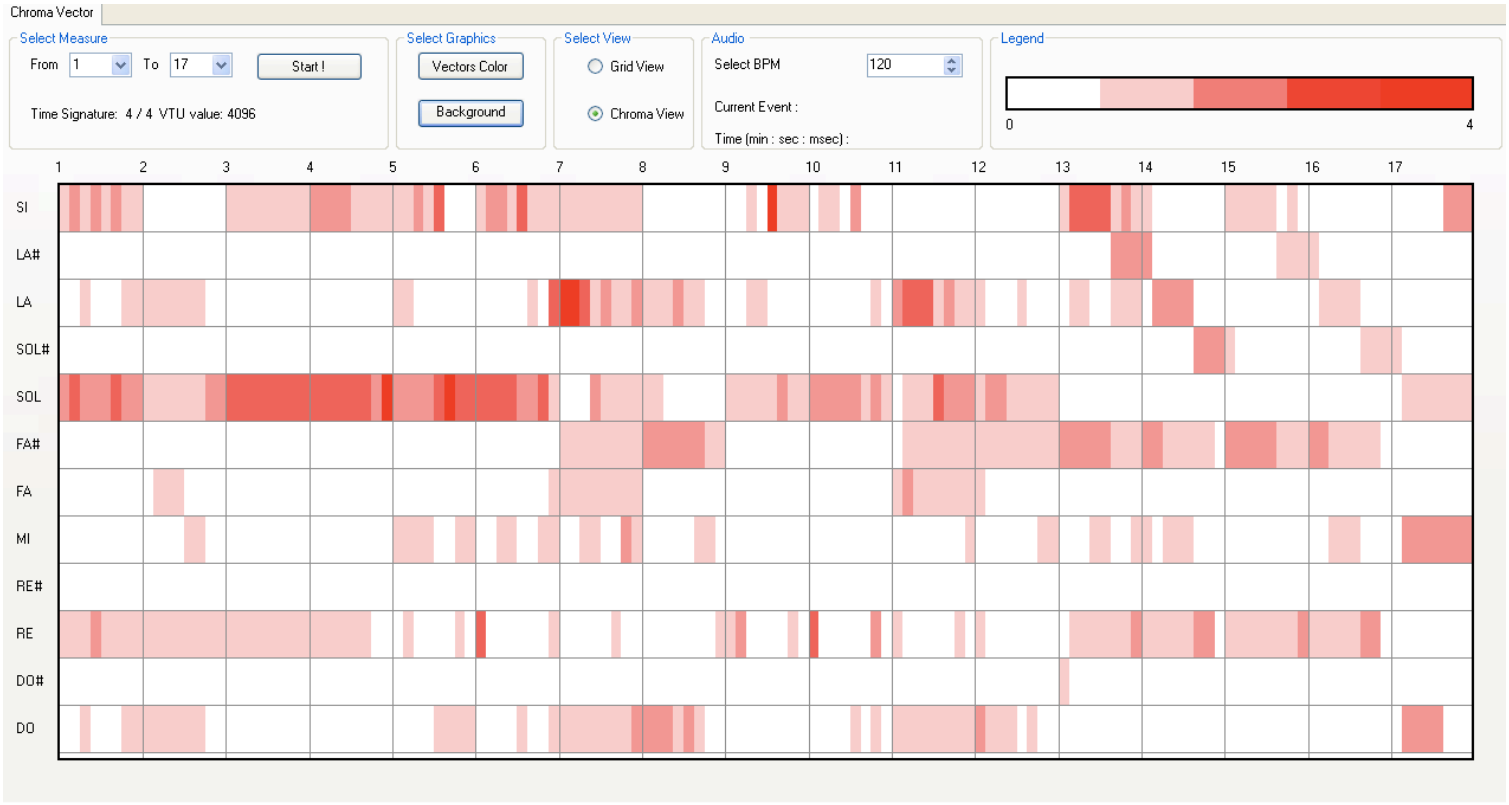

Fig. 49 - Chroma View del brano "Got to get into my life - The Beatles" da battuta 1 a battuta 17

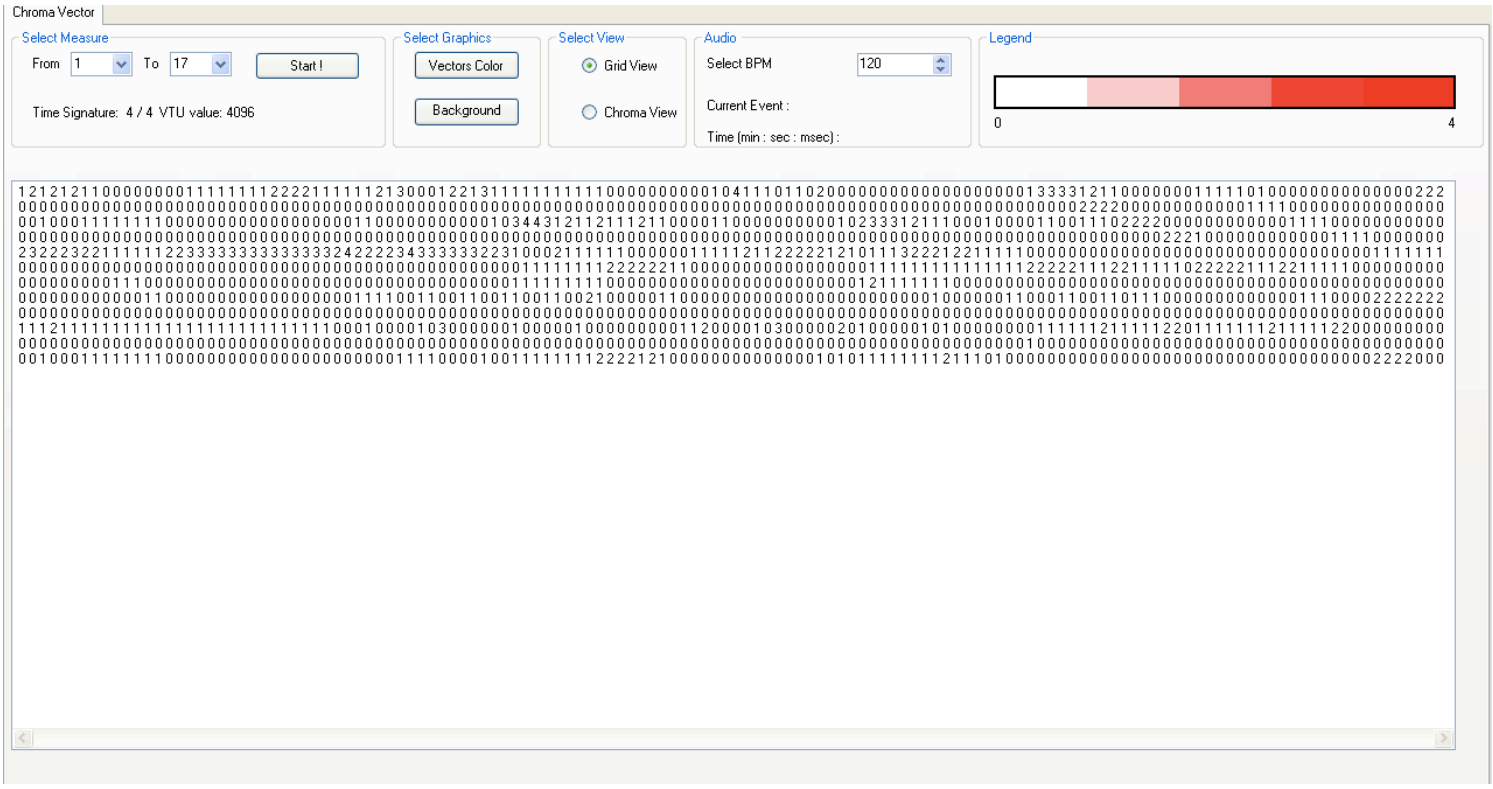

Fig. 50 - Grid View del brano "Got to get into my life - The Beatles" da battuta 1 a battuta 17

Questa trattazione ha visto come argomenti cardine: lo studio della caratteristica di *chroma* applicata in ambito musicale e del formato standard IEEE 1599 di xml musicale. L'aspetto motivazionale più forte è stato quello di integrare queste due realtà di informatica musicale fra loro. Ciò che è stato sviluppato è infatti un plug-in, integrato nel Framework-MX proprietario del formato (sviluppato nel LIM), denominato Chroma Vector con la capacità di estrarre da una partitura digitale strutturata all'interno di un file .xml, il relativo chromagram che ne evidenzia il flusso di energia suddiviso per i livelli di tono della scala temperata, privati delle differenze di ottava.

Il risultato dell'estrazione da un file di xml musicale è sicuramente un riproduzione più fedele dell'informazione veicolata in partitura, rispetto alla tesi sviluppata nell'articolo scientifico [3], che operava questa traduzione, e quindi il calcolo del chromagram, unicamente tramite scanning di software OMR.

Delle migliori scelte implementative potrebbero risolvere i pochi bugs presenti nel sistema come utilizzare una *pictureBox* di dimensione variabile, con l'aggiunta di uno *ScrollBar* orizzontale per una gestione dinamica delle grandezze dei vettori, oppure, migliorare la gestione dei cambi di tempo, e dei relativi cambi in termini di VTU.

All'interno del secondo capitolo viene illustrata una procedura di estrazione di chromagram da brani in formato audio, operando con dei filtraggi mirati sulle frequenze standard delle note. La creazione di questi grafici da audio è fondamentale per sviluppare algoritmi di MIR, confrontando i due diversi chromagram ottenuti dalle due diverse rappresentazioni musicali. Uno possibile sviluppo futuro di questa tesi, potrebbe appunto essere, la creazione di un algoritmo capace di svolgere questa estrazione, e la sua successiva integrazione in MX per un effettivo utilizzo nella creazione di applicativi, ad esempio, di audio matching.

La forte correlazione creata in IEEE 1599 fra le diverse rappresentazioni della musica e la versatilità della corrispondenza biunivoca dei chromagram potrebbero conferire al formato ulteriori sbocchi in ambito di Music Infromation Retrieval, oppure anche solamente per aumentarne il grado di strutturazione e indicizzazione delle informazioni musicali in esso contenute.

Bibliografia

1. R. Shepard, “Circularity in judgements of relative pitch”, *Journal of the Acoustical Society of America*, vol 36, 1964.

2. S. Pauws, “Musical key extraction from audio”, in *Music perception*, pag 544-552, Eindhoven, 2000.

3. F. Kurth, M. Muller, C. Fremerey, “Autometed synchronization of scanned sheet music with audio recording”, in *Proc. Int. Conf. on Music Information Retrieval ISMIR-07,* Università di Bonn, 2007.

4. M. Muller, F. Kurth, M. Clausen, “Audio matching via chroma-based statistical features”, in *Proc. Int. Conf. on Music Information Retrieval ISMIR-05,* London 2005.

5. D. P. W. Ellis, G. Poliner, “Identifying ‘cover songs’ with chroma features and dynamic programming beat tracking”, Columbia University dipartimento di Ingegneria Elettronica *LabROSA*, 2006.

6. M. Bartsch, G. Wakefield, “To cacth a chorus: using chroma-based rappresentations for audio thumbnailing”, University of Michigan EECS department, 2005.

7. D. Damm, M. Muller, “Lyrics-based audio retrieval and multimodal navigation in music collections”, Università di Bonn, *Information Retrieval for Music and Motion*, 2007.

8. L. A. Ludovico, “Key concepts of the IEEE 1599 Standard”, Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects, pp. 15-26, IEEE CS, Lugano, Switzerland, 2008.

9. S. Baldan, L. A. Ludovico, D. A. Mauro, “PureMX: Automatic Transcription of MIDI Live Music Performances into XML Format”, Proceedings of Computer Music Modeling and Retrieval 2009 (CMMR 2009), pp. 195-201, Copenhagen, Denmark, 2009.

10. A. Baratè, G. Haus, L. A. Ludovico, G. Vercellesi, “MXDemo: a Case Study about Audio, Video, and Score Synchronization”, Proceedings of AXMEDIS 2005 - 1st International Conference on Automated Production of Cross Media Content for Multi-channel Distribution, pp. 45-52, Florence, Italy, 2005.

11. M. Chand, ”GDI+ Programming with C#”, Addison Wesley, 2003.

12. J. Sharp, J. Jagger, “Microsoft Visual C# :NET Step by Step”, Microsoft Press, 2002.

13. E. White , “Pro .NET 2.0 Graphics Programming”, 2006.

Indice delle figure