UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE,
FISICHE E NATURALI
Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

Trascrizione automatica di performance

MIDI in IEEE1599

Candidato: *Marco Nardi Matr. 739500*

Relatore: *Prof. Luca Andrea Ludovico*

Correlatore: *Dott. Adriano Baratè*

Anno Accademico 2010-2011

Indice

Capitolo 1

Introduzione

Lo scopo della seguente tesi,come dice il titolo, è creare un programma per trascrivere automaticamente le performance eseguite con qualsiasi strumento o periferica in grado di comunicare attraverso il MIDI in un formato simbolico basato su XML, IEEE1599. La necessità di creare un programma di questo tipo nasce dalla volontà di migliorare l'esperienza di fruizione musicale da parte di un appassionato di musica mettendo a disposizione di quest'ultimo la possibilità di utilzzare i vantaggi offerti da questo formato. Il software Advanced MIDI to IEEE1599 consente, infatti, di salvare l'esecuzione della performance MIDI in formato SMF, insieme al documento IEEE1599 che permette di collegare i file salvati.

La tesi si suddivide in 7 capitoli:

- Il primo capitolo è dedicato a un'introduzione del lavoro.
- Il secondo fornisce una descrizione di IEEE1599 e delle sue parti inerenti la tesi.
- Il terzo fornisce una panoramica del MIDI.
- Il quarto da una panoramica degli strumenti informatici usati nella realizzazione del programma.

- Il quinto presenta il software realizzato evidenziando particolari algoritmi d'interesse.
- Nel sesto si descrivono i test effettuati per verificare il funzionamento del programma.
- Infine,nel capitolo conclusivo si verificano le possibili proposte di lavoro future.

Capitolo 2

Una breve Introduzione a IEEE1599

Lo standard fino ad oggi più utilizzato è il MIDI, nato come supporto alle performance musicali esso presenta alcune limitazioni dal punto di vista simbolico come la mancata rappresentazione delle pause, o l'ambiguità nella rappresentazione del tono. Per questo sono nati nuovi formati che hanno lo scopo di ben descrivere lo spartito musicale nelle sue varie sfaccettature. L'esempio più noto è il MusicXML, proposto dall'azienda americana Recordare operante nell'ambito dell'edizione digitale di spartiti musicali, che in questo aspetto della musica si sta proponendo come standard. Tuttavia questi standard non possono interagire fra di loro, per cui i simboli negli spartiti non possono riferirsi a frammenti audio, video, etc. Con un approccio del tutto innovativo IEEE1599 cerca di superare queste limitazioni, cercando di riunire insieme queste diverse forme di rappresentazione della musica, dalla notazione, all'esecuzione.

IEEE1599 è un formato basato su XML sviluppato presso il LIM (Laboratorio di Informatica Musicale - Dipartimento di Informatica e Comunicazione - Università degli Studi di Milano) e riconosciuto come standard internazionale nel 2008. Esso fornisce una descrizione della musica completa nei suoi vari aspetti: audio,video,spartiti e immagini correlate. Grazie alla possibilità di sincronizzare e collegare fra loro i diversi oggetti musicali è possibile vedere applicati i vantaggi di IEEE1599 in diversi settori dall'educazione musicale

alla commercializzazione di musica. Ad esempio, nel primo caso, la registrazione video dell'esecuzione di un singolo strumento musicale effettuata da un professionista e sincronizzata con la relativa parte dello spartito musicale, permette a uno studente di musica di imparare a eseguire passaggi ad alto coefficiente di difficoltà guardando direttamente l'esecuzione del passaggio da parte del professionista. Nel secondo caso, la possibilità di avere accesso a un database in cui le varie interpretazioni di un brano musicale assieme alle loro diverse proprietà siano riferibili ad uno unico prodotto, permette a un consumatore di musica di scegliere consapevolmente in un'ampia gamma di esecuzioni.

Le caratteristiche chiavi di IEEE1599 possono essere cosi sintetizzate[1]: ricchezza nella descrizione dei multimedia relativi allo stesso pezzo musicale. I contenuti simbolici, testuali, grafici, audio, e video possono essere codificati in un unico documento.

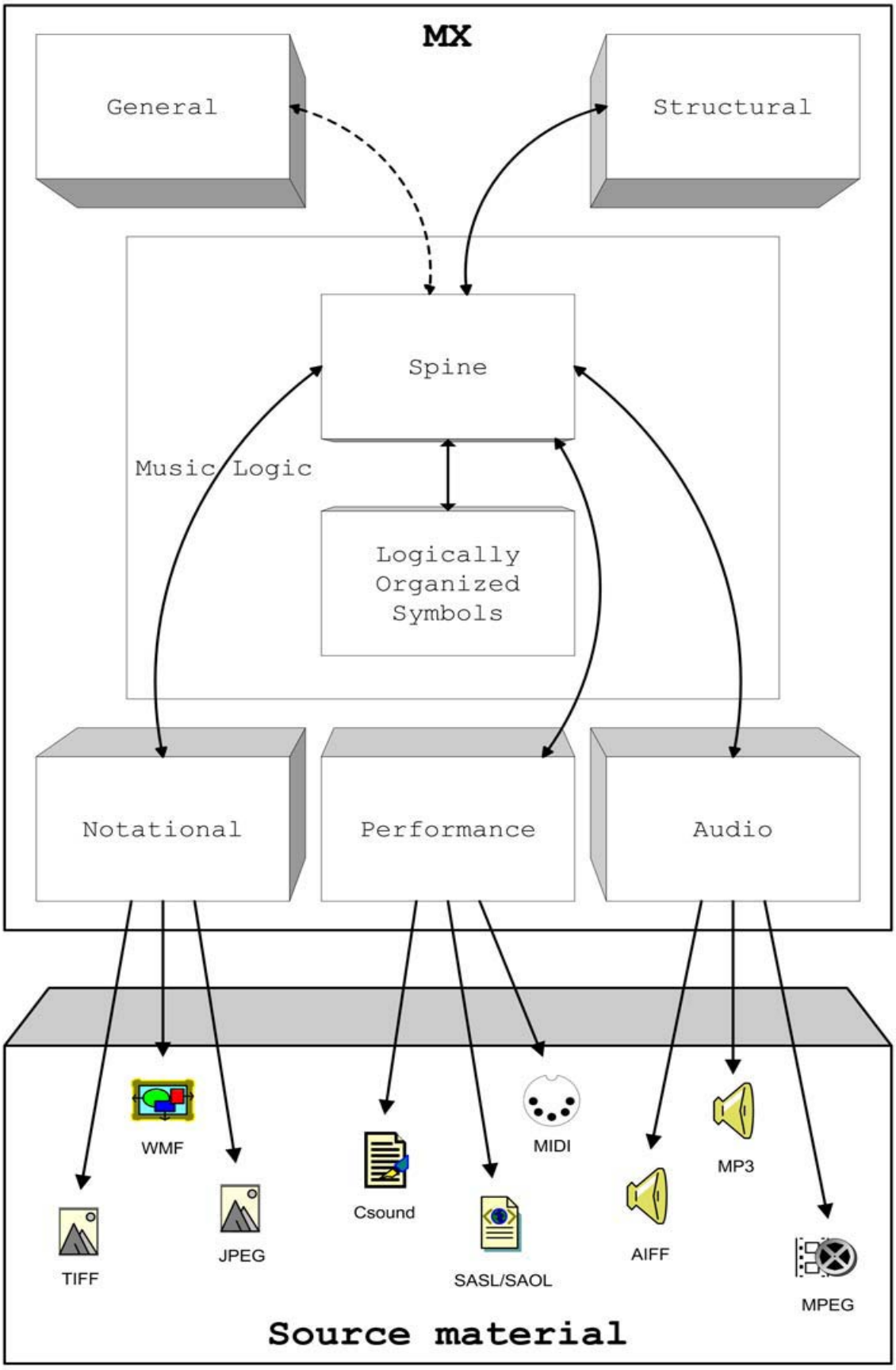
- possibilità di descrivere e collegare media dello stesso tipo, ovvero molte performance dello stesso pezzo, o molti spartiti appartenenti a diverse edizioni.
- pieno supporto per la sincronizzazione tra contenuti basati sul tempo. In un player dedicato, i contenuti audio e video possono essere sincronizzati con l'avanzamento dello spartito mentre la musica è riprodotta, anche cambiando fra una particolare esecuzione e un' altra.
- facile interazione con i contenuti musicali. In un navigatore IEEE1599 l'utente può selezionare una parte dello spartito e l'audio viene automaticamente spostato al punto corrispondente.

2.1 I Layers

Per descrivere, collegare, sincronizzare le diverse informazioni musicali, è stato pensato un ambiente multi-strato, in particolare sono stati usati 6 diversi strati (layer): General, Logic, Structural, Notational, Performance, Audio. I vari layer, ad esclusione del General, sono collegati insieme da una struttura astratta che rappresenta la relazione spazio-temporale intrinseca nella musica, lo spine. Ognuno di questi layer può contenere al proprio interno diverse descrizioni del brano musicale, ad esempio il layer Audio potrebbe contenere diverse registrazioni dello stesso brano codificate in diversi formati (WAV, MP3,...) oppure il layer Structural potrebbe fornire differenti analisi dello stesso brano. L'informazione musicale contenuta nei diversi strati può essere di due tipi: esplicita o implicita. In particolare i layer

General, Structural, Logical codificano esplicitamente i loro contenuti, mentre i rimanenti tre layer codificano l'informazione musicale implicitamente, ovvero collegano informazioni presenti in IEEE1599 con file esterni che rappresentano i diversi media. Vediamo in concreto una breve panoramica di questi strati:

- General: contiene i metadati che forniscono una descrizione generale del lavoro musicale, ad esempio informazioni di catalogo riguardanti il brano quali il titolo del brano, il titolo dell'album, l'autore, eccetera.
- Logic: fornisce una descrizione della musica dal punto di vista simbolico.
- Structural: contiene la descrizione esplicita degli oggetti musicali e della loro relazione sia da un punto di vista compositivo sia da un analisi musicologica svolta manualmente o automaticamente. In altre parole questo layer si occupa di identificare le relazioni interne al brano in esame: temi musicali, soggetti, sequenze o segmenti che si ripetono, o che presentano un particolare interesse.
- Notational: contiene le rappresentazioni grafiche dello spartito.
- Performance: supporta i formati file che codificano i parametri delle note da eseguire e i parametri dei suoni da creare, ma sempre al fine di una produzione musicale sintetica, ossia da parte dell'elaboratore. Alcuni di questi formati file sono il MIDI e Csound. La temporizzazione degli eventi in questo layer è relativa.
- Audio: contiene le registrazioni digitali del pezzo. Per poter collegare un generico file audio allo spine, è necessario estrarre le caratteristiche degli eventi musicali relative alla loro localizzazione temporale. In questo layer la temporizzazione è assoluta.


MX
General
Structural
Spine
Music Logic
Logically Organized Symbols
Notational
Performance
Audio
WMF
TIFF
JPEG
Csound
SASL/SAOL
MIDI
AIFF
MP3
MPEG
Source material

2.2 Logic layer

Di particolare interesse per noi è il layer Logic poiché è all'interno di questo che i dati MIDI relativi alle note vengono estrapolati e trascritti in CWN (Common Western Notation). Esso si scompone in 3 sottoelementi: lo spine, il los, il layout.

- Spine: come detto in precedenza, è il collante dei diversi strati. Esso è composto da una serie di eventi, ordinati e marcati tramite un identificatore univoco, ciascuno dei quali presenta un riferimento nel dominio dello spazio e del tempo. Questi eventi non corrispondono a simboli dello spartito o campioni audio, ma l'autore del documento decide di volta in volta a cosa correlare i vari eventi; la mancanza di correlazione fra l'evento e un qualunque dato fa perdere l'utilità stessa di quell'evento.

La possibilità per oggetti eterogenei di riferirsi a uno stesso evento logico permette a questi di legarsi e sincronizzasi fra loro. La posizione spaziale e temporale degli eventi è espressa in maniera relativa, ovvero facendo riferimento all'evento precedente, per questo sono usati due attributi, il timing per la distanza temporale, e l'hpos per la distanza spaziale. Il timing viene espresso in VTU (Virtual Time Units) e l' hpos in VPX (Virtual Pixels), Le VTU non hanno un'unità di misura fissata, il significato da attribuire alle VTU viene di volta in volta deciso dall'utente. La scelta delle VTU da associare a un simbolo influisce sulla durata delle note rappresentabili, poiché il valore delle VTU dev'essere sempre un numero intero. Se ad esempio alla nota da 1/4 fossero assegnate 16 VTU la massima durata rappresentabile sarebbe data dal valore ritmico di 1/64

- LOS: in questo sub-strato si trova il contenuto informativo simbolico del brano (note, pause, segni di articolazione, dinamiche,…). Gli eventi musicali dello spine vengono definiti nello standard CWN all'interno del los. Qui il pitch viene descitto come nome della nota e ottava di appartenenza e il valore di durata viene espresso sotto forma di frazione. All'interno del los non è obbligatorio usare lo standard CWN per descrivere gli eventi, questo permette una compatibilità di IEEE1599 con altri sistemi di notazione come la notazione gregoriana. L'informazione in questo sotto livello è fortemente strutturata[2]:

1.una partitura è costituita da più accollature;

2.un’accollatura è formata da più pentagrammi;

3.un pentagramma contiene (o meglio, può contenere) più parti;

4.una parte può essere suddivisa in più voci;

5.una voce intera viene considerata battuta per battuta;

6.una battuta contiene accordi e pause;

7.un accordo contiene una o più note.

Alcune osservazioni: per prima cosa la descrizione delle battute parte per parte (prima i contenuti delle misure da 1 a n della parte 1, poi quelli delle misure da 1 a n della parte 2, e così via) non consente una chiara visione dei rapporti di sovrapposizione verticale o di sincronizzazione temporale a livello di partitura, al contrario di quanto avviene in una partitura stampata o manoscritta. In secondo luogo, a causa della sua posizione gerarchica, l'accordo rappresenta la sovrapposizione di note all’interno di una stessa parte o voce, ma non quella tra diverse linee melodiche. Infine si osservi che la singola nota, viene vista e codificata come accordo degenere. Questo principio è di particolare importanza, poiché troverà una corrispondenza logica nel codice.

Lo schema proposto si traduce in IEEE1599 nel seguente scheletro:

```
<los>
        <!-- descrizione dell’accollatura -->
        <staff_list>
                <!-- descrizione delle proprietà dei pentagrammi -->
                <staff id="staff_1">   … </staff>
                <staff id="staff_2">   … </staff>
        </staff_list>
        <!-- descrizione delle singole parti -->
        <part id="flauto">
                <!--elenco delle voci della parte -->
                <voice_list>
                        <voice_item id="voce_flauto"/>
                </voice_list>
                <measure number="1"> … </measure>
                <measure number="2">         … </measure>
                …
        </part>
        <part id="pianoforte" >
                <voice_list>
                        <voice_item id="voce1"/>
```

```
                <voice_item id="voce2"/>
                …
            </voice_list>
                <measure number="1"> … </measure>
                <measure number="2">        … </measure>
                …
                </part>
             <!-- descrizione delle parti restanti… -->
</los>
```

- Layout: contiene informazioni generiche sull'implemetazione grafica dei contenuti simbolici, ma non è di interesse per la nostra applicazione.

Capitolo 3

MIDI - Musical Instrumental Digital Interface

MIDI è uno standard per il collegamento digitale di strumenti musicali elettronici: permette a computer, sintetizzatori, schede audio, campionatori e drum machine di controllare l'un l'altro e scambiarsi dati di sistema in tempo reale. Questo standard nasce grazie alla collaborazione di diverse compagnie di sintetizzatori: la Roland e la Sequential a cui si aggiunsero in seguito la Yamaha e la Oberheim. Grazie a questa forte collaborazione esce nel 1983 lo standard MIDI 1.0; nello stesso anno uscì sul mercato il primo strumento MIDI, il Sequential Circuits Prohet-600, subito seguito dal Roland JX-3P.

ll MIDI 1.0 standard prevede:

- specifiche hardware che determinano come devono essere trasferite le informazioni digitali.
- specifiche file;
- specifiche messaggi.

Di queste tre specifiche a noi interessano principalmente le specifiche sui messaggi o protocollo.

3.1 Messaggi MIDI

I messaggi MIDI più semplici che un sintetizzatore può ricevere sono la comunicazione di avvio e di interruzione della riproduzione di una specifica nota. Altre informazioni che possono essere inviate al sintetizzatore sono il volume, il cambio dei suoni, la modulazione della nota, e così via.

La comunicazione tra periferiche MIDI avviene attraverso l'invio di messaggi codificati in byte, questi possono essere byte di stato o data byte. Si distinguono due principali categorie di messaggi: i Channel message e i System message. I Channel message sono i messaggi relativi ad un certo canale o a più canali, i System message sono relativi al sistema nel suo insieme, dando ad esempio istruzioni relative al tempo, alla sincronizzazione, etc. Il protocollo MIDI distingue i due tipi di messaggio grazie a una particolare sequenza di 8 bit chiamata Status byte. I byte di stato hanno il bit più significativo impostato a 1, nei Channel message i 3 bit seguenti possono avere un valore che va da 000 a 110 e indicano al dispositivo il comando da eseguire, gli ultimi 4 bit identificano il canale MIDI. Nei System message i 3 bit seguenti hanno il valore di 111 e i seguenti 4 bit identificano i differenti messaggi di sistema. Ma vediamo più da vicino questi messaggi.

Channel message:

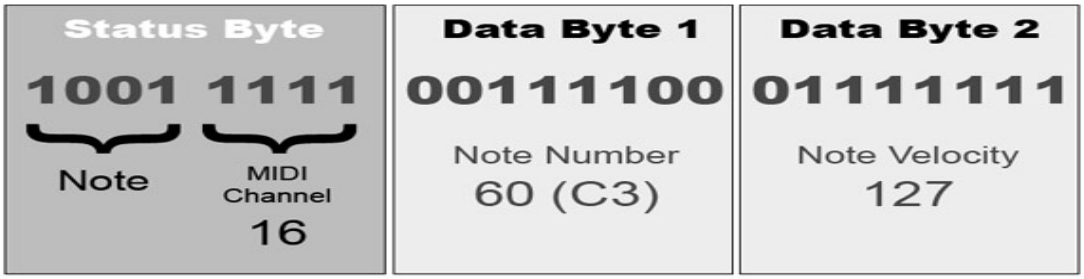
- I Channel Voice messages sono gli eventi MIDI essenziali che rappresentano un' esecuzione musicale, i più comuni sono i messaggi di Note On e Note Off.
- I Channel Mode messages istruiscono i dispositivi a mandare o ricevere messaggi in una certa maniera definita dalla modalità di invio. Ad esempio spegnere tutte le note o suoni.

System message:

- I System Common messages sono comuni a tutti gli strumenti o periferiche della configurazione MIDI. Spesso i sequencer li utilizzano per sincronizzare la sequenza con altri media come il video attraverso il MIDI Time Code (misura del tempo assoluta). Altri usi sono la posizione della canzone e la selezione della canzone.
- I System Real Time sono comandi di sincronizzazione fra apparecchiature MIDI basate sul clock (misura del tempo relativa alla velocità della sequenza) utilizzati per controllare le sequenze, come i comandi start e stop .
- I System Exclusive messages sono chiamati "esclusivi" perchè destinati a inviare le

informazioni specifiche di ogni produttore il quale, distintamente per ogni dispositivo MIDI, può stabilire liberamente il formato del messaggio e le funzionalità relative. Alcuni esempi di utilizzo sono l'invio o la ricezione di impostazioni degli strumenti come le patch, le memorie di esecuzione o i campioni delle forme d'onda.

Per l'applicazione sviluppata saranno presi in considerazione solo i Channel Voice messages.. Essi, come detto prima, sono composti da 1 byte di stato seguito da 1 o 2 data byte[4].



I byte di stato hanno il bit più significativo impostato a 1, i 3 bit seguenti indicano al dispositivo il comando da eseguire, infine gli ultimi 4 bit identificano il canale MIDI. Il MIDI permette di operare su 16 canali diversi. Le periferiche MIDI possono accettare o ignorare gli status byte a seconda dei canali che sono impostati a ricevere messaggi.

I data byte hanno il bit più significativo impostato a 0 e seguono gli status byte impostandone i parametri, questi assumono un significato diverso a secondo del comando cui sono associati. Ad esempio, un comando program change è seguito da un data byte contenente il numero del timbro dello strumento (o patch) che deve essere utilizzato. I channel voice message più importanti ai fini della nostra applicazione sono quelli di Note On e Note Off.

	Status byte		1 Data byte	2 Data byte
	Command	Channel	Note number	Velocity
MIDI values	Note On	Ch. 3	60	100
Binary	1001	0010	0011 1100	0110 0100
MIDI values	Note Off	Ch. 3	60	100
Binary	1000	0010	0011 1100	0110 0100

In questi messaggi i data byte rappresentano l'altezza della nota e la velocità di rilascio (in linguaggio musicale corrisponde alla dinamica: un valore di velocity alto indica un ”forte”, mentre un valore basso indica un “piano”). I messaggi di Note On e Note Off si distinguono per i primi 4 bit dello status byte, ma in realtà anche un messaggio Note On con il valore della velocity impostato a 0 equivale ad un messaggio di Note Off, per cui il modo in cui la nota viene spenta dipende dal dispositivo. E' da notare che essendo il tono della nota e l'ottava di appartenenza rappresentati in termini numerici (al Do centrale, C4, è assegnato 60, la nota appena sopra , C#4, è 61, e quella sottostante, B3, è 59) non viene fatta distinzione fra diesis e bemolle per cui una nota il cui valore è 61 può indifferentemente corrispondere a un Do diesis o a un Re bemolle.

Nel MIDI la durata della nota è codificata come una coppia di messaggi NoteOn - NoteOff e questo significa che nel MIDI una semiminima legata ad una croma è equivalente a una semiminima puntata.

	Status byte		1 Data byte	2 Data byte
	Command	Channel		
MIDI values	Program Change	Ch. 7	Distortion Guitar	/
Binary	1100	0110	0001 1110	0110 0100
MIDI values	Control Change	Ch. 5	All Sound Off	0
Binary	1011	0100	0111 1000	0000 0000

Nella tabella sopra sono illustrati 2 esempi, lo status byte del primo messaggio contiene un comando di program change che si riferisce al canale 7, questo comando dà l'istruzione di cambiare il timbro allo strumento MIDI nel programma numero 30 che corrisponde al timbro distortion guitar, in questo caso si è supposto (come nella norma) che lo strumento adotti lo standard GM. E' da notare che col comando program change viene inviato solo un data byte contenente il numero di programma. Nel secondo esempio della tabella lo status byte del messaggio contiene un comando di control change e il canale a cui inviare il messaggio, questo si differenzia un po' dagli altri messaggi poichè il primo data byte viene usato per specificare la funzione di control change da eseguire, mentre il secondo data byte ha il normale compito di indicare il valore del comando scelto. In questo caso la funzione che viene eseguita è All Sound Off e il secondo data byte contiene il valore default di 0. Questa funzione ha l'effetto di spegnere immediatamente senza tener conto degli effetti applicati tutti

suoni e in particolare i suoni attivati da messaggi Note On che non hanno ancora ricevuto il messaggio di spegnimento Note Off. Questo messaggio è solitamente utilizzato dai sequencer quando si vogliono interrompere immediatamente tutti i suoni con la pressione del pulsante Stop.

3.2 Standard Midi File (SMF)

Per rendere elaborabili, riproducibili, accessibili da diversi sistemi le performance MIDI è necessario che i messaggi MIDI vengano in qualche modo salvati. Per questo è stato creato lo Standard MIDI File definito ufficialmente nel 1988 dalla MMA (Manufactuer MIDI Association). Esso è un comune formato file usato dalla maggior parte dei software musicali e periferiche hardware per salvare informazioni inclusi il titolo, i nomi delle tracce, gli strumenti da usare e la sequenza degli eventi musicali, come le note, essenziali per la riproduzione della performance. Affianco agli eventi MIDI viene salvato un time-stamp che indica quanti tick aspettare prima di riprodurre l'evento. Poiché non viene salvato nessun dato audio, lo spazio occupato da un file SMF è molto piccolo, specialmente comparato a un file audio. Questa standardizzazione permette a un software di creare e salvare file MIDI che possono poi essere letti da un qualsiasi programma capace di riprodurre MIDI.

Esistono tre diversi formati SMF numerati da 0 a 2 :

- Formato 0: tutte le tracce o canali sono combinati in una singola traccia che contiene tutte le informazioni degli eventi di tutte le tracce del brano.
- Formato 1: ad ogni canale MIDI è assegnata una diversa traccia. Il tempo e la metrica sono salvate nella prima traccia che fa da riferimento a tutte le altre. E' il formato più utilizzato dai sequencer.
- Formato 2: come nel formato 1 ad ogni canale MIDI è assegnata una diversa traccia, ma in questo caso è possibile assegnare ad ogni traccia una metrica e un tempo differenti. E' particolarmente adatto per salvare una serie di pattern di batteria per una drum machine.

3.2.1 La struttura dello SMF

I byte usati per salvare o trasmettere informazioni a un SMF sono detti MIDI chunk. Essi si dividono in 2 categorie i MThd chunk (MIDI Track, header chunk) e i MTrk chunk (MIDI Track, track chunk). Per capire qual è il tipo di chunk trasmesso vengono usati 4 bytes chiamati chunk ID che rappresentano caratteri ASCII, questi sono seguiti da altri 4 byte che rappresentano la lunghezza dei dati e infine vi sono un numero variabile di byte che rappresentano l'informazione dello specifico chunk.
Chunks:

TYPE 4 bytes (ASCII)	TYPE 4 bytes (32-bit binary)	DATA Variable Bit Length (binary data)		
MThd	6	FORMAT	TRACKS	DIVISION
MTrk	Length	Delta Time	Events	

Il MThd chunk si trova una sola volta all'inizio dello SMF e definisce il file MIDI. La lunghezza dei dati è fissata a 6 byte, 2 byte rappresentano il formato SMF, 2 byte indicano il numero delle tracce della canzone, e i rimanenti 2 byte sono usati per la divisione della sequenza che definisce l'unità predefinita del delta time (il tempo trascorso fra un evento e il precedente evento). Il delta time può essere misurato o in PPQN (numero di pulsazioni in una nota da un quarto) con il bit 15, il più alto, impostato a 0 o in SMPTE, in questo caso il bit 15 ha il valore impostato a 1 e i 7 bit seguenti definiscono il valore dei frame per secondo, mentre gli altri 7 definiscono il numero di tick per frame.

Il chunk MTrk contiene la registrazione di tutti gli eventi MIDI e come il chunk MThd è diviso in tre parti: il chunk ID che lo identifica, la lunghezza che rappresenta il numero di bit che contengono i dati e infine i dati stessi. I dati contengono gli eventi MIDI e il loro delta time insieme ad altri dati non MIDI (come il nome della traccia, le impostazioni di tempo, etc) per una traccia. Il numero dei chunk MTrk che seguono il chunk MThd dipende dal tipo di formato e il numero di tracce impostati nel chunk MThd. Ad esempio se il formato imposto è lo 0 ci sarà un solo chunk MTrk dopo il chunk MThd. I dati che si possono trovare in un chunk MTrk sono:

- eventi MIDI, sono il cuore del file MIDI poichè rappresentano tutti i messaggi channel voice e channel mode e il running status (mantiene lo stesso status byte per i

successivi eventi MIDI risparmiando i dati MIDI necessari per comunicare informazioni);

- eventi SysEx cioè i dati specifici di un particolare produttore;
- meta eventi usati per il nome della traccia, i testi, il tempo, la metrica, eccetera. Ogni Meta evento inizia per FF.

4D546864		MThd
00000006		Lunghezza del blocco
0000		Formato 0
0001		Una traccia
0060		96 PPQN
4D54726B		MTrk
00000029		
Delta-time	**Evento**	
00	FF580404021808	5804: Metrica; 0402: tempo di 4/4; 18: 24 MIDI clocks/click; 08: 8 note da 1/32 in 24 MIDI clocks
00	FF51030F4240	51: Tempo, 0F4240: 1*10^6 microsecondi per nota da 1/4
00	904A54	90: Note On Ch1; 4A: numero nota (RE5); 54: velocity (84)
00	4D54	running status; 4D: numero nota (FA5); 54: velocity (84)
0C	4A00	running status; 4A: numero nota (RE5); velocity (0)
00	4D00	running status; 4D: numero nota (FA5); velocity (0)
00	FF2F00	Fine della traccia.
60	91434060	

E' interessante notare l'utilizzo del running status. Questa è una tecnica utilizzata per evitare di trasmettere informazioni ridondanti, essa consiste nell'omissione dello status byte del messaggio nel caso questo sia uguale allo status byte del messaggio precedente. Ad esempio nel caso di più messaggi Note On consecutivi l'informazione dello status byte (90) può essere trasmessa una sola volta, mentre vengono trasmessi i data byte indicanti la nota e la velocity per ogni messaggio.

3.2.2 General MIDI (GM)

La qualità del suono riprodotto dipende dagli strumenti MIDI utilizzati per la riproduzione e

questo comporta il problema di non avere sempre un suono uniforme. Nonostante alcuni strumenti MIDI abbiano una qualità del suono molto elevata, e nonostante il comando program change consenta a noi di dire allo strumento MIDI di usare una specifica patch, non è specificato che suono noi otterremmo, questo infatti può variare da strumento a strumento; ciò significa, per esempio, che un brano riprodotto da un sintetizzatore MIDI col suono di un organo potrebbe essere riprodotto col suono della batteria o altro su un diverso strumento MIDI. Per risolvere questo problema è stato creato il General MIDI (GM). Pur non essendo una parte del MIDI, GM definisce le caratteristiche specifiche per gli strumenti MIDI. Di conseguenza, con uno strumento GM, noi sappiamo che il numero di program change 10 è riservato a suoni di percussione, e sappiamo anche che una qualsiasi nota trasmessa sul canale a cui è assegnato questo program change produrrà sempre un suono di percussioni. Ad altri numeri di program change corrispondo specifici strumenti: l'acoustic grand piano si trova al numero 1, il violino al numero 41, e così via.

Inoltre GM dà delle specifiche riguardo la polifonia, la velocity, e la modalità multi timbro. Quindi rispettando lo standard GM si moltiplicano le possibilità di ottenere una riproduzione corretta sui vari sistemi.

Capitolo 4

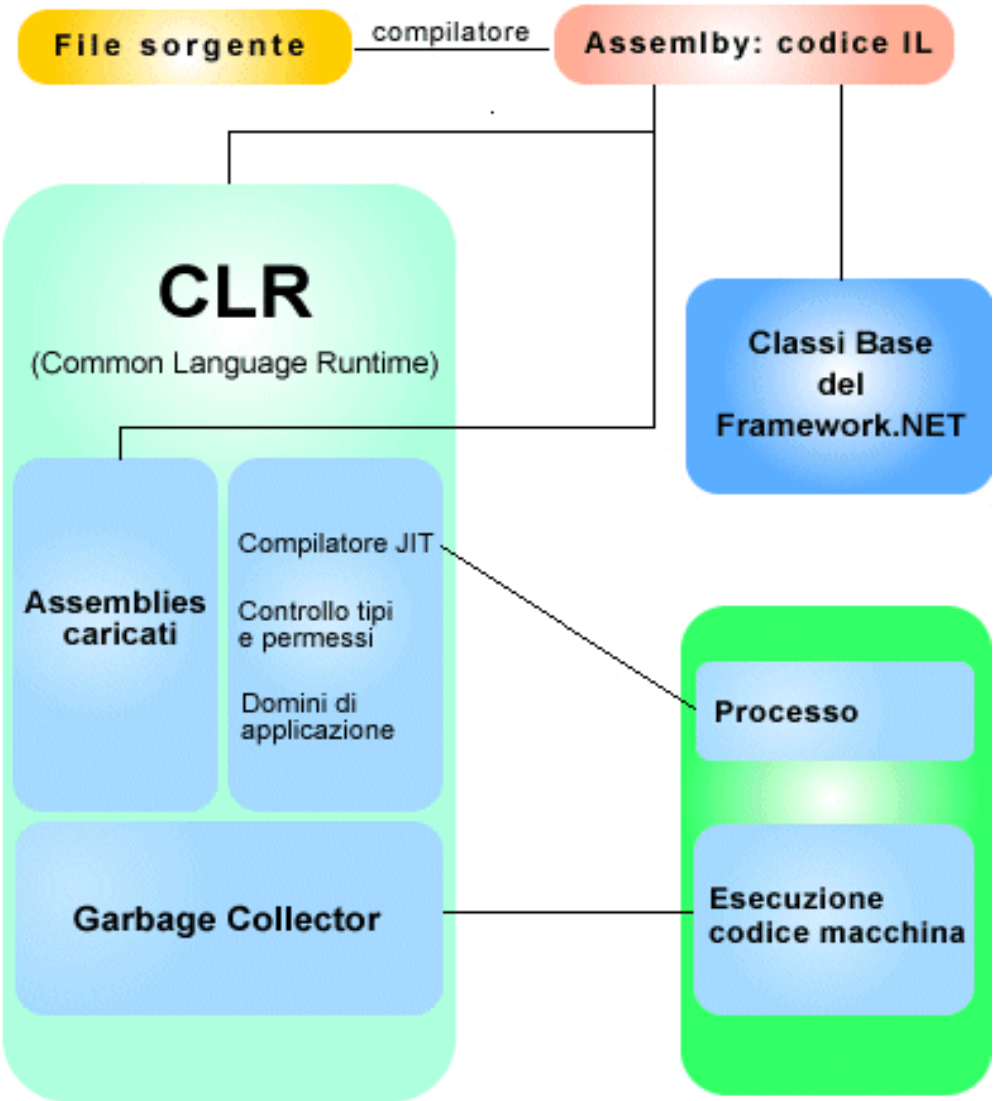
Le Tecnologie Informatiche Utilizzate

Il programma è stato scritto nel linguaggio C# all'interno del framework .NET in ambiente di sviluppo Visual studio 2010. Le informazioni riguardanti queste tecnologie sono state tratte dai libri Professional C# 4 and .NET 4[5], e Pro WPF in C# 2010[6].

4.1 .NET

Microsoft lanciò l'inziativa .NET nel Giugno del 2000. Il framework .NET fu concepito come completamente indipendente dalla piattaforma, come Java, e indipendente dal linguaggio; ciò significa che i programmatori che sono abituati all'utilizzo di linguaggi diversi supportati dal framework .NET possono contribuire allo sviluppo dello stesso software scrivendo nel proprio linguaggio preferito. .NET è orientato allo sviluppo di Web Application. Il framework gestisce l'esecuzione di applicazioni e servizi Web. Esso fornisce, inoltre, molte altre funzionalità incluse la sicurezza dell'applicazione e la gestione della memoria. Quindi, come Java, C# ha una garbage collection che autogestisce. Gli sviluppatori Microsoft hanno, inoltre, progettato .NET per essere indipendente dalla piattaforma. Ciò è reso possibile grazie alla virtual machine Common Language Runtime (CLR) che entra in azione dopo che l'IDE .NET ha compilato il codice nel Microsoft Intermediate Language (MSIL). MSIL permette la portabilità fra vari sistemi operativi, linguaggi di programmazione, e fornisce caratteristiche come la gestione della memoria e la sicurezza. Se uno sviluppatore crea un programma con linguaggi differenti, il compilatore converte questi in MSIL e dopo usa CLR per unirli. CLR

gestisce l'esecuzione dei programmi .NET compilando MSIL in linguaggio macchina. Grazie a un processo chiamato Just-in-Time (JIT) il CLR ha una buona efficienza nelle prestazioni. Il processo JIT compila in linguaggio macchina solo le porzioni di codice nel momento in cui queste vengono chiamate e che a turno vengono eseguite dalla CPU. CLR oltra ad assicurare i servizi di allocazione e gestione della memoria, e di sicurezza, garantisce l'applicazione dell'indipendenza dai tipi, la gestione delle eccezioni e dei thread., e infine rende il codice specifico per ogni macchina poiché compila il codice nel linguaggio macchina nativo della piattaforma specifica.



I linguaggi supportati da .NET sono: C#, Visual C++ .NET, Visual Basic .NET, JScript, Perl, etc. Indipendentemente dal linguaggio utilizzato .NET mette a disposizione degli sviluppatori una libreria ricca di classi, la FCL (Framework Class Library), che permette di saltare l'implementazione di molte funzioni fondamentali, grazie alla riusabilità delle sue componenti.

La scelta è ricaduta sul framework .NET a causa di alcune sue peculiarità:

- il framework è interamente basato sui principi della programmazione object-oriented.
- fornisce all'utente una base class library che copre diversi campi come la user interface, il data access, web application,etc. In particolare la libreria contiene una classe in grado di consentire all'utente di manipolare documenti XML, per cui si rivela utile nel caso di IEEE1599.

4.2 C#

C# è un linguaggio di programmazione object-oriented sviluppato da Microsoft appositamente per il framework .NET. Ufficialmente Microsoft definisce C# come un linguaggio semplice, moderno, indipendente dai tipi e orientato a oggetti derivato da C e C++. In realtà uno sguardo più vicino al codice di C# fa notare come questo sia simile al C++, ma anche al JAVA tanto che molte parole chiave sono le stesse. Le caratteristiche principali di C# sono:

- Pieno supporto per classi e programmazione orientata a oggetti, incluse sia le interfacce sia l'implementazione dell'ereditarietà.
- Un consistente e ben definito insieme di tipi base.
- Un supporto incorporato per la generazione automatica di documentazione XML.
- Pulizia automatica della memoria allocata dinamicamente.
- Completo accesso alla libreria base di classi di .NET e un facile accesso alle Windows API.
- Puntatori e accesso diretto alla memoria nel caso questo si renda necessario.
- C# può essere usato per scrivere pagine web dinamiche ASP.NET e servizi web XML.

4.3 WPF (Windows Presentation Foundation)

Sviluppato da Microsoft, WPF(Windows Presentation Foundation) serve a facilitare al programmatore la creazione di un' interfaccia utente.

Le caratteristiche di WPF:

- Invece di usare controlli fissati a precise coordinate, WPF enfatizza l'utilizzo di un layout flessibile che dispone i controlli basandosi sul loro contenuto.
- Anziché disegnare pixel, in WPF si possono usare delle primitive: forme base, blocchi di testo, e altri oggetti grafici. Inoltre permette di sovrapporre più layer con differente opacità.
- Supporto per media audio e video.
- Interfaccia utente dichiarativa. Sebbene sia possibile costruire una finestra WPF tramite codice, Visual Studio ha un approccio diverso: esso serializza ogni contenuto della finestra in un insieme di tag XML all'interno di un documento XAML. XAML (Extensible Application Markup Language) è la dichiarazione XML usata per definire e collegare vari oggetti User Interface, essa consente inoltre di specificare i comportamenti degli oggetti; il vantaggio di questo approccio è che l'interfaccia utente è completamente separata dal codice.

4.4 Visual Studio

Visual Studio è un ambiente di sviluppo integrato (IDE) appartenente a Microsoft ed è' usato come supporto per i programmatori. Le sue caratteristiche sono:

- editor di visualizzazione del progetto. permette di collocare l'interfaccia utente e i controlli di accesso ai dati all'interno del progetto.
- supporto delle finestre. queste permettono di vedere e modificare gli aspetti del progetto, come le classi nel codice sorgente e le proprietà disponibili per le classi Windows Form. Queste finestre possono essere anche usate per specificare le opzioni di compilazione, così come le assemblies a cui il codice deve far riferimento.
- l'abilità di compilare all'interno dell'ambiente: anziché dover avviare il

compilatore C# dalla linea di comando, Visual Studio esegue questa operazione passando al compilatore i parametri rilevanti, come le assemblies a cui far riferimento e il tipo di assembly che deve essere creato (.dll, exe). Esso può anche eseguire l'eseguibile compilato.

- debugger integrato: possibilità di inserire breakpoint e osservare il comportamento delle variabili all'interno dell'ambiente.
- accesso alla documentazione MSDN.

4.5 C# MIDI toolkit

E' una libreria per lavorare con il MIDI sviluppata da Leslie Sanford, essa vuole essere uno strumento d'aiuto per gli sviluppatori che vogliono creare applicazioni MIDI.

Della libreria a noi interessa particolarmente la gestione dei dati in ingresso, per cui adesso vedremo un esempio di codice e commenteremo le classi principali con le relative proprietà e metodi utilizzati nello sviluppo del software.

```
private Sequence sequence;
private Track track
private InputDevice inputDevice;
private MidiInternalClock midiClock;
private void buttonRecord_Click(object sender, EventArgs e)
{
      sequence = new Sequence();
      sequence.Add(track);
      inputDevice = new InputDevice(0);
      inputDevice.ChannelMessageReceived += HandleChannelMessageReceived
      midiClock = new MidiInternalClock();
      midiClock.Start();
      inputDevice.StartRecording();
}


void HandleChannelMessageReceived(object sender, ChannelMessageEventArgs e)
{
      sequence[0].Insert(midiClock.Ticks, e.Message);
}
private void buttonSave_Click(object sender, EventArgs e)
```

```
{
      inputDevice.StopRecording();
      midiClock.Stop();
      sequence.Save("output.mid");
}
```

Questo codice si occupa di catturare i MIDI channel message (solitamente inviati alla pressione di un tasto) trasmessi da un qualsiasi dispositivo MIDI connesso al computer.

Nel codice sopra è stato creato un pulsante Record alla pressione del quale inizia la registrazione delle informazione MIDI ed è stato creato un pulsante Save per salvare la performance.

Le classi utilizzate nel frammento di codice sono:

InputDevice, si occupa di descrivere un dispositivo in ingresso ovvero le diverse porte MIDI.

- **Eventi**: ChannelMessageReceived, viene sollevato ogniqualvolta viene trasmesso sulla porta un messaggio MIDI Channel Message, nelle performance live questo avviene alla pressione di un tasto sullo strumento. Il metodo chiamato all'occorrenza dell'evento è definito usando un delegate. Vi sono eventi simili anche per gli altri principali messaggi MIDI (System Exclusive, System RealTime, System Common).
- **Metodi**: StartRecording viene usato per iniziare la ricezione dei messaggi MIDI dall' InputDevice scelto; StopRecording ferma la ricezione.

Track, rappressenta la traccia, ovvero un contenitore responsabile di mantenere gli eventi MIDI secondo un ordine stabilito, a questi è affiancato un tick.

- **Metodi**: Insert, permette di inserire gli eventi MIDI nella traccia. Esso ha come parametri un int rappresentante i tick trascorsi dall'evento precedente e un IMidiMessage rappresentante un messaggio MIDI.

Sequence, rappresenta una sequenza MIDI, ovvero una collezione di tracce. Essa fornisce anche le funzionalità per caricare e salvare file MIDI.

- **Proprietà**: Division, rappresenta la risoluzione temporale della sequenza, come

visto precedentemente questa può avere due tipi di valore, o il numero di pulsazione nella nota da un quarto o SMPTE (la presente libreria supporta solo il primo); Format, rappresenta il formato MIDI con cui sono salvati i dati nella sequenza.

- **Metodi**: Add, attraverso questo metodo è possibile aggiungere tracce a una sequenza; Load, permette di caricare un file MIDI; Save, permette di salvare la sequenza come file MIDI.

MidiInternalClock, questo clock genera internamente dei tick MIDI utilizzando come risoluzione il numero di pulsazioni in una nota da un quarto.

- **Proprietà**: Tempo, rappresenta il valore di una nota da un quarto espresso in microsecondi.
- **Metodi**: Started, avvia il clock dall'inizio della sequenza; Stopped, ferma il clock; Continued, fa ripartire il clock dalla posizione corrente nella sequenza; isRunning, indica se il clock è in esecuzione.

Le seguenti classi non sono state usate nel codice di esempio, ma si rivelano fondamentali per la creazione dell'applicazione:

MidiEvent, questa classe contiene il time-stamp e il messaggio MIDI.

- **Proprietà**: DeltaTicks, rappresenta il numero di tick trascorsi dall'evento MIDI precedente, AbsoluteTicks, rappresenta il numero di tick trascorsi dall'inizio del brano. MidiMessage è il messaggio MIDI.

ChannelMessage, come dice il nome stesso rappresenta un messaggio di tipo channel.

- **Proprietà:** Command, rappresenta il comando contenuto nello status byte; Data 1 e Data 2 rappresentano i dati contenuti nei 2 data byte; MidiChannel indica il canale del messaggio.

Capitolo 5

Advanced MIDI to IEEE1599

5.1 L'organizzazione dei dati MIDI in IEEE1599

Il software prevede in ingresso dei dati MIDI e li converte in IEEE1599. Il MIDI è un formato scarso di informazioni e si rende difficile la sua conversione a IEEE1599 che, al contrario, contiene una grande quantità di informazioni. In particolare le informazioni saranno salvate nel layer los che contiene un più alto livello di informazioni simboliche rispetto al layer performance che contiene il MIDI. Informazioni quali le pause, le alterazioni, eccetera, devono essere ricavate da semplici messaggi di pressione e rilascio della nota, inoltre il MIDI si riferisce indistintamente a un B# e C con il numero 60. Per questo l'utente prima d'iniziare la registrazione del brano deve inserire le informazioni di tonalità e chiave.

Come detto in precedenza questo programma si occupa principalmente di trascrivere l'informazione MIDI in informazione logica, per questo i layer che principalmente interessano il programma sono entrambi figli del layer logic e sono lo spine (elemento sempre richiesto in un documento IEEE1599) e il los. Qua sotto analizzeremo gli elementi che compongono il los e lo spine e come questi sono stati trattati.

Spine: event.

los: staff_list, staff, key_signature, time_signature, clef, part, voice_list, voice_item, measure, voice, chord, nothead, pitch, duration.

Vediamo adesso cosa sono questi elementi e come vengono ricavati i dati utili a completarli [3].

- staff_list: rappresenta le accollature e contiene i vari pentagrammi (staff).
- staff: contiene gli elementi key_signature, time_signature, clef. I dati utili a creare questi elementi sono definiti dall'utente all'avvio dell'applicazione. Per semplificazione, poiché è difficile trovare brani dove i vari strumenti suonano in tonalità differenti e seguono indicazioni di tempo diverse, gli elementi key_signature e time_signature sono uguali per tutte le parti. L'unico elemento che cambia a secondo dello strumento è la chiave (clef).
- part: rappresenta, ovviamente, la parte che uno strumento deve eseguire. Essa trova una sua corrispondenza logica nel canale MIDI. Ogni canale rappresenta concettualmente uno strumento musicale: infatti si è deciso qualora ve ne fosse la presenza di assegnare ad ogni parte il nome dello strumento selezionato da un eventuale program change, alternativamente essa assumerà il nome di default part più un numero progressivo.
- voice_list: contiene l'elenco delle voci, i voice_item, in cui è suddivisa una parte. Questa è una conoscenza che non si ha a priori, ma dev'essere calcolata. Si occupa di questo la funzione assignVoice che vedremo più avanti.
- measure: rappresenta la battuta. Il numero di queste, per semplificazione, è stato calcolato come lo stesso per tutte le voci di una parte. In particolare, ciò è reso possibile dall'informazione di tempo (time_signature) fornita dall'utente all'avvio dell'applicazione. Grazie a questa informazione il numero delle battute è calcolato secondo la formula $len/vmd \square 1.$ Dove *len* è uguale al tempo espresso in VTU dell'evento Note Off in fondo alla traccia e *vmd* è la durata di una singola battuta espressa in VTU. Nel caso *len* sia multiplo di *vmd*, la formula può essere ridotta a *len/vmd.*
- voice: per la suddivisione degli eventi note in più voci si faccia riferimento alla funzione assignVoice.
- chord: questo elemento rappresenta l'accordo e contiene gli elementi notehead e duration.
- rest: può essere visto come l'opposto di chord, esso rappresenta una pausa e in quanto tale contiene solo l'elemento duration.

- duration: questa informazione non può essere ricavata direttamente dai dati MIDI, ma dev'essere diversamente calcolata. A proposito si faccia riferimento al paragrafo appositamente dedicato.
- notehead: rappesenta la nota nelle sue principali informazioni quali il pitch e gli eventuali valori (nota legata, nota puntata,...).
- pitch: rappresenta la nota tramite gli attributi actual_accidental (alterazione), step (grado) , e octave (ottava). Questi attributi vengono estratti dal numero della nota MIDI e dalle informazioni di chiave e tonalità. Ma vedremo questo più avanti nella funzione getStepAndOctave.
- event: prima di creare gli elementi event è necessario che le note che hanno durata espressa in VTU vengano convertite come somma di note legate con denominatore potenza di 2, poiché ognuna di queste si riferisce a un nuovo evento.

Da notare che la differenza tra nota legata e nota puntata non viene presa in considerazione dal programma che esclude la seconda possibilità. Infatti una nota dalla durata di 1/4 puntata e uguale a una nota da 1/4 legata a una nota da 1/8.

5.2 La durata della nota

Le informazioni MIDI fondamentali che interessano il programma sono i messaggi MIDI di Note On - Note Off: il primo indica l'attivazione della nota, il secondo la disattivazione. La durata della nota viene calcolata come differenza della posizione temporale dei messaggi di Note On e Note Off che hanno nel primo data byte il medesimo valore numerico, ovvero la stessa frequenza della nota.

Il calcolo di durata della pausa viene calcolato come differenza fra i VTU di ricezione del messaggio Note Off e i VTU di ricezione del successivo Note On, appartenente alla stessa voce.

E' da notare come la durata della nota abbia un valore in VTU, mentre nel los è rappresentata in notazione CWN con simboli che hanno un valore ritmico calcolato in potenze di 2 (ottavi,quarti,...)[7].

L'algoritmo matematico che permette di attuare la conversione fra i valori misurati

in VTU e i valori ritmici usati nella CWN può essere così spiegato, per semplificazione assumiamo che tutte le note siano rappresentabili con VTU intere: il valore di durata misurata in VTU *dvtu* della nota viene approssimato come somma di *n* prodotti dei VTU equivalenti alla nota semibreve *nvtu* per potenze in base 2^k. Come si può evincere la somma non è infinita ma limitata da un valore massimo di *k* dipendente da *mv* ovvero la nota di durata ritmica più piccola che si è scelto di rappresentare. Ogni valore trovato sarà sottratto a *dvtu* e chiamato *pn*. I *pn* corrisponderanno alle durate delle note in cui verrà scomposto *dvtu* e la potenza 2^{-k} verrà salvata nella variabile *res(n).* Poiché la sommatoria è finita il valore di *dvtu* può non coincidere con quello delle somme *pn,* e quindi può rimanere una differenza tra *dvtu* e le sue approssimazioni che chiameremo *err*. Questo può essere considerato come un errore umano di esecuzione o come una limitazione della notazione CWN. Sarà:

- *n* = 0*;*

k = 0*;*

finchè $k \leq log2(mv)$

$p = nvtu \cdot 2^{-k}$

se $p \leq dvtu$ allora $res(n) = 2^{-k}$ e $dvtu = dvtu - p$

$n = n + 1$ e $k = k + 1$

altrimenti $k = k + 1$

usciti dal ciclo si chiama *err=dvtu*

se *dvtu* = 0 allora approssimazione esatta

altrimenti *err* contiene l'errore commesso

$$dvtu = \sum_{n=0}^{nmax} pn \ .$$

Il valore della nota *dvtu* in CWN è rappresentato da *n* note legate il cui valore ritmico è *res(n).*

A esempio vediamo come viene approssimata una nota con $dvtu = 1536$ se $nvtu = 4096$ e $mv = 1/64$. Seguendo l'algoritmo i primi due valori ottenuti per $k = 0$ e $k = 1$ vengono scartati, infatti sia $p = 4096 \cdot 2^0 = 4096VTU$ sia $p = 4096 \cdot 2^1 = 2048$ sono entrambi valori maggiori di *dvtu,* per $k = 2, p = 4096 \cdot 2^{-2} = 1024 < 1536$ allora $dvtu = 1536 - 1024 = 512$ e $res(1) = 2^{-2}$, proseguendo $k = 3, p = 4096 \cdot 2^{-3} = 512 \leq 512$ per cui $res(2) = 2^{-3}$, a questo

punto poiché $dvtu = 512 - 512 = 0$ viene fermato l'algoritmo. Il valore 1536 VTU è stato a questo punto scomposto in notazione CWN come due note legate il cui rispettivo valore ritmico è *res(1)*= 1/4 e *res(2)*= 1/8.

Nel caso sia presente *err* e vi sia quindi un avanzo di VTU si è pensato di applicare un algoritmo di quantizzazione. Questo algoritmo scarta le note inferiori a $nvtu\_mv \square nvtu\_mv/2 \square /2$ e arrotonda al valore di $nvtu\_mv$ i valori superiori, per cui $res\_n = log2\_mv$.

Le VTU *spe* di cui viene spostato l'evento successivo dipende da *err,* nel caso *err* sia arrotondato a *mv spe* è uguale a $nvtu\_mv - err$, altrimenti se *err* non viene arrotondato *spe è* uguale a $-err$. La posizione temporale *tem* dell'evento successivo sarà spostata di *tem+spe* VTU. Se *spe* è un intero positivo l'evento sarà posticipato temporalmente, mentre se *spe* è negativo significherà che l'evento viene anticipato. Per cui all'algoritmo precedente saranno aggiunti i seguenti passi:

- *err = dvtu;*
  se $err \square nvtu\_mv \square nvtu\_mv/2 \square /2$ allora $spe = -err$;
  altrimenti scrivi $res\_n = log2\_mv$ e $spe = nvtu\_mv + err$;
  $tem = tem \square spe.$

Riprendendo l'esempio di prima e cambiando il valore di *dvtu* a 1590 VTU e aggiungendo la posizione temporale assoluta dell'evento successivo *tem=6333VTU,* i passi dell'algoritmo saranno gli stessi analizzati nell'esempio precedente solo che *dvtu* anziché assumere il valore 0 per $k = 3$ e comportare la conclusione dell'algoritmo, assumerà il valore 54 per cui verranno valutati anche i casi per *k*=4,*k*=5,*k*=6. Poiché per ognuno di questi valori *k* il valore di *p* è maggiore di 54 ($k = 4, p = 4 \square 1024 \square 2^{-4} = 256;\ k = 5, p = 4 \square 1024 \square 2^{-5} = 128;\ k = 6, p = 4 \square 1024 \square 2^{-6} = 64;$) e poiché *k*=6 è il livello superiore della sommatoria vi è un avanzo di VTU pari a 54 per cui $err = dvtu = 54$ essendo $err \square 4096/64 \square 2048/64 \square /2, err \square 48$, viene aggiunto la variabile $res\_3 = 2^{-6}$ e $spe = 64 - 54 = 10$. A questo punto l'evento successivo sarà spostato temporalmente avanti, infatti $tem = 6333 \square 10 = 6343$. Il valore 1590 VTU è stato a questo punto scomposto in notazione CWN come tre note legate il cui rispettivo valore ritmico è *res(1)*= 1/4, *res(2)*= 1/8 e *res(3)*= 1/64.

Sebbene il valore del singolo errore sia sempre inferiore alla metà del minimum value, la somma di questi in brani particolarmente lunghi può arrivare a spostare il brano avanti o indietro temporalmente di alcuni secondi (su un brano di 4 minuti in media il brano è posticipato o anticipato di 1 secondo), rendendo, in ogni caso, compatibile la performance eseguita con il CWN.

5.3 L'interfaccia utente

Il programma di trascrizione automatica di performance MIDI in IEEE1599 ha un'interfaccia grafica molto semplice.

Prima di iniziare a suonare o impostare i parametri MIDI sul sintetizzatore quali il program change, si consiglia di avviare l'applicazione e la registrazione della performance.

Avviata l'applicazione l'utente si trova di fronte i sequenti pulsanti: New, Open, Save, Preferences, General, Signature, About, Rec, Pause. Attraverso il pulsante General l'utente può inserire le informazioni di descrizione del brano che verranno scritte nel layer General del documento IEEE1599, il pulsante Signature permette di impostare la tonalità e il tempo in cui verrà suonato il brano e aggiungere il tipo di chiave utilizzato

per ogni diverso canale, il quale corrisponde concettualmente a uno strumento musicale.

Il pulsante Preferences permette di scegliere il device di input, ovvero la porta dalla quale ricevere i messaggi MIDI, la granularità delle VTU corrispondenti al quarto con le quali verranno temporizzati gli eventi nello spine, i BPM e infine,il valore di durata minimo delle note registrabili,espresso in potenza di 2 e notazione CWN, il minimum value. Conclusi i vari settaggi l'utente può iniziare la registrazione del brano MIDI cliccando il pulsante Rec, il pulsante Pause permette di fermare la registrazione. Le informazioni ricevute vengono salvate dal programma che le inizia a elaborare dopo che l'utente clicca sul tasto Save. Nel caso i messaggi MIDI arrivino da canali diversi, il software aggiungerà una traccia per ogni canale perchè le tracce MIDI rappresentano concettualmente parti musicali diverse dello spartito, quindi il software considererà ogni canale come un differente strumento. Accanto ai messaggi MIDI il software memorizza un time-stamp calcolato in VTU, ispirandosi allo SMF. Nel momento in cui viene salvato il file XML, viene salvato anche il file MIDI con lo stesso nome del file XML e questo viene aggiunto nel layer performance. Il pulsante Open permette di caricare un file MIDI già salvato, il pulsante New riavvia semplicemente l'applicazione permettendo di lavorare con un nuovo brano e infine il pulsante About dà informazioni sulla versione del software.

5.4 Dentro il programma

Vediamo in dettaglio come opera il programma. Innanzitutto viene inizializzato il contesto e l'input Device, ovvero la MIDI port da cui vengono ricevuti i messaggi MIDI.

Il software considera solamente i messaggi Channel message ed esclude i System message, poichè solo i primi contengono le informazioni essenziali per attuare la conversione in IEEE1599. All'arrivo del messaggio viene richiamata la funzione HandleChannelMessageReceived: qui viene innanzitutto fermato il timer, nel caso fosse attivato, per non influenzare il tempo da associare ai MIDI message con il tempo che impiega il programma ad eseguire le istruzioni contenute nella funzione. Per prima cosa viene verificato da quale canale arriva il messaggio, se questo proviene da un canale non presente nel dizionario

track_list, viene aggiunta al dizionario una nuova traccia corrispondente al canale in cui verranno salvati tutti i messaggi in arrivo dello stesso canale. Insieme a questi nelle tracce viene salvato un time-stamp espresso in VTU calcolato all'arrivo di ogni nuovo messaggio. Come ultima istruzione della funzione viene avviato o fatto ripartire il timer. Quando l'utente clicca sul tasto Save viene salvata la performance MIDI in SMF e viene richiamata la funzione createXML passandogli come oggetto la track-list e le informazioni di partitura quali la chiave e il tempo, inserite dall'utente .

All'interno di createXML viene chiamata la funzione assignVoice che si occupa di dividere la traccia in più voci, dopo aver assegnato ad ogni evento la voce di appartenenza.

A questo punto gli eventi MIDI vengono passati a splitNote che si occupa di dividere la durata delle note come somme di potenze in base 2.

Per convertire dal MIDI a CWN la frequenza delle note si parte dal fatto che 60 corrisponde al pitch C e ottava 4 , dopo di che si va di 12 in 12 e per sapere il nome si guarda nell'array di nomi delle scale major o minor.

5.4.1 Le funzioni principali

La funzione assignVoice

In una sequenza registrata di eventi MIDI un messaggio di Note On potrebbe immediatamente seguire un altro Note On: in questo caso il programma considera i due messaggi come parte dello stesso accordo se essi e i relativi messaggi di Note Off hanno la stessa posizione temporale. Nel caso uno di questi messaggi abbia una posizione temporale differente, il programma considererà il messaggio come parte di un'altra voce che verrà aggiunta nell documento IEEE1599.

La funzione assignVoice ha la finalità principale di scegliere o creare la voce a cui assegnare il MIDI event Note On in ingresso. Essa permette, inoltre di decidere se un evento Note On è un accordo.

La scelta della voce:

```
for (int i = 0; i < voice_dict.Keys.Count; i++)
{
    int c = voice_dict[i].Count;
    if (voice_dict[i][c - 1].elapsed_vtu > midievent.AbsoluteTicks)
      voice_number++;
    else
      break;
}
```

voice_dict: dizionario con il numero della voce per chiave e EventDetail (classe contenente le informazioni rilevanti del MIDI event) come valore.
event_noteon: evento Note On.
voice_number: variabile inizializzata in assignVoice che rappresenta il numero della voce a cui sarà assegnato il Note On.

Il significato di questa porzione di codice è verificare se l'ultimo evento Note Off per ogni voce a partire dalla prima abbia un tempo di inizio maggiore del Note On in ingresso, nel caso ciò sia verificato voice_number viene incrementato di uno. Il ciclo andrà avanti fin quando la condizione sopracitata non sarà falsa, in questo caso il Note On in ingresso sarà assegnato a una voce esistente.

E' un accordo?

```
if (nextmidievent.AbsoluteTicks - event_noteon.AbsoluteTicks <
                                ((4 * signature.vtu) / 2*signature.min_value)))
{
     MidiEvent endmidievent = getNoteOff(track, event_noteon, eventindex);
     MidiEvent endnextmidievent = getNoteOff(track, nextmidievent, eventindex);
     if (endnextmidievent.AbsoluteTicks - endmidievent.AbsoluteTicks <
                                 (4 * signature.vtu) /  2*signature.min_value))
     {
         jump_midievent.Add(eventindex);
         note_numbers.Add(nextchanmess.Data1);
     }
}
```

nextmidievent: evento MIDI immediatamente successivo ad event note_on, in questa parte di codice è verificato che esso sia un messaggio Note On.
event_index: indice dell'evento nextmidievent.
note_numbers: dizionario che contiene i valori MIDI della frequenza della note.
jump_midievent: dizionario che contiene la posizione nella traccia degli eventi MIDI che per diversi motivi devono essere saltati, è usato anche all'interno della funzione assignVoice ma non è di nostro particolare interesse.

Questa parte di codice, che si trova all'interno di un ciclo do while che continua fin quando nextmidievent non è corrispondente al Note Off relativo a event_noteon, decide se l'evento Note On nextmidievent deve appartenere a una nuove voce o se dev'essere considerato come parte di uno stesso accordo assieme ad event_noteon.

Per prima cosa viene valutato che la differenza fra l'inzio di nextmidimidevent e di event_noteon sia minore della metà del minimum_value (espresso in VTU); in quel caso si vanno a prendere i Note Off dei due eventi richiamando la funzione getNoteOff che scorre gli eventi nella traccia sino a trovare i Note Off relativi ai due eventi: se anche questi hanno come risultato della loro differenza un valore minore della metà del minimum_value (convertito in VTU) allora si considerano i due eventi come parte di uno stesso accordo e per questo a note_numbers viene aggiunto il primo message data del nextmidievent, ovvero il messaggio contenente l'altezza della nota in rappresentazione numerica.

La funzione getStepAndOctave

I messaggi MIDI di Note On e Note Off contengono il pitch e l'ottava espressi in termini numerici della nota attivata/disattivata.

L'operazione di trasformazione delle note dalla forma numerica alla forma stabilita nella CWN viene eseguita dalla funzione GetStepAndOctave che si trova all'interno della classe EventDetail.

```
private void GetStepAndOctave(List<int> note_numbers, int accidental_num)
{
	List<int> octave = new List<int>();
	List<char> step = new List<char>();
	List<string> accidental = new List<string>();
```

```
        for (int i = 0; i < note_numbers.Count; i++)
        {
                int one_octave = 0;
                int notenum = note_numbers[i];
                while (notenum >= 12)
                {
                        notenum -= 12;
                        one_octave++;
                }
                octave.Add(one_octave-1);
                if (accidental_num < 8)
                {
                        switch (notenum)
                        {
                                case 0: step.Add('C');
                                        accidental.Add("normal");
                                        break;
                                case 1: step.Add('C');
                                        accidental.Add("sharp");
                                        break;
                                ..............................
                                ..............................
                                case 11: step.Add('B');
                                        accidental.Add("normal");
                                        break;
                        }
                }
        }
        this.rawnote = new RawNote(octave, step, accidental);
}
```

one_octave: numero dell'ottava.

accidental_num: numero di alterazioni, da 0 a 8 siamo nel caso dei diesis, dai 9 in poi dei bemolle.

note_numbers: dizionario contenente il numero MIDI delle note

step,octave,accidental: dizionari contenenti il grado, l'ottava, e le alterazioni della nota.

Questa funzione si basa sulla conoscenza a priori che la nota MIDI 0 corrisponde al nome C e ottava -1 (in realtà la convenzione dell'ottava varia di volta in volta a seconda del software e hardware utilizzato: Yamaha si rifeisce al C centrale come C3 mentre logica vorrebbe che ci si

riferisse a questa nota con C5, in modo che la nota MIDI dal valore numerico 0 corrisponda a C0; in realtà basandosi su altri lavori svolti su IEEE1599, è stato scelto di adottare la convenzione della Roland che si riferisce al Do centrale come C4)[6]. Sapendo che ogni 12 semitoni il tono della nota si ripete (C corrisponde a 0, 12, 24,etc.), a ogni numero nella lista note_numbers viene sottratto il valore di 12 finquando esso non sia un valore minore di 12. Il blocco if valuta l'accidental_num, in particolare se questo è minore di 8 si entrerà nel primo blocco switch-case dove le alterazioni sono in diesis, altrimenti si entrerà nel secondo blocco switch case che usa le alterazioni in bemolle. In questi blocchi switch-case sono valutati i numeri da 0 a 11, quindi a seconda del numero ottenuto vengono aggiunti alle liste step e accidental il nome della nota e il tipo di alterazione. Ogni volta che viene effettuata la sottrazione viene incrementato one_octave, che viene poi aggiunto alla lista octave decrementato di uno per rispettare la convenzione che il Do centrale corrisponda a C4, perchè altrimenti sarebbe stato uguale a C5. Le informazioni così ottenute (nome della nota, ottava, alterazione) vengono salvate nella classe RawNote. Da notare che questa classe emula perfettamente il concetto di singola nota vista come caso particolare di accordo, infatti in questa classe l'ottava, il grado e l'alterazione sono dizionari che possono contenere uno o più dati.

La funzione divideAtMinimumValue

L'impostazione di VTU dispari associata al quarto presenta evidenti problemi nella scelta dei valori in VTU da assegnare alle note inferiori al quarto poichè le VTU devono essere sempre interi e quindi non possono avere valore decimale. Se ad esempio il valore delle VTU associate al quarto è 55, il valore della nota da un ottavo dovrebbe essere la metà, cioè 22,5. Per risolvere questo problema è stato inizialmente pensato di assegnare alla prima nota da un ottava il valore di 28 VTU e alla seconda 27 VTU in modo che la somma rimanga 55.

In realtà questo approccio crea dei problemi poichè il programma si aspetterebbe una nota da un 1/8 cui assegnare 27 VTU, ma potrebbe invece capitare che le note che seguono siano due note da 1/16. Perciò si è deciso di scrivere il valore totale in VTU della misura come somma dei valori in VTU del minimum_value. Ma vediamo come questo viene fatto dalla divideAtMinimumValue e come questo viene gestito dalla funzione splitNote .

```
public void divideAtMinimumvalue(int vtu_length)
{
      int lim_low=(int)(signature.vtu*4/signature.min_value);
      int lim_high = (int)Math.Ceiling(signature.vtu * 4 /
                                                (double)signature.min_value);
      int a = (int)(vtu_length / 2);
      int b = (int)Math.Ceiling(vtu_length / 2.0);
      if (a != lim_low && b != lim_high)
      {
            divideAtMinimumvalue(b);
            divideAtMinimumvalue(a);
      }
      else
      {
            decomposed_number.Add(b);
            decomposed_number.Add(a);
      }
}
```

decomposed_number: array contenente il valore totale della misura come somma di note dalla minima durata ritmica espresse in valori VTU.
vtu_length: valore VTU da scomporre.
lim_low, lim_high: sono i 2 valori in VTU che può assumere la nota di durata minima.
a,b: valori la cui somma dà vtu_length.

Quando viene chiamata la funzione viene passato come parametro vtu_length la durata dell'intera misura in VTU. Finquando a e b sono diversi viene effettuata una ricorsione della funzione, infine al dizionario decomposed_number vengono aggiunti i valori di a e b.

Supponiamo che la nota di valore minore sia 1/32 e il valore in VTU di 1/4 sia 13 e il tempo sia impostato a 4/4, il dizionario sarà composta dalla seguente sequenza: {2,2,2,1,2,1,2,1;2,2,2,1,2,1,2,1;2,2,2,1,2,1,2,1;2,2,2,1,2,1,2,1}.

Ogniqualvolta un valore dell'array viene utilizzato, il suo valore sarà impostato a 0. Ognuno di questi valori rappresenta una nota da 1/32 perciò una nota da 1/4 sarà composta da 8 valori (1/4=8/32).

Secondo questo algoritmo anche 8 valori di 1 rappresentando ognuna un 1/32 potrebbe rappresentare 1/4. Ciò non è vero poiche un quarto ha un valore di 13 VTU mentre così ne varrebbe solamente 8. Per questo ogni nota deve prendere i valori dell'array da sommare

partendo da una posizione specifica e in ordine sequenziale.

Questa posizione non deve contenere un valore di 0 e può essere un qualunque multiplo, del rapporto fra la nota di durata minima (1/32) e la nota a cui devono essere assegnati i VTU (es. 1/4), purchè non superi la grandezza dell'array. Una volta che tutti i valori dell'array saranno stati utilizzati, significherà che la misura è stata completata e quindi l'array viene reinizializzato con i precedenti valori.

La funzione splitNote e turnsInCWN

Queste due funzioni mettono in pratica l'algoritmo discusso nel paragrafo 5.2 convertendo la singola nota con durata espressa in VTU in più note legate tra di loro con durata espressa in potenze di 2. Le due funzioni permettono inoltre di stabilire in quale misura inserire le note, di esprimerne il denominatore in CWN e di ottenere il tempo assoluto in VTU corretto di un errore di quantizzazione a cui la nota dev'essere suonata.

1)

```
public ArrayList splitNote(double note_duration_value, double elapsed_vtu,
                                    Signature signature, ref int error_value)
{
   double note_measure = 1.0 + elapsed_vtu / signature.vtu_measure_duration;
   ArrayList tied_notes = new ArrayList();
   if (elapsed_vtu + note_duration_value > (signature.vtu_measure_duration *
                                                              (int)note_measure))
   {
       double first_value = (signature.vtu_measure_duration *
                                              (int)note_measure) - elapsed_vtu;
       note_duration_value -= first_value;
       tied_notes.AddRange(turnsInCWN(ref first_value, ref elapsed_vtu,
                                                                  signature));
   }
   if (note_duration_value > signature.vtu_measure_duration)
   {
      note_measure = 1.0 + elapsed_vtu / signature.vtu_measure_duration;
      while (note_duration_value > signature.vtu_measure_duration)
      {
         double temp_value = signature.vtu_measure_duration;
         tied_notes.AddRange(turnsInCWN(ref temp_value, ref elapsed_vtu,
                                                                  signature));
         note_duration_value -= signature.vtu_measure_duration;
      }
```

```
   }
   tied_notes.AddRange(turnsInCWN(ref note_duration_value, ref elapsed_vtu,
                                                                  signature));
   error_value += (int)note_duration_value;
   return tied_notes;
}
```

note_duration_value: rappresenta la durata della nota espressa in VTU

elapsed_vtu: è l'istante di tempo in cui è stata suonata la nota.

In questa funzione viene calcolato se la somma tra elapsed_vtu e note_duration_value sia maggiore dell'inizio della misura successiva. In questo caso viene sottratto da note_duration_value le VTU necessarie a completare la prima misura. Queste vengono passate alla funzione turnsInCWN. Alla fine della funzione l'error_value viene incrementato delle VTU scartate o aggiunte dalla funzione turnsInCWN.

2)

```
public ArrayList turnsInCWN(ref double note_duration_value, ref double
                                                elapsed_vtu, Signature signature)
{
   Note note;
   ArrayList tied_notes = new ArrayList();
   int temp_value = (int)note_duration_value;
   bool quantize = false;
   for (int i = 0; i <= Math.Log(signature.min_value, 2); i++)
   {
      while ((signature.vtu * 4) / (1 << i) <= note_duration_value &&
              (signature.vtu * 4) / (1 << i) <= signature.vtu_measure_duration)
      {
         note = new Note();
         note.vtu_duration_value = 0;
         note.den = 1 << i;
         note.note_measure = (int)(1.0 + elapsed_vtu /
                                                signature.vtu_measure_duration);
         note.timing = (int)elapsed_vtu;
         if (((signature.vtu * 4) / (double)signature.min_value) % 1 != 0)
         {
            int quantity = signature.min_value / note.den;
            int array_position = 0;
            while (temp_decomposed_number[array_position] == 0)
```

```
            {
               array_position += quantity;
            }
            for (int J = array_position; J < array_position + quantity; J++)
            {
               note.vtu_duration_value += temp_decomposed_number[J];
               temp_decomposed_number[J] = 0;
               count_dec++;
            }
            if (count_dec == decomposed_number.Count)
            {
               count_dec = 0;
               temp_decomposed_number.Clear();
               temp_decomposed_number.AddRange(decomposed_number);
            }
         }
         else
         {
            note.vtu_duration_value = (signature.vtu * 4) / note.den;
         }
         elapsed_vtu += note.vtu_duration_value;
         tied_notes.Add(note);
         note_duration_value -= note.vtu_duration_value;
         int vtu_minimum_value=(signature.vtu * 4)/signature.min_value;
         if (note_duration_value > vtu_minimum_value * 3/4.0  &&
                                    note_duration_value < vtu_minimum_value)
         {
            temp_value = (int)note_duration_value - vtu_minimum_value;
            note_duration_value = vtu_minimum_value;
            quantize = true;
          }
      }
   }
   if (quantize)
   {
      note_duration_value = temp_value;
      for (int i = tied_notes.Count - 1; i > 0; i--)
      {
         Note second = (Note)tied_notes[i];
         Note first = (Note)tied_notes[i - 1];
         if (first.den == second.den)
```

```
            {
                first.den /= 2;
                tied_notes.RemoveAt(i);
                tied_notes.RemoveAt(i - 1);
                tied_notes.Insert(i - 1, first);
            }
            else
                break;
        }
    }
    return tied_notes;
}
```

note_duration_value: valore totale della nota in VTU

tied_notes: array che va a contenere le note legate.

note.den: valore ritmico della nota.

quantity: rappresenta il numero di valori dell'array decomposed_number che servono per rappresentare la nota dal valore ritmico note.den.

temp_decomposed_number: array contenente il valore totale della misura come somma di note dalla minima durata ritmica espresse in valori VTU.

L'idea della funzione turnsInCWN è di scomporre il valore dell'intera nota in somma di potenze in base 2. Ogni volta che viene torvato un valore questo viene memorizzato nella classe Note, sottratto al valore totale della nota note_duration_value, e aggiunto al tempo trascorso (elapsed_vtu), il secondo valore verrà calcolato sul nuovo valore della nota e così via. La funzione inizia con un ciclo for che va da 0 (valore ritmico massimo della nota) al logaritmo in base 2 della nota di durata minima (valore ritmico minimo della nota) impostata all'inizio. Prima di entrare nel ciclo while vengono valutate le condizioni che il valore (signature.vtu * 4) / (1 << i) sia minore della durata in VTU della misura e sia minore della durata della nota in VTU; all'interno del ciclo while vengono salvate nella classe Note informazioni come il denominatore in CWN (i<<1), il tempo in VTU al quale l'evento nota avviene, la misura in cui inserire la nota, calcolata come 1+ il rapporto tra il tempo trascorso e la durata della singola misura in VTU, e la durata della nota in VTU essenziale nel caso in cui non tutte le note siano rappresentabili esattamente da VTU interi. A seguito viene valutato se la conversione del minimum value in VTU sia un numero decimale, nel caso lo sia viene calcolato l'int quantity, a questo punto attraverso un ciclo while vengono considerati le posizioni dell'array temp_decomposed_number multiple di quantity, se queste sono diverse da

0 il ciclo si ferma e si prendono i primi quantity valori a partire dalla posizione ottenuta e si assegna a questi il valore di 0. Quando tutti i valori dell'array saranno uguali a 0 significherà che la misura è stata completata, quindi viene reinizializzato l'array coi precedenti valori. Alla fine della funzione viene effettuata l'operazione di quantizzazione descritta al paragrafo 5.2. Quando questa viene effettuata può darsi di avere nell' array due note con lo stesso valore rappresentabili da un'unica nota più grande, ad esempio se minimum_value = 1/64 potrei avere due note da 1/64 che sarebbero meglio rappresentate da un'unica nota da 1/32. L'ultimo blocco if si occupa proprio di questo andando a guardare nell'array se ci sono note dalla stessa durata che nel caso vengono eliminate e sostituite da una nota unica di grandezza doppia.

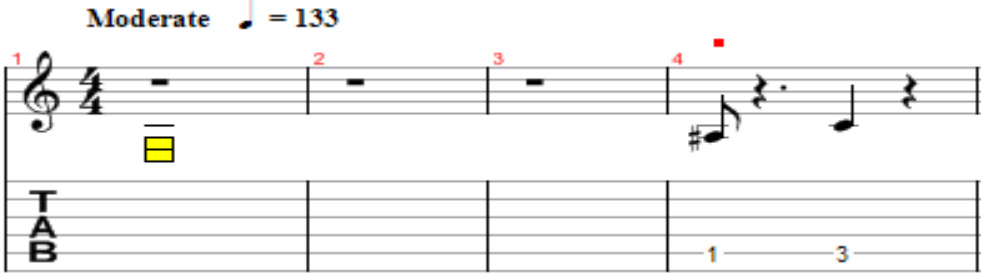
Capitolo 6

Testing

La corretta funzionalità del software è stata verificata inviando dati MIDI sia attraverso il collegamento di un pianoforte Casio Privia PX-310 ad una interfaccia MIDI M-audio esterna al PC, sia attraverso una tastiera MIDI virtuale (Virtual MIDI Keyboard di Wouter van Beek) collegata a una porta MIDI virtuale, la LoopBe1 Internal MIDI port. LoopBe1 è descritta sul sito ufficiale http://nerds.de/en/loopbe1.html come una periferica MIDI interna per trasferire dati MIDI fra programmi del computer. In particolare è stata verificata la corretta suddivisione metrica delle note per qualsiasi tempo e la corrispondenza perfetta fra gli eventi dello spine e gli eventi descritti nel los. Verranno descritti adesso alcuni test che sono stati effettuati per controllare il corretto funzionamento del software sviluppato.

Al programmma sono stati dati in ingresso diversi file MIDI rappresentanti le principali informazioni musicali quali la durata, il tono delle note e il corretto inserimento di quest'ultime all'interno della battuta rispettando la metrica del brano. I dati contenuti nel file XML generato sono stati confrontati con i simboli dello spartito visualizzato dall'editor di partiture GuitarPro5. L'altezza della nota è stata verificata usando un sempilce file MIDI

rappresentante una scala di Do Maggiore. Il suono di più note contemporanee è stata verificato attraverso il caricamento di un file rappresentante sia un accordo sia più voci, in particolare si è potuto notare come il software riesca bene a dicscernere fra le note appartenenti lo stesso accordo e le note riferite a voci diverse. Per la metrica sono stati utilizzati più file MIDI con durate ritmiche differenti. Infine è stato caricato un file MIDI più complesso, la canzone degli Aqua My oh my, che racchiudesse la maggior parte di queste problematiche. Di questa si offre sotto un esempio:




```
- <measure number="1">
  - <voice voice_item_ref="Harpsichord_1voice0">
    - <rest event_ref="Harpsichord_1_voice0_e0">
        <duration num="1" den="1"/>
      </rest>
    </voice>
  </measure>
- <measure number="2">
  - <voice voice_item_ref="Harpsichord_1voice0">
    - <rest event_ref="Harpsichord_1_voice0_e1">
        <duration num="1" den="1"/>
      </rest>
    </voice>
  </measure>
- <measure number="3">
  - <voice voice_item_ref="Harpsichord_1voice0">
    - <rest event_ref="Harpsichord_1_voice0_e2">
        <duration num="1" den="1"/>
      </rest>
    </voice>
  </measure>
```

```
- <measure number="4">
  - <voice voice_item_ref="Harpsichord_1voice0">
    - <chord event_ref="Harpsichord_1_voice0_e3">
      - <notehead staff_ref="staff_1">
          <pitch actual_accidental="sharp" step="A" octave="2"/>
        </notehead>
        <duration num="1" den="8"/>
      </chord>
    - <rest event_ref="Harpsichord_1_voice0_e4">
        <duration num="1" den="4"/>
      </rest>
    - <rest event_ref="Harpsichord_1_voice0_e5">
        <duration num="1" den="8"/>
      </rest>
    - <chord event_ref="Harpsichord_1_voice0_e6">
      - <notehead staff_ref="staff_1">
          <pitch actual_accidental="normal" step="C" octave="3"/>
        </notehead>
        <duration num="1" den="4"/>
      </chord>
    - <rest event_ref="Harpsichord_1_voice0_e7">
        <duration num="1" den="4"/>
      </rest>
    </voice>
</measure>
```

Nell'esempio sopra sono state prese in esame le prime 4 battute del clavicembalo della canzone degli Aqua My Oh My, la figura in alto mostra la lettura del file MIDI da parte dell'editor di partiture Guitar Pro5, sotto vi è l'immagine del file XML generato dal software.

Possiamo notare come le pause semibrevi delle prime tre battute dello spartito siano correttamente trascritte come pause di durata 1 e occupino ognuna un'intera misura. Anche per

la quarta battuta, composta da una nota croma LA# seguita da una pausa semiminima puntata, una nota semiminima DO# e una pausa semiminima, la trascrizione nel file XML risulta corretta, infatti all'interno della misura 4 si può notare come il primo elemento sia un chord con alterazione sharp (diesis), tono A (La) e ottava 2, il secondo elemento sia una pausa di durata 1/4, il terzo elemento sia una pausa di durata 1/8, il quarto sia un chord con alterazione normal (assenza di alterazioni), tono C (Do), ottava 3, seguito dal quinto elemento, una pausa di durata 1/4. In questo caso possiamo notare come le informazioni metriche e l'altezza delle note siano correttamente estrapolate dal file MIDI e trascritte nel documento XML, è necessario, però, evidenziare che la pausa puntata è stata divisa in due elementi distinti poiché il software non ha capacità di distinguere fra nota puntata e nota legata e quindi per semplificazione, poiché il significato semantico è il medesimo,una nota puntata è scritta come due note legate.

Il funzionamento del programma è stato verificato anche effettuando performance live in queste è stato fatto ausilio di un metronomo, per limitare gli errori umani spesso presenti ed evidenziati da piccole pause, cercando di suonare il brano secondo una scansione temporale precisa in modo da poter fare un confronto corretto con le informazioni ottenuto dal software e trascritte nel file XML. In questo caso sono stati suonati spartiti molto semplici, spesso scale, ed anche in questo caso le informazioni riportate dal software si sono rilevate corrette.

I test effettuati nelle esecuzione live hanno riguardato le stesse informazioni musicali testate attraverso il caricamento di file MIDI ma è stata posta particolare attenzione alle voci multiple e agli accordi per verificare che il software distinguesse correttamente i due tipi di eventi musicali. Come per le altre problematiche il software ha risposto positivamente mostrando una buona capacità di identificare correttamente quando le note appartenevano allo stesso accordo e quando queste appatenevano a voci separate.

Capitolo 7

Conclusioni

Questa tesi ha preso spunto dall'auspicio contenuto nel lavoro di Baldan[8], affinchè lo standard IEEE1599 possa trovare una più ampia diffusione visti i vantaggi che questo comporta.

Il lavoro di Baldan ha portato allo sviluppo di un plugin in grado di operare all' interno di Pure Data mentre con questo abbiamo sviluppato un programma autonomo che può operare in ambiente Windows. Questa è una caratteristica che può permettere una più ampia diffusione del formato IEEE1599 e fruibilità specialmente a chi ignora o non utilizza l'ambiente di programmazione Pure Data, quindi anche ad un musicista con semplici conoscenze di base di informatica. In ambiente Windows questo programma si è dimostrato in grado di ottenere da una performance MIDI le informazioni che costituiscono il layer logic, e in particolare i substrati dello spine e del los. Il programma permette inoltre all'utente di fornire le informazioni che andranno a completare il layer general, di salvare l'esecuzione in un file MIDI e inserire questa e i relativi dati all'interno del layer performance. Alcuni sviluppi futuri

potrebbero riguardare la creazione di tablature chitarristiche dall'informazione contenuta nel layer los o di altri svariati applicativi utili al musicista.

Bibliografia

[1] Luca A. Ludovico, An XML Multi-layer Framework For Music Information Description, Phd Thesis, Università degli studi di Milano, 2005.

[2] Ludovico, L. A.: Key concepts of the IEEE 1599 Standard. In: Baggi, D., Haus, G. (eds.) Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects. pp. 15–26. IEEE CS, Lugano (2008)

[3] IEEE Computer Society (2008), IEEE Recommended Practice for Defining a Commonly Acceptable Musical Application Using XML.

[4] Robert Guerin, MIDI Power!: The Comprehensive Guide 2nd edition. Ed. Course Technology Cengage Learning, 2008.

[5] Christian Nagel, Bill Evjen, Jay Glinn, Karli Watson, Morgan Skinner, Professional C# 4 and .NET 4. Ed. Wiley Publishing, 2010.

[6] Matthew MacDonald, Pro WPF in C# : Windows Presentation Foundation in .NET 4.0. Ed. Apress, 2010.

[7] Baldan, S., Ludovico, L.A., Mauro, D.A.: Algorithms for an Automatic Transcription of Live Music Performances into Symbolic Format. In: Aramaki, M., Jensen, K., Kronland-Martinet, R.,

Ystad, S. (eds.) Auditory Display. Genesis of Meaning in Sound and Music. 6th International Symposium, CMMR/ICAD 2009, Copenhagen, Denmark, May 18-22, 2009 : revised papers. LNCS, vol. 5954, pp. 422–438. Springer-Verlag, Heidelberg / Berlin (2010)

[8] Stefano Baldan (2007), PureMX: Codifica real-time di una performance MIDI in formato MX. Tesi di laurea Triennale, Università degli Studi di Milano.

[9] The Complete MIDI 1.0 Detailed Specification version 96.1 Second Edition. Publisher: MIDI Manufacturer's Association, La Habra, CA. November 2001