Università degli studi di Milano
Facoltà di scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione
Corso di Laurea in Informatica


UNIVERSITÀ DEGLI STUDI
DI MILANO

# MODELLI DI RICONOSCIMENTO AUTOMATICO DELL'INTERPRETAZIONE MUSICALE A PARTIRE DAL MIDI

Relatore: Prof. Goffredo HAUS
Correlatore: Dott. Ing. Luca Andrea LUDOVICO

Tesi di laurea di:
Rosario Squillante
Matricola: 553511

Anno Accademico 2006-2007

*Alla mia famiglia*

# Indice

## Cap. 1 – Introduzione



## Cap. 2 – Teoria Musicale



## Cap. 3 XML e MX

## Cap. 4 – Il MIDI

## Cap. 5 – MIDI2MX



## Appendice



## Bibliografia

# Capitolo 1

# INTRODUZIONE


*Nel 1769 il tredicenne Johannes Chrisostomus Wolfgang Theophilus Mozart si reca colla famiglia a Roma in occasione della settimana santa per ascoltare, nella cappella Sistina, l'esecuzione del Miserere di Gregorio Allegri, carica di elementi suggestivi.*
*La magia dell'atmosfera della cappella Sistina spiega perché il Miserere non possa essere eseguito che in quella sacra cornice. Ma c'è un'ulteriore motivo: tempo addietro, l'imperatore d'Austria Leopoldo I chiese e ottenne dal pontefice (che doveva essere Innocenzo XII o Clemente XI) una copia di quella musica, con l'intento di farla eseguire a Vienna.*
*L'esito fu tremendo: piatto, scialbo, privo di colore e di suggestione, il celebre Miserere sembrò una composizione da due soldi e destò nei presenti il dubbio che il pontificio maestro di cappella, attraversato da un impeto di gelosia, avesse mandato a Vienna un brano diverso da quello richiesto.*

*Le proteste della corte indispettirono il papa: dapprima licenziò in tronco il maestro di cappella; in seguito, dopo un lungo lavoro diplomatico, si convinse di quanto il malcapitato maestro andava affermando. E cioè che il Miserere non può eseguirsi a regola d'arte che nella cappella Sistina, a meno di ridurlo ad una musica qualunque. E fu riassunto.*
*Al termine della prima esecuzione, rapito dalla suggestione del momento, Wolfgang si reca nella sua camera d'albergo e riscrive l'intera melodia. Alla replica, ne segue la linea tenendo nascosto il manoscritto nel suo cappello, e ne corregge alcuni punti. Malgrado il tentativo del piccolo di tenere nascosto questo suo lavoro, i romani ne vengono a conoscenza e, credendo di andare incontro ad una serata divertente, chiedono che Mozart canti in concerto quel Miserere.*

*Accontentati, devono ricredersi: l'esecuzione è splendida e suggestiva. Si racconta che: "Il castrato Cristofori che lo aveva cantato nella cappella Sistina, e che era presente, con la sua stupefazione completò il trionfo di Mozart (...) e nulla è più difficile, in fatto di belle arti, che suscitare stupore a Roma".* [1]

Abbiamo aperto con questo episodio leggendario per mettere in risalto una eccezionale dote che il grande compositore austriaco possedeva. La capacità di ricreare, con il semplice ascolto, le linee guida di una melodia con i suoi tempi ed i suoi ritmi. In breve, a riscriverne mentalmente la partitura.
Poter ricreare tale dote in un computer in modo da ottenere il procedimento in modo automatico è uno degli obiettivi dell'informatica musicale. Un piccolo contributo a questo progetto viene dato in questa tesi, che cerca di coprire uno dei passaggi inerenti al nostro caso.

Un musicista in fase di componimento tiene traccia di ciò che elabora allo strumento (quale che esso sia) appuntando sul pentagramma le note eseguite. A queste assegna un valore indicativo, cioè la durata delle note in senso relativo.
Anche se non esistono line guida generali sulle metodologie e le fasi del componimento musicale è però abbastanza naturale pensare che le note e la loro altezza siano le prime cose ad essere trascritte.
Mettiamo che aggiunga una chiave, che è un simbolo che indica all'esecutore l'esatta denominazione e posizione della nota sullo strumento, quindi il tempo. Infine regola le

dinamiche (la forza con cui vanno accentati o meno i suoni prodotti) e l'agogica (commenti indicativi sull'approccio del pezzo, es. Allegro piuttosto che Andante)
A questo punto consegna il manoscritto ancora incompleto ad un musicista, chiedendogli di eseguirlo, senza che questi lo abbia sentito comporre.
L'esecuzione sarà molto diversa da quella concepita e pensata dall'autore.
Tralasciando i dettagli formali della descrizione musicale tradizionale che saranno trattati più avanti, enunciamo un primo concetto.

*Una partitura descrive cosa e come suonare ma in modo tale che sia però l'esecutore a doverne dare una propria lettura secondo la sua arte e la sua sensibilità.*

In effetti riproporre un brano musicale esattamente come è stato concepito la prima volta è virtualmente impossibile.
E'chiaro che ciò non è affatto un limite, anzi. Classici immortali della musica restano sempre vivi ed attuali proprio grazie alla capacità di saperli rileggere ogni volta con spirito nuovo, ammodernandoli o reinterpretandoli.
Una partitura musicale non è diversa da una poesia in versi, da un copione teatrale o da una sceneggiatura.
Abbiamo le parole, la punteggiatura e le descrizioni ma sta alla bravura di chi decanta o recita dare vita a ciò che è semplice inchiostro su carta. Le parole di Amleto son le stesse, ma quanti Amleto ci sono e ci saranno? Quanti Dante erranti nell'Inferno? Ognuno diverso, non tutti sempre all'altezza, qualcuno meglio di altri.
Chopin, Charlie Parker e Gershwin non sono dissimili. Ci sarà sempre qualcuno che raccoglierà i loro scritti e li leggerà attraverso la propria anima.

Detto questo abbiamo capito come l'interpretazione, che è propria dell'uomo, sia parte fondamentale dell'esecuzione musicale (attenzione non composizione ma semplice esecuzione).
Quindi, dire ad una macchina (un computer) cosa suonare è un'operazione diversa dallo scrivere una partitura. Una macchina non può interpretare con quanta grazia o quanta sofferenza suonare una nota semplicemente perché ad oggi, non ha né sensibilità né anima.
La macchina ha bisogno di notizie esatte e precise e numericamente limitata in certi range di valori.

Se abbiamo modo di fornirgliele, la macchina suona esattamente come noi vogliamo, grazia e leggiadria incluse. Però è evidente che i due linguaggi (quello musicale tradizionale, e quello della computer music sono diametralmente opposti. Potremmo dire che nel primo c'è un rapporto da uno a molti, e nel secondo, da uno a uno.
Quindi se da una partitura di Chopin si possono ricavare decine se non centinaia di esecuzioni sensibilmente diverse fra loro, da una *partitura* MIDI[1] dello stesso pezzo ne otteniamo solo una. E dovremmo cambiare le informazioni presento nel file MIDI per ottenerne un'altra anche solo leggermente diversa.

Adesso proviamo a pensare al problema inverso.
Abbiamo l'esecuzione di un pezzo, digitalizzata attraverso un formato musicale che risponde quindi ai criteri di cui sopra. E' possibile risalire ad una partitura di tipo tradizionale? Cioè è possibile ricostruire attraverso un'esecuzione (una delle tante) la partitura generatrice di quella esecuzione? [2]
Suggestivamente il problema può essere paragonato all'ottenere la primitiva di una funzione matematica data.
Ovviamente le problematiche sono altre ma la suggestione rimane.

Il lavoro di tesi seguente mira a 3 obiettivi principali.

1. **Interpretare i comandi MIDI, relativi all'esecuzione di un pezzo, per ottenere informazione musicale *tradizionale*, relativa quindi alla scrittura compositiva.**

2. **Scrivere un programma che generi automaticamente una struttura dati equivalente alle informazioni musicali presenti in partitura (l'MX)**

3. **Rendere tale programma scalabile, modulabile e il più possibile indipendente dal *parser*, il programma che analizza l'input. Nel nostro caso un parser MIDI[2], in futuro qualsiasi altro formato musicale.**

[1] *Abbiamo detto MIDI ma ovviamente esistono diversi formati musicali ognuno con i propri criteri.*
[2] Vi possono essere anche parser diversi per lo stesso formato. Per il lavoro di tesi, ne useremo solo uno

# Capitolo 2

# TEORIA DELLA MUSICA

Prima di addentrarci nei discorsi tecnici è bene fare il punto su uno dei soggetti principali di studio.
E' chiaro infatti che per ottenere una completa formalizzazione informatica della notazione musicale, dobbiamo in primis conoscere tutte le implicazioni che quest'ultima genera nella sua lettura "classica".
In pratica cerchiamo di capire cosa dicono le note a noi esseri umani. In seguito, lo spiegheremo all'elaboratore.

## *2.1 - Cenni storici [3] [4] [5]*

### *2.1.1 - le origini*

La teoria musicale occidentale, risalente ai greci antichi, ha attinto fortemente dai popoli egiziani e della Mesopotamia: questi conoscevano già gli intervalli consonanti di quinta, quarta ed ottava, e ne facevano il punto di partenza di diversi sistemi di scale. Venne elaborato un sistema di relazioni tra altezza delle note e lunghezza delle corde (o dei flauti) necessaria per produrre tali note, a prescindere dalle relazioni matematiche alla base della produzione del suono che poi sarebbero state codificate da Pitagora, che studiò in Egitto ed anche in Mesopotamia.
Era quello un periodo in cui la visione del mondo era comunque fortemente condizionata da superstizioni e religioni: i mesopotamici, ad esempio, adoravano i pianeti e ritenevano che l'armonia tra uomo ed universo fosse regolata dai numeri e si rispecchiasse proprio nella musica.

Nella Grecia antica tra l'VIII ed il VII secolo a.C. coesistettero tre tendenze musicali. Gli *aedi*, o rapsodi, professionisti che cantavano le gesta degli eroi e degli dei accompagnandosi con il *kitharis*, una lira di grandi dimensioni. Successivamente gli aedi si interessarono a temi di attualità o popolari.
Nelle campagne la musica e la danza avevano come protagonista principale la *syrinx*, ossia il flauto di Pan.
Infine, il canto corale accompagnava le cerimonie religiose e quelle civili in generale.

Tra il VI ed il V secolo a.C. il teatro classico raccolse la tradizione della lirica, con autori del calibro di Eschilo, Euripide ed Aristofane. Il coro che accompagnava queste opere era rigorosamente all'unisono, accompagnato eventualmente dalla lira o dall'*aulos* (una specie di flauto doppio): esso faceva da commento alla rappresentazione, ma eseguiva anche la danza, detta *orchesis*, stando nello spazio davanti la scena (che per questo venne detto orchestra). Il termine assunse l'odierno significato durante le prime esecuzioni di opere italiane, quando i musicisti sedevano davanti al palcoscenico.

La musica (*mousike* = cultura dell'intelletto) non era concepita dai greci come attività indipendente, ma come nucleo principale dell'educazione, assieme alla cultura fisica: lo stesso Platone ne sottolineò l'importanza educativa.
Le scale erano basate su gruppi di quattro note di intonazione discendente, detti *tetracordi* (quattro, come le corde delle prime lire). La nota iniziale e quella finale di ogni tetracordo formavano un intervallo di quarta perfetto: le note interne potevano essere alterate per formare diversi tetracordi (genera). Le scale di sette note, o *harmoniai*, erano formate collegando tra loro due tetracordi ed estese per coprire due ottave. Tra queste era poi possibile scegliere scale d'ottave diverse dette modi: il modo da un do all'altro, ad esempio, corrispondeva all'attuale scala di do maggiore.
Pitagora trovò le relazioni numeriche tra la frequenza dei suoni e la lunghezza di una corda, includendo tutto questo nella sua cosmologia: questo pensiero influì poi profondamente nella cultura occidentale.

Anche nell'antica Roma la musica ebbe una importante funzione, soprattutto quale accompagnamento nelle feste religiose. I Romani non ebbero uno stile musicale proprio, ma seppero piuttosto adattare, fondere e sviluppare gli stili delle diverse civiltà con le quali venivano a contatto. La musica fu però utilizzata dai Romani per rallegrare riunioni e

intrattenimenti familiari, oppure per accompagnare le evoluzioni dei commedianti o per allietare i sontuosi festini dei patrizi.
In ogni caso, non restano tracce di questa musica, proprio per la mancanza di una notazione musicale.

Conclusosi il millennio musicale greco-romano, i canti e gli strumenti musicali dell'antichità scomparvero o caddero nell'oblio, lasciando posto ad una musica profana, figlia del decadimento, e ad una musica religiosa, quella cristiana, emergente.
L'origine dei canti religiosi più antichi, di grande semplicità, non risale però nella musica greca, ma nella recitazione cadenzata usata dagli Ebrei quando leggevano i passi delle Sacre Scritture nelle sinagoghe. Ogni parola era divisa nelle sue sillabe ed ogni sillaba era cantata su di una nota, che in genere era la stessa per tutte; era questa la cosiddetta "salmodia ", di cui si può avere esempio nelle attuali litanie.
In seguito, a questa recitazione sillabica, severa e monotona, si aggiunsero altri canti il cui testo non era più tratto dalla Bibbia, ma appositamente composto: i cosiddetti *inni*.

Gli inni si svolsero sempre più ricchi finché la musica cominciò a prevalere sulle parole: a tratti, come dice Sant'Agostino, la voce del cantore si staccava dal testo per abbandonarsi a vocalizzi senza parole, quasi gridi gioiosi ed esultanti che si modulavano per esprimere ciò che era inesprimibile con le parole. Di qui ebbero origine i *canti allelujatici* nei quali il testo era costituito dalle sole quattro sillabe della parola *alleluja* sulle quali si spiegava il canto.

Il pontefice Gregorio I detto *Magno*, negli ultimi anni del VI sec. e nei primi del VII sec., operò grandi sforzi per rivedere e raccogliere molti di questi canti cristiani; il suo *Antiphonarium* (l'antifona è il breve testo delle Scritture che veniva cantato nelle liturgie del IV sec.) è la prima raccolta scritta che ha permesso agli studiosi di comprendere molte caratteristiche della musica dell'epoca, che viene appunto detta gregoriana dal nome di questo papa. Sappiamo che era prettamente vocale, ossia cantata senza accompagnamento musicale, e *monodica*, ossia cantata ad una sola voce. Con ciò non si vuol dire che ci fosse un solo cantore: i cantori potevano anche essere molti ma tutti cantavano esattamente lo stesso motivo musicale seguendo un'unica linea melodica; prendeva forma così quella che tutt'ora conosciamo come *musica a cappella*.

E' al Papa Gregorio Magno che dobbiamo l'istituzione della *Schola Cantorum* per la diffusione delle melodie e il formarsi della notazione musicale.

Intorno al IX sec., alla melodia gregoriana originale si iniziò a sovrapporre un'altra linea melodica, normalmente improvvisata, che procedeva parallelamente alla prima.
Era l'inizio di una pratica musicale che nei secoli a venire avrebbe determinato uno sviluppo impensato nel nostro linguaggio musicale: la *polifonia*.

Si ha polifonia (molti suoni) quando due o più voci cantano o suonano contemporaneamente melodie diverse.
La necessità di sovrapporre con precisione le note di più melodie favorì lo sviluppo della scrittura musicale detta *mensurale*, cioè misurabile nel tempo. Utilizzando le figure musicali si stabilivano con precisione i rapporti di durata tra una nota e l'altra.

I Greci indicavano le note per mezzo di lettere e questo sistema fu a lungo in vita presso i Bizantini; al contrario, in Occidente, la notazione andò in disuso tanto che nel VI sec. non esisteva più un sistema sicuro e da tutti adottato per scrivere la musica. Vi erano solo dei segni speciali, chiamati *neumi*, che probabilmente deriva da un termine greco che significa "respiro" e forse stavano ad indicare, con le loro forme, l'ascendere o il discendere della voce, o il numero di note da cantarsi su di una sillaba.

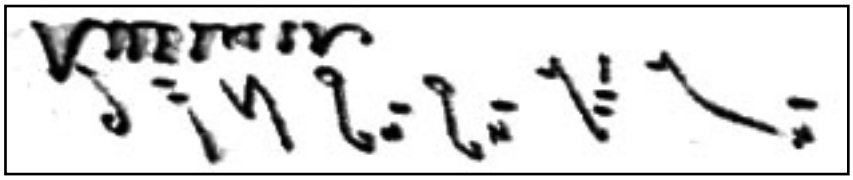


**Figura 1 - neumi**

Questo elementare sistema di notazione si andò evolvendo fino a che il monaco Guido d'Arezzo (IX sec. d.C.) creò il *tetragramma* ( quattro linee e tre spazi),
che introduceva il primo riferimento per rappresentare l'altezza precisa dei suoni. Una linea rossa che indicava la nota Fa, una seconda linea, di colore giallo per la nota Do, poi una terza, fino ad arrivare ad un rigo musicale di quattro linee. Il tetragramma appunto.
Inoltre, derivò il nome delle sei note che lo componevano dalle prime sillabe dei versetti di un conosciutissimo inno a San Giovanni:

" ***Ut*** *queant laxis* ***Re****sonare fibris* ***Mi****ra gestorum* ***Fa****muli tuorum,* ***Sol****ve polluti* ***La****bii reatum, Sancte Joannes* "[1]

Questo inno presentava poi una particolare caratteristica: la nota corrispondente alla prima sillaba di ogni verso saliva di un grado, così da formare una scala di sei note chiamata *esacordo* .

Era finalmente nata la scala moderna, allora di sei note ( Ut, Re, Mi, Fa, Sol, La ) , il principio base per lo sviluppo della futura scala musicale.

### *2.1.2 – L'Ars Nova*

Con il termine *ars nova* si suole designare la musica polifonica del '300 in Francia e in Italia. Esso comparve per la prima volta nel 1320 come titolo di un trattato del teorico, poeta e compositore parigino *Philippe de Vitry* (1291-1361), a quell'epoca non faceva riferimento a novità di carattere estetico o ideale, come più volte si è pensato nei tempi moderni, ma, in conformità alla concezione medievale dell'*ars*, indicava semplicemente le innovazioni che si andavano introducendo nella tecnica compositiva, soprattutto per quanto riguardava la notazione censurale.

Le novità tecniche di rilievo dell'ars nova, esposte nel 1319 dal matematico dell'Università di Parigi *Johannes de Muris* (1298-1350) e perfezionate l'anno successivo nell'Ars Nova di Philippe de Vitry, consistono nel riconoscimento, accanto al metro ternario di derivazione modale e franconiana, anche del metro binario, cioè del metro in presenza del quale non potrà aversi un valore perfetto, ma solo un valore imperfetto.
Se nell' *Ars Antiqua* Francone da Colonia determinava la durata dei suoni con le figure: *longa* ( tre brevis), *brevis* (tre semibrevis), e *semibrevis*, grazie a de Vitry, a de Muris e ad altri illustri teorici francesi, nasce l'innovativo sistema di *prolazione* che include al tempo ed alle suddivisioni ternarie (tanto care al clero di quel periodo, se si pensa alla simbologia

[1] *Trad.: "Affinchè i fedeli possano cantare con tutto lo slancio le tue gesta meravigliose, liberali dal peccato che ha contaminato il loro labbro, San Giovanni."*

cattolica del numero tre come rappresentazione di perfezione e Trinità) il tempo e le suddivisioni binarie.

Inoltre, i principi del mensuralismo franconiano vengono estesi a una scala di figure (ossia di forme grafiche rappresentanti determinate durate) assai più ampia: al di sopra della longa si aggiunge la *maxima*, che, a differenza della *duplex longa*, può esistere anche come maxima perfetta (di tre longae); trattamento analogo a quello accordato alle altre figure riceve la semibreve, ragion per cui anche la brevis può essere perfetta o imperfetta a seconda che contenga due o tre semibrevi; al di sotto della semibreve, sta la minima.

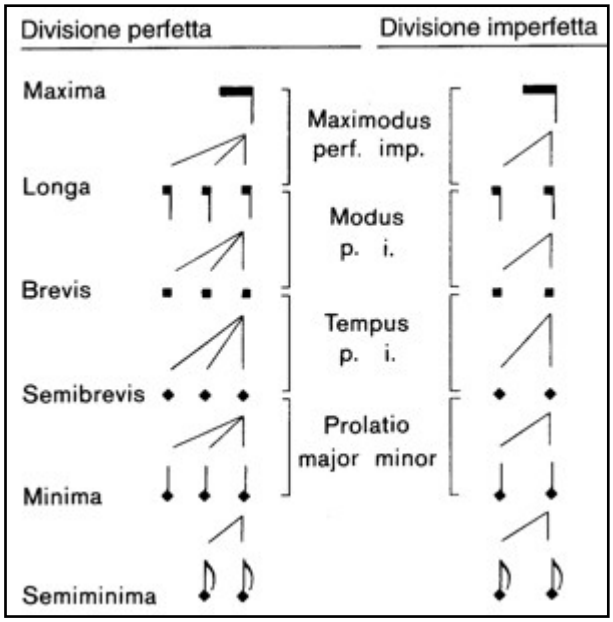



**Figura 2 - metro binario e ternario**

La possibilità di avere sia il metro binario sia quello ternario, poi, consente di non presentare lo stesso metro lungo tutta la scala di valori, nel senso che un determinato pezzo di musica può contemplare, ad esempio, una ternarietà nel rapporto fra longa e brevis, una binarietà nel rapporto fra brevis e semibreve e una ternarietà nel rapporto fra semibreve e minima. *Maximodus* si chiamava il rapporto, ternario o binario, fra maxima e longa: *perfetto* se ternario, imperfetto se binario; *modus* era il rapporto fra longa e brevis, l'unico conosciuto dall'ars antiqua (infatti il termine modus si richiamava ai modi ritmici); *tempus* il rapporto fra brevis e semibreve; *prolatio* la relazione fra semibreve e minima, chiamata *maggiore* se ternaria *minore* se binaria.

Philippe de Vitry escogitò inoltre l'uso dell'inchiostro rosso al posto di quello nero per le note presenti in due al posto di tre, o in tre al posto di due, come le *duine* e le *terzine* degli odierni *gruppi irregolari*.

Le composizioni rimaste di Philippe de Vitry consistono solo in *mottetti*[2].
Vitry non proseguì lungo la strada additata da Petrus de Cruce, della prevalenza della voce superiore, preferendo invece far dialogare il *motetus* e il *triplum* in modo paritario: al di sotto stavano una o due voci con carattere di *tenor* (il secondo tenor, quando c'era, era chiamato *contratenor*), che approfittavano dell'ampia scala di valori e di rapporti dell'ars nova per adoperare note più lunghe di quelle delle voci superiori (ad esempio, se queste ultime erano scritte con semibrevi e minime, e di conseguenza il loro rapporto era costituito dalla prolatio – maggiore o minore – il tenor e il contratenor usavano longae e breves, e quindi per loro era rilevante il modus – perfetto o imperfetto); in tal modo, il metro e il ritmo della composizione erano come stratificati in più piani diversi e contrastanti.

Tipica di questi mottetti è *l'isoritmia*, cioè un certo tipo di organizzazione del ritmo e della melodia del tenor, secondo cui da un lato i valori delle note erano disposti ripetendo sempre una certa successione di valori, detta *tàlea*, e dall'altro la melodia, non molto lunga, veniva ripetuta più volte; ogni ripetizione della melodia era detta *color*, e l'ultimo color spesso si presentava con i valori diminuiti secondo una certa proporzione, ad esempio dimezzati; costruito con criteri di matematica razionalità, il tenor era l'impalcatura che reggeva tutta la composizione e prevedeva un'esecuzione strumentale.

### *2.1.3 – dal rinascimento all'era contemporanea*

A partire da questo periodo la semiografia musicale comincia ad evolversi verso i parametri attuali.
La Scuola franco fiamminga innovò grandemente le preesistenti forme della messa, del mottetto e della chanson. Ponendo le consonanze per terze (ancora oggi familiari all'orecchio occidentale) e la forma imitativa del canone alla base delle loro procedure compositive, i fiamminghi rivoluzionarono la pratica della polifonia ereditata dall' Ars nova e dall'Ars

[2] Anticamente per mottetto si intendeva una forma di canto sacro che consisteva nell'aggiungere due voci (motetus e triplum) al tenor del gregoriano. Verso la metà del 1200 il mottetto cambiò radicalmente stile, apponendo al motetus ed al triplum testi profani che giocavano su un tenor spesso eseguito con uno strumento. Nel 1300 la sua struttura raggiunse la perfezione con l'uso dell'isoritmia.

antiqua. Il lavoro di questi compositori poneva le basi per lo sviluppo di quella che sarebbe stata la teoria dell' *armonia*. Il XVI secolo vide anche il verificarsi di uno degli eventi più significativi per la diffusione della musica: la nascita dell'editoria musicale.

La musica occidentale si sviluppò con straordinaria rapidità attraverso i secoli successivi, anche perfezionando il suo sistema tonale: una pietra miliare è costituita dalle composizioni di Johann Sebastian Bach del Clavicembalo ben temperato (I libro 1722, II libro 1744) che introducono il cosiddetto *temperamento equabile*, adottato universalmente da allora in poi fino ai giorni nostri, in cui tutti i dodici semitoni assumono eguale distanza relativa all'interno di una ottava e pari alla radice dodicesima di due.

Nel secolo d'oro della musica classica occidentale, gli anni che vanno dal 1750 al 1850, essa si esprime in forme sempre più ricche ed elaborate, sia in campo strumentale (uno straordinario sviluppo ebbe la forma della *sinfonia*) che in campo operistico, sfruttando sempre più estesamente le possibilità espressive fornite dal sistema armonico e tonale costruito nei secoli passati.

In seguito alla crisi del sistema tonale, a cavallo tra Ottocento e Novecento si avvia una frenetica ricerca di nuovi codici linguistici su cui basare la composizione musicale. Le soluzioni proposte sono diverse: dal ritorno alla modalità, all'adozione di nuove scale, di derivazione extrauropea, come quella per toni interi (proposta per primo da Claude Debussy), al cromatismo atonale e poi dodecafonico che tende a scardinare la tradizionale dualità di consonanza/dissonanza.

In particolare, nel secondo decennio Arnold Schönberg, assieme ai suoi allievi giunge a delineare un nuovo sistema, noto come "dodecafonia", basato su serie di 12 note. Alcuni ritennero questo l'inizio della musica contemporanea, spesso identificata con la musica d'avanguardia.

## *2.2 – La notazione musicale [6] [7] [8] [9]*

Che cos'è la partitura? Quali informazioni veicola? Come si legge?

Ci porremo ora al centro del problema. Interpretare cioè un testo musicale scritto, per poterlo poi formalizzare dal punto di vista informatico attraverso il linguaggio XML. Ciò è indispensabile per il lavoro che ci accingiamo a svolgere.

Obiettivo principe infatti, è la ricostruzione delle informazioni presenti in partitura. Una ricostruzione che parte dall'esecuzione musicale.

Per capire esattamente qual è la problematica, citiamo qui un estratto di un articolo del dott. Daniele Barbieri [6] relativo all'interpretazione degli ipertesti. Esso ci da una chiara idea della natura del problema dal punto di vista della semiotica e del linguaggio.

*"[…] Dobbiamo infatti precisare che il linguaggio musicale si esprime in due forme ben distinte. Sotto forma di partitura scritta su carta e sotto forma di esecuzione musicale. La partitura è poco o per nulla nota ai più, e la maggior parte delle persone non possiede nemmeno i codici per interpretarla; ma è tutto quello che abbiamo ricevuto dall'autore del testo musicale, ed è dunque la manifestazione testuale più autentica. Eppure, con poche eccezioni, la partitura musicale è solo uno strumento per la generazione del testo musicale vero e proprio, quello che si produce attraverso l'esecuzione. Ma l'esecuzione richiede un secondo autore, il pianista, violinista o direttore d'orchestra che è stato prima lettore della partitura, e sulla base di quello ha deciso come eseguire la musica. Nel farlo, l'esecutore è costretto a eseguire una serie di scelte, realizzando una delle infinite esecuzioni potenziali diverse che qualsiasi partitura musicale potrebbe generare. Per analizzare un brano musicale e cercare di scavare nelle profondità della sua struttura, la forma testuale da utilizzare sarebbe dunque certamente quella scritta, la partitura, perché solo lì l'istanza dell'autore emerge senza la mediazione dell'esecutore. Ma supponiamo di non avere accesso alla partitura, perché non ce la possiamo procurare, oppure semplicemente perché non sappiamo leggere la musica. Questo ci impedirà forse una lettura analitica del brano? In linea di massima no, rendendocela tuttavia in ogni caso più problematica, perché ci troveremo nella situazione di dover ricostruire la struttura del testo originale attraverso l'interpretazione dell'esecutore. […]"*

Teniamo bene a mente queste parole, le quali ci danno un'ottima visuale della questione musicale, musica suonata e musica scritta. L'esecuzione di un brano, una volta eseguita, è unica. Una sola partitura, seppur unica, da' origine ad esecuzioni ogni volta diverse. Tralasciamo in questa sezione il problema tecnico informatico che ci porterà al risultato finale. Ci concentreremo infatti, su quella *struttura del testo* da dover ricostruire.

Premettiamo che alcune delle sezioni seguenti, cercheranno di spiegare al lettore profano i cardini della teoria musicale. Quindi, non necessaria per chiunque ne sia già a conoscenza.

### *2.2.1 – la partitura*

Per chiarire subite le idee, analizziamo la figura 3.

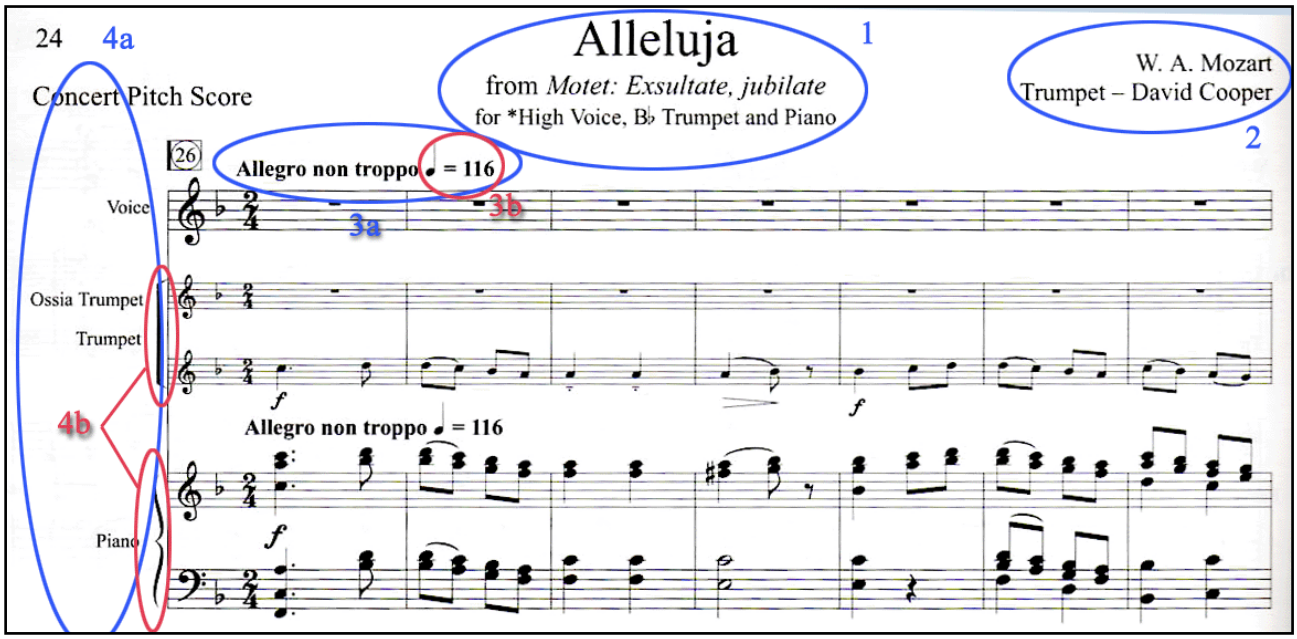



**Figura 3**

E' la prima pagina di una partitura di un brano musicale. Per la precisione, il primo rigo della pagina. Prima ancora di addentrarci nella lettura delle note, vediamo che ci sono molte altre indicazioni. Tutte molto importanti.
Analizziamo per il momento le parti cerchiate e numerate.

*1)* - Vi è il titolo del brano, l'opera da cui è estratto e il dettaglio sugli strumenti usati.

*2)* - L'autore del brano. Nello specifico, c'è un secondo autore per la parte della tromba

*3)* *a* - L'indicazione di agonica, informa l'esecutore ( o gli esecutori) sull'andamento ritmico da dare al pezzo.
*b* - è l'impostazione da dare al metronomo

*4)* *a* - le voci timbriche che compongono il brano. Nello specifico una voce solista, una tromba e un pianoforte.
*b* - uno strumento può avere più voci (come il pianoforte, dove una mano è indipendente dall'altra). Il segno in figura è un'accollatura e specifica questo caso particolare.

Alcune delle informazioni sopra elencate potranno risultare banali o scontate, ma non lo sono affatto!
Dobbiamo entrare nell'ottica di voler formalizzare ogni aspetto di un brano musicale. Ed inoltre, di volerlo fare in automatico ed a partire dall' *esecuzione musicale.*
Ci occuperemo nel prossimo capitolo di come farlo attraverso il linguaggio XML.
Per ora concentriamoci sulla notazione in sé, analizzando la figura 4.

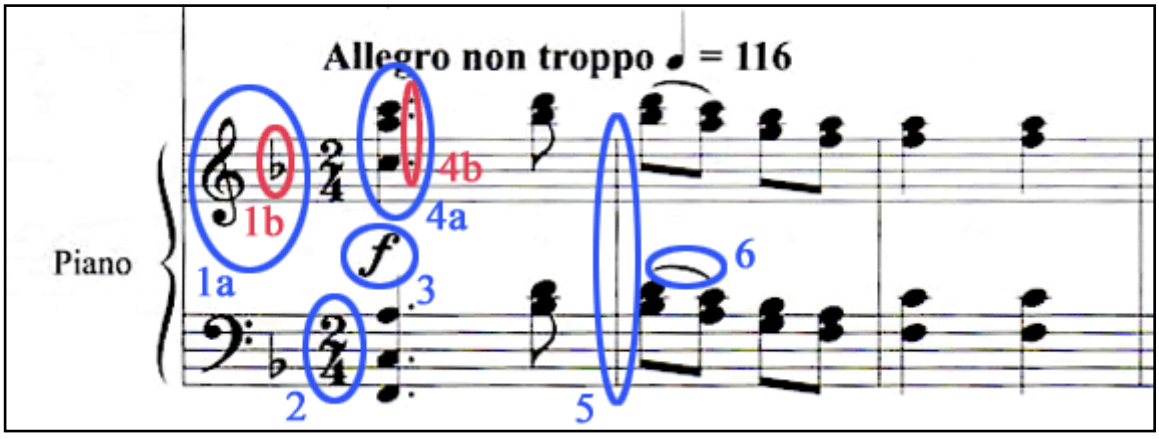



**Figura 4**

Si tratta del dettaglio della parte del piano, visto nella figura 3.
Tre strumenti, tre parti distinte, ognuno con una o più voci al suo interno. Questo sviluppo nucleico non è casuale. Oltre che essere musicalmente intuitivo è anche la base della struttura dati che useremo nello sviluppo del programma software.

Analizziamo come già fatto precedentemente le parti cerchiate

1) a – La *chiave*, presente per ogni pentagramma, serve ad identificare l'altezza delle note. La chiave indicata è una chiave detta *di violino*

b – l'armatura di chiave, presenta un segno chiamato bemolle. E' un alterazione della nota segnata. Tutte le note di quell'altezza saranno in realtà abbassate di un semitono.

2) Il *tempo*. O per meglio dire, la *divisione di durata* di ogni singola *battuta*

3) La *dinamica*. Indica quanto forte o quando piano debba essere eseguito un suono

4) a – La *nota*. Indica quale altezza debba essere suonata. E quanto dura rispetto alla divisione di durata. Nel caso specifico abbiamo 4 note che suonano nello stesso momento. Questo fenomeno è detto *accordo*.

b – il *punto di valore*, opzionale, aumenta la durata della nota indicata di metà del suo valore originario. Ve ne possono essere diversi, ognuno aumenta della metà di quello precedente

5) La stanghetta di *battuta*, dichiara la chiusura di una battuta e l'inizio di una nuova. La divisione di durata si riferisce proprio ad essa.

6) La *legatura*. Vi sono due tipi di legature. *Di frase* e *di valore*. Quella segnata è una legatura di frase. Indica che i due accordi fanno parte della stessa frase, quindi andranno eseguiti in maniera *continua*. Laddove non sia presente la legatura, fra un suono e l'altro può intercorrere un breve *respiro.*

E' probabile che alcuni dei punti appena segnati non siano molto chiari. A meno di non conoscere già qualche elemento di teoria musicale.
Tuttavia lo scopo è quello di mostrare quante siano le informazioni che porta con sé una partitura. Addirittura nel solo primo rigo.
A livello esecutivo, quello proposto non corrisponde neanche ad un minuto secondo di musica. Fra l'altro, del solo pianoforte, della sola mano destra!

E' chiaro quindi, che la partitura di un pezzo orchestrale con circa 40 elementi, della durata esecutiva di circa 30 minuti, porta con sé un grosso carico di informazioni. E noi dovremmo considerarle tutte.

### *2.2.2 – ritmo, melodia e armonia*

Prima di addentrarci nella spiegazioni degli elementi grafici che compongono la scrittura musicale, cominciamo col vedere in quali modi gli uomini possono utilizzare le caratteristiche dei suoni per costruire un discorso musicale. Più che il suono in sé infatti, sono importanti le relazioni che intercorrono tra i suoni.
E' il caso della durata.
In una successione di suoni, le durate di questi ce li fanno percepire come più lunghi o più brevi l'uno relativamente all'altro. Il rapporto di durata tra i suoni è alla base del *ritmo*.
Il più elementare modello di ritmo potrebbe essere ad esempio quello del ticchettio di un orologio. Qui gli intervalli sono tutti uguali e parliamo di ritmo *isocronico*.
Musicalmente, non è molto interessante, ma sono ritmi che incontriamo spesso nella vita comune. Pensiamo ai rintocchi di una campana, o ai passi di una persona durante una marcia.
C'è già tuttavia una piccola differenza.
Nel caso del ticchettio udiamo brevi suoni intervallati da silenzio. I rintocchi di una campana invece, sono suoni lunghi che non si interrompono fra un rintocco e l'altro (ciò è dovuto ovviamente alla vibrazione della campana che prolunga il suono). Ciononostante parliamo di ritmo isocronico in quanto *l'attacco* del suoni (il momento in cui essi hanno inizio) è a uguale distanza nel tempo. La presenza o assenza di silenzio tra due suoni ci da' una prima rudimentale spiegazione delle definizioni di *staccato* e *legato.*
Anche il battito cardiaco è un ritmo isocronico. Con la differenza che abbiamo due suoni molto vicini e un silenzio, due suoni molto vicini e un silenzio…
E' facile comprendere quindi come il ritmo sia il prodotto di rapporti di durata non solo dei suoni, ma anche degli eventuali silenzi tra un suono e l'altro. Tali silenzi, li chiameremo *pause*. Nel discorso musicale, le pause sono importanti quanto i suoni stessi e saranno anch'esse *eventi musicali*.
Unendo suoni e pause fra loro possiamo cominciare ad avere figure ritmiche leggermente più articolate.

Proviamo ad esempio a contare ad alta voce fino a 4 per dieci volte in totale. E omettiamo di dire “4” per tutte e dieci le volte. Abbiamo costruito un ritmo isocronico che ricorre ogni 4 unità di tempo, dato dal rapporto di tre pieni e un vuoto (tre suoni e una pausa).
Analogamente potremo omettere la parola “4” ma stavolta prolungando il “3”.
Ed il risultato sono due suoni brevi e uno lungo.

Un altro elemento che può avere molta importanza nella costruzione di una forma ritmica è l’*accentazione.* Grazie alla presenza di accenti (simili a quelli della lingua parlata) possiamo dare una configurazione ritmica ad una semplice successione di suoni senza bisogno di introdurre differenze di durata o di pause. Se ripetiamo l’esperimento precedente, stavolta enunciando anche il “4” ma accentando l’ ”1”, otteniamo comunque un ritmo che ricorre ogni 4 unità di tempo.

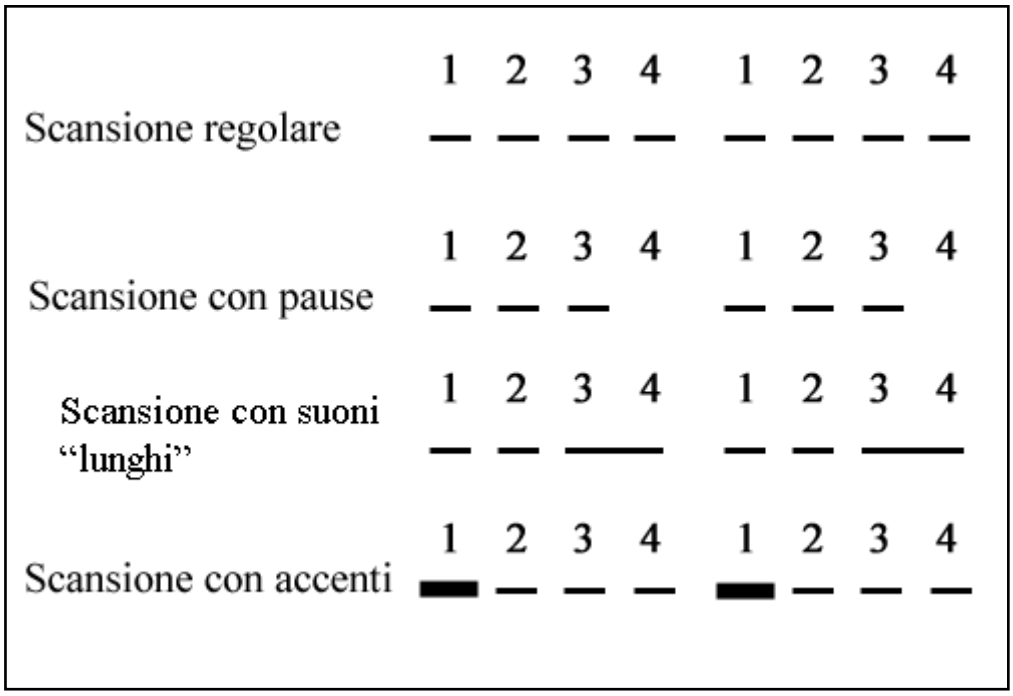



**Figura 5**

La figura 5, ci mostra come potrebbero essere rappresentati graficamente le tre forme ritmiche create.

Se invece di contare fino a 4, contiamo fino a 3, otteniamo un ritmo *ternario*. Tipico ad esempio, dei *valzer*.
Se contiamo fino a 2, una *marcia*.

Il ritmo, può quindi nascere da fatti diversi (rapporti di durata, presenza di pause, accentazione). Ci sono sistemi che privilegiano l'una o l'altra di queste componenti, ma nella pratica musicale, una configurazione ritmica è quasi sempre il risultato dell'insieme di questi fattori, e delle relazioni che vi sono fra di essi.

L'*altezza* dei suoni, è un'altra importante caratteristica che si relaziona con la musica. Infatti, ogniqualvolta che ad esempio fischiettiamo un motivo a noi familiare, non facciamo altre che riprodurre (soffiando ed inspirando), in successione nel tempo, i suoni che lo compongono. Ciò che ci rende riconoscibile "quel" motivo, è principalmente il rapporto tra le altezze dei suoni.
Questo primo risultato è la *melodia*. Essa è l'aspetto della musica che riguarda i rapporti di altezza tra i diverso suoni di una successione. Stiamo quindi escludendo, per il momento, i rapporti di altezza che possono esserci tra suoni *simultanei* (è evidente che il nostro fischio può produrre un solo suono per volta).
Proviamo a graficare anche quest'aspetto della musica.
Se nel caso del ritmo, per rappresentare le durate ci muovevamo da sinistra a destra, adesso sfrutteremo anche la dimensione verticale.
Così, rappresentando in figura "Fra Martino campanaro" (un pezzo largamente abusato nella teoria musicale per la sua semplicità e popolarità) otteniamo il seguente risultato della figura 6.

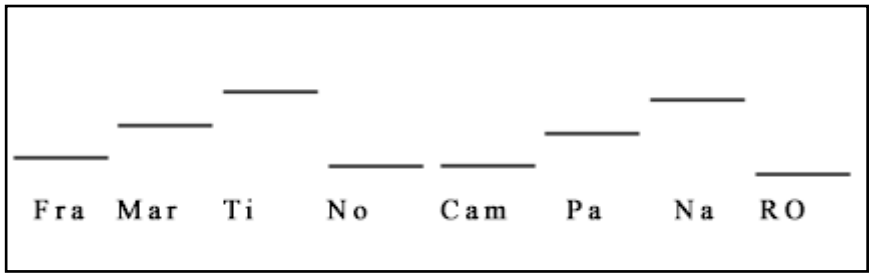



**Figura 6**

Le linee sopra rappresentano il percorso della voce (ora salendo, ora scendendo) quando intoniamo una melodia. Se la voce seguisse un altro percorso, non saremmo più in grado di riconoscere quel motivo.
Quando definiamo una melodia, non ci riferiamo ad una qualsiasi successione di suoni di altezza diversa. Come per il ritmo, abbiamo diverse *configurazioni melodiche* organizzate secondo svariati schemi. Anche piuttosto complessi.

Ed è altresì importante, che la melodia sia formata da una successione di suoni limitata nel tempo. Che abbia cioè una conclusione, e permettere così al cervello umano di ricostruirla e rendercela nel suo intero (se non fossimo capaci di fare questo, e quindi fossimo privi di memoria, percepiremmo sempre e solo un suono alla volta, quello udito in ogni singolo istante).
Possiamo fare un'analogia con il linguaggio verbale.
Se leggiamo una frase scritta molto lunga, con molte virgole e parentesi, potrebbe essere difficile percepire il senso del discorso. Infatti, nel linguaggio parlato, si tendono ad usare frasi più brevi e concise.
Anche per la melodia vale questo discorso. Ma non è il solo.
Così come nel linguaggio verbale, non bastano le parole. Ci vuole organizzazione e logica.
Ed un senso, ovviamente!
Ed un senso musicale, lo hanno anche le *frasi* melodiche.

Il discorso musicale si compone di più livelli, ognuno con diversi strati al suo interno. Per riprendere le analogie col linguaggio verbale, paragoniamo un brano musicale ad un testo letterario. Il quale può essere analizzato dividendolo in singole parole, in frasi e in periodi più o meno complessi.
I musicisti hanno sviluppato tutto un linguaggio specifico per questo tipo di analisi, detto *fraseologia*. In essa, le parti più o meno grandi del discorso musicale vengono definite con termini tecnici quali *inciso*, *semifrase*, *frase, periodo*.
Ad esempio l'inciso (il più piccolo elemento melodico) viene detto anche *motivo*, se ha una fisionomia musicale molto spiccata e riconoscibile. Pensiamo a titolo di esempio, alle prima quattro note della quinta sinfonia di Beethoven.
Quello che invece nel linguaggio comune è detto motivo, si definisce *tema*, nella teoria musicale.

E' bene sottolineare che i rapporti di cui stiamo parlando riguardano strettamente l'aspetto musicale. Essi risulterebbero chiari al nostro orecchio anche se ascoltassimo la melodia eseguita con un violino, e quindi senza sentirne le parole. E' chiaro che l'esempio dell'inno di Mameli è volutamente semplificativo in quanto non sempre i brani musicali (omettiamo infatti anche le parole e concentriamoci sulla musica pura) seguono una struttura fraseologica così ordinata. Può essere molto più complesso, e in definitiva, anche più

interessante. Di solito una trovata più originale ed anche più elegante da parte dell'autore, desta nell'ascoltatore maggiore apprezzamento.

Per completare il discorso formale della struttura musicale, esaminiamo il caso di suoni simultanei.
Infatti, se per esempio volessimo cantare accompagnati da un pianoforte, dovremmo porre attenzione non solo al rapporto fra i suoni che produco, ma relazionarli anche con quelli eseguiti dal pianoforte. E del resto, chi suona solo il pianoforte, produce normalmente diversi suoni allo stesso tempo. Ci stiamo riferendo alla *polifonia*, già trattata storicamente nei paragrafi precedenti.
Dobbiamo quindi distinguere due dimensioni nella grafia musicale.
Quella orizzontale, identifica i suoni[3] successivi nel tempo. Quella verticale, la convivenza e la combinazione dei suoni simultanei.
L'esempio più semplice di polifonia, lo abbiamo nell'esecuzione di un accordo. Cioè 2 o più note suonate nello stesso istante. Come due tasti di un pianoforte abbassato nello stesso istante.
Una serie di accordi, suonati insieme ad una voce monodica, viene detta *melodia di accompagnamento*.
I principi che regolano la formazione degli accordi (i rapporti di altezza fra i diversi suoni simultanei che lo compongono, come passare da un accordo all'altro, o in che modo una serie di accordi possa accompagnare una determinata melodia) fanno parte della *teoria armonica*. O semplicemente *armonia*.
Infatti, come già per il ritmo e la melodia, non basta avere due suoni a caso simultanei per avere un accordo *consono*.
Esistono delle regole, legate fra l'altro anche al periodo storico ed al genere musicale, per produrre un accompagnamento che dia un *sostegno armonico* ad una precisa melodia. E vi possono essere più armonie che ben si sposano alla stessa melodia, dandone ciononostante, una caratterizzazione completamente diversa.
Oltre questa, esiste un'altra tecnica per la combinazione simultanea dei suoni. Il *contrappunto*.
Con essa, vengono eseguite più linee melodiche contemporaneamente.

[3] In realtà più che suoni (quindi note), parliamo di tutti i simboli presenti in una partitura.

Se nell'accompagnamento armonico c'è una precisa gerarchia (la melodia è la parte principia mentre l'armonizzazione è il supporto), nel contrappunto tutte le parti simultanee hanno la stessa importanza. Almeno in linea di principio.

Anche in questo caso, esistono delle regole per cui la combinazione della voci[4] abbia un risultato *musicale*. E non semplicemente cacofonico.

Una tecnica comune del contrappunto è l'imitazione. In cui le diverse voci riprendono in momenti successivi lo stesso elemento melodico.

E ci basti come esempio, il provare a cantare Fra Martino insieme ad altre persone. Dove ognuno però, attaccherà un qualche istante dopo colui che lo precede.

[4] Per voce non si deve intendere necessariamente una voce umana, ma anche una parte eseguita da uno strumento musicale.

## *2.3 – La grafia*

All'inizio del capitolo, abbiamo dato uno sguardo alla storia e all'evoluzione della notazione musicale occidentale. Ne risulta un sistema estremamente ricco di segni grafici. Uno sguardo al suo funzionamento permetterà certamente una maggiore comprensione del lavoro di tesi svolto.
Noteremo anche come, il diverso uso della grafia, permette di costruire e meglio comprendere ritmo, armonia e melodia.

### *2.3.1 – l'altezza*

La *nota* è il segno grafico che identifica un suono. La forma che la rappresenta è un pallino. La posizione sul *pentagramma*, cioè le 5 linee parallele su cui vanno segnate tutte le note a partire dal basso verso l'alto, ne determina l'altezza. Più è bassa la nota, più il suono è grave.
Poiché il pentagramma (o *rigo musicale*) può contenere solo un numero limitato di suoni, per quei suoni che ne varcano il limite, si usano i tagli addizionali. Essi sono lineette che indicano la continuazione del rigo musicale. Se la nota è attraversata dal taglio, si considera posta sul rigo; se ne è preceduta, si considera segnata nello spazio.

I *nomi* delle note, sono sette (do, re, mi, fa, sol, la, si). Dando uno sguardo alla figura X che rappresenta una porzione dei tasti bianchi di un pianoforte, ci accorgiamo che esse ricorrono in successive sezioni della tastiera. Ogni tasto ha di conseguenza lo stesso nome di tutti quelli che si trovano a distanze regolari sia verso il grave che verso l'acuto. Essendo i tasti otto, la distanza fra suoni di stesso nome è definito *ottava*.
Un pianoforte ha otto ottave, più di ottanta tasti. E' chiaro che il solo pentagramma, seppur con i tagli addizionali, è insufficiente a raffigurare tutte le note eseguibili su tale strumento.

Per identificare il nome delle note, e la loro altezza c'è bisogno della *chiave*.
Nel sistema musicale moderno, la chiavi sono sette e si dividono in tre specie. Tre sono anche i simboli grafici che le rappresentano. Indicano le note DO, FA e SOL nella loro notazione alfabetica (C, F, G).

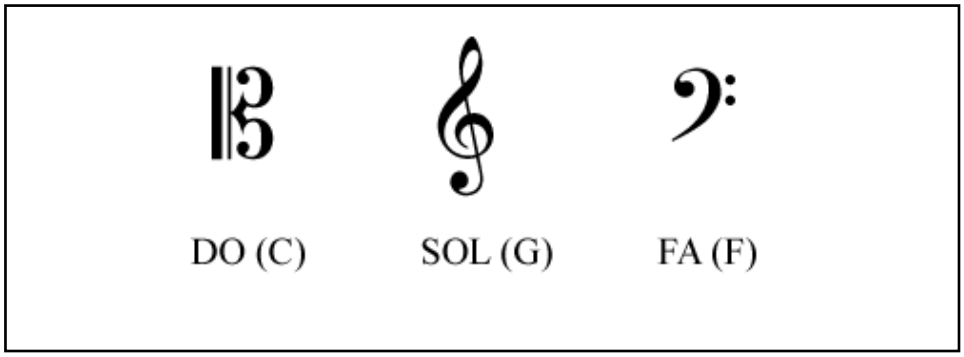



**Figura 7 - I tre simboli delle chiavi**

Ogni chiave interessa una porzione ben determinata della gamma grave-acuto; una nota in una certa posizione (sia essa su di un rigo o in uno spazio) assume nome diverso, e quindi altezza diversa, in relazione al segno di chiave posto all'inizio del pentagramma.
La chiave indica il nome della nota associata al rigo su cui è posta. Se ad esempio, avessimo la chiave di *sol* posta sul secondo rigo, tutte le altre note assumeranno denominazione e altezza in relazione al sol. Logicamente non un *sol* generico, ma il sol di quarta ottava.
Chiave unica di do è quella che si pone su una linea intermedia che unisce due pentagrammi, formando così il rigo di undici linee. La nota segnata su questa linea corrisponde al do centrale (do di quinta ottava).
Questo sistema serve per gli strumenti a grande estensione come il pianoforte o l'organo.
La figura Y esemplifica quanto detto finora.

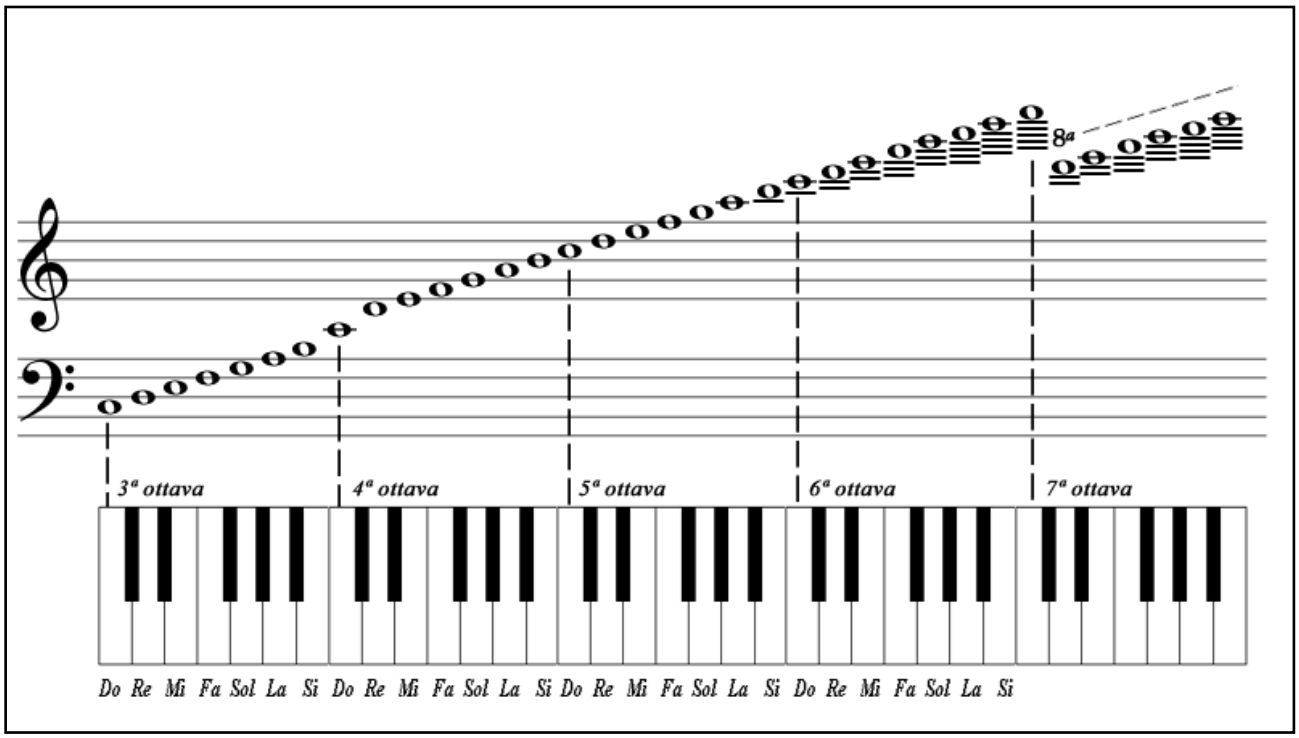



**Figura 8 - rapporto tra note e altezze**

Le due chiavi rappresentate, sono rispettivamente la *chiave di violino* (SOL) e la *chiave di basso* (FA). E sono indubbiamente le più frequenti.

Quello che risalta da questo esempio grafico, è la mancanza di riferimenti per quelli che sono i tasti neri.
Se infatti le note sono sette per ogni ottava, è anche vero che tale ottava è divisa in 12 suoni diversi in altezza. Più precisamente in 12 *semitoni.* Un semitono è quindi, nel nostro attuale sistema musicale, il più piccolo intervallo tra un suono e l'altro. Due semitoni uniti, formano un *tono*.
Analizzando di nuovo un'ottava di un pianoforte, la distanza che intercorre tra due tasti contigui (sia due tasti bianchi o uno nero e uno bianco) è appunto di un semitono. Distano di un semitono ad esempio il Mi e il Fa e il Do con il primo tasto nero.

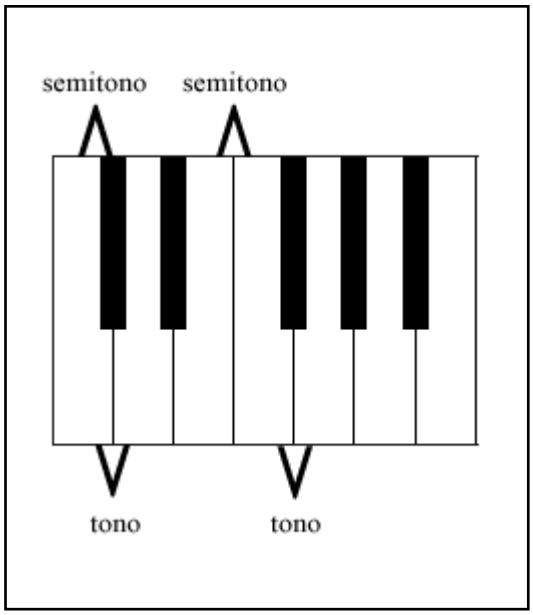



**Figura 9 - toni e semitoni**

I segni grafici e quindi anche i nomi che accompagnano le note (che rimangono sempre sette!) per identificare questa particolarità, sono le *alterazioni*.
Esse aumentano o diminuiscono i suoni di un semitono o di un tono.
Il *diesis* (#) aumenta di un semitono.
Il *bemolle* (b) lo abbassa. Doppi diesis e doppi bemolli aumentano o abbassano di un tono.

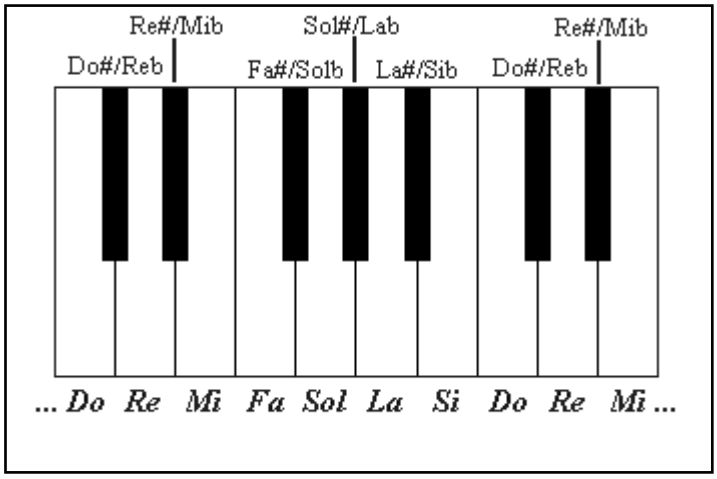



**Figura 10 - omofonia**

Come si vede, uno stesso suono può essere rappresentato graficamente in diversi modi. In questo caso, parliamo di suoni *omofoni*.
Le alterazioni possono essere fisse o transitorie. Quelle fisse sono segnate all'inizio della composizione, subito dopo la chiave e stabiliscono la *tonalità* del pezzo. Tutte le note che incontreremo durante l'esecuzione del brano dovranno essere considerate alterare di un semitono (in più o meno a seconda che ci siano diesis o bemolle[5]) in base al numero di alterazioni presenti (logicamente, massimo 7).

Le alterazioni transitorie sono segnate prima del suono e durano per quella sola misura[6].
Esiste un'ulteriore alterazione, il *bequadro*, che riporta la nota al valore naturale.

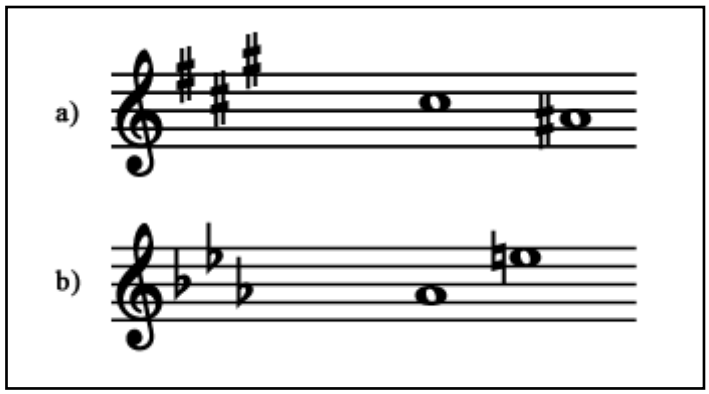



**Figura 11 - alterazioni**

[5] Non è possibile avere sia diesis che bemolle nell'armatura di chiave. O l'una o l'altra, oltre al caso ovvio di nessuna alterazione fissa.
[6] Le misure saranno trattate nel paragrafo delle durate.

Il numero e il tipo di alterazioni in chiave definiscono, fra le altre cose, la *tonalità*, di cui ci occuperemo più avanti.

### *2.3.2 – le durate*

Graficamente, la durata di una nota si evince dal *gambo* che accompagna il pallino e dal riempimento o meno di quest'ultimo, e dalla presenza o meno di una o più code (*cediglie*) sul gambo. Questi elementi compongono la *figura musicale*. Oltre ad esse troviamo e le pause, cioè i silenzi da noi sperimentati nel primo paragrafo di questo capitolo.

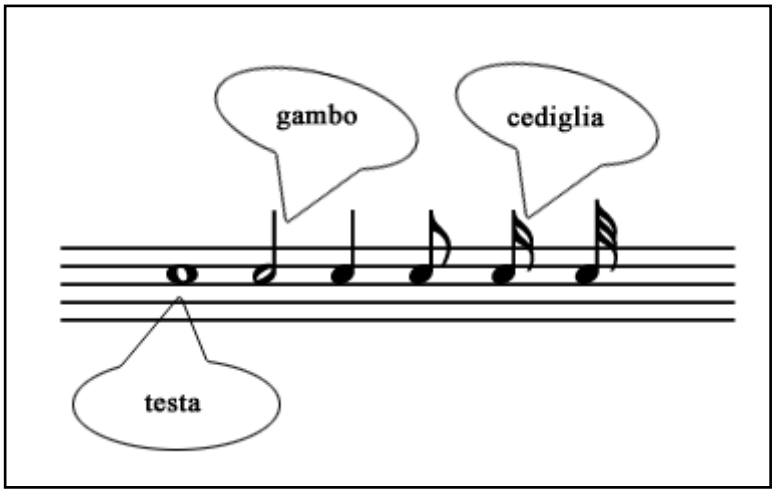



**Figura 12 - figure musicali**

Prima tuttavia di elencare le diverse forme possibili, è bene chiarire cosa vogliono dire *tempo* e *durate* nella teoria musicale.
L'indicazione temporale non sta a dirci in quanto tempo "fisico" dobbiamo o meno eseguire un pezzo. Piuttosto, ed in maniera anche piuttosto rigorosa dal punto di vista matematico, ci da *un metodo di divisione*.
Questo vuol dire che assegnato un valore unitario ad una certa figura musicale, tutte le altre saranno multiple o sottomultipli di tale valore di riferimento.
Le figure musicali e le pause vengono utilizzate per determinare la durata dei suoni e dei silenzi in
un brano musicale.
Le figure e le pause vengono inserite nel pentagramma in gruppi di valore uguale, chiamati *battute* o *misure*.
Le battute sono delimitate da lineette verticali chiamate *spezzabattute* o *stanghette*.

E' all'interno della battuta che viene applicato il metodo di divisione. Ed è il susseguirsi delle battute che determina lo scorrimento del brano musicale in partitura.

Torniamo per un momento alle note ed alle pause. Ed alla loro durata.
Nel nostro sistema musicale si tende ad attribuire il valore unitario alla nota detta *semibreve* (un pallino bianco senza gambo). Tutte le altre (con le rispettive pause) valgono in serie, esattamente la metà della precedente.
Lo schema raffigura quanto detto.

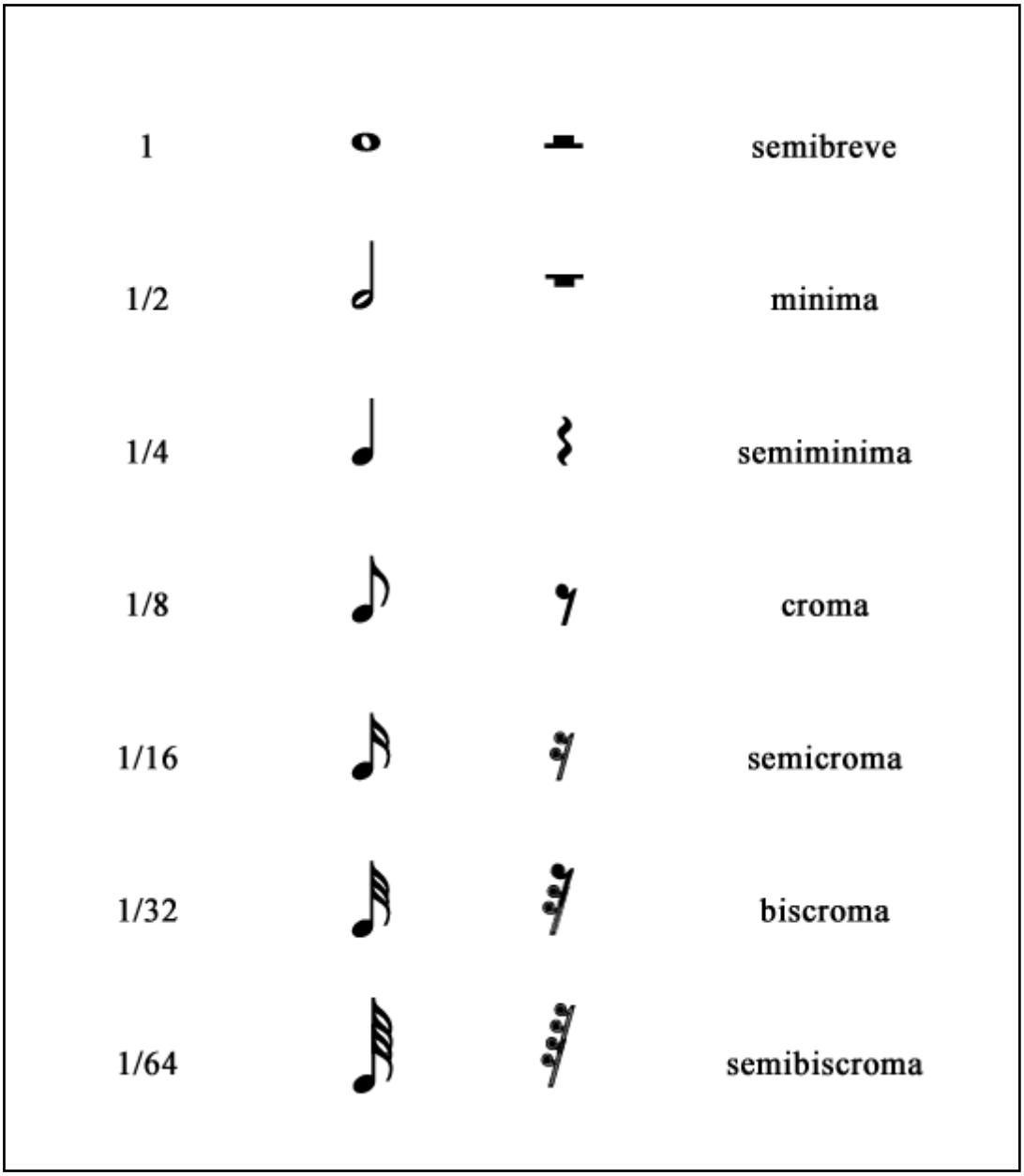



**Figura 13 - durata delle note**

Inoltre, vi sono due espedienti grafici che modificano la durata di un valore.
La *legatura di valore*, che collega due o più suoni di stessa altezza, indica che si deve in realtà eseguire un solo suono, di durata pari alla somma dei valori dei suoni legati tra loro.
Il *punto di valore*, scritto a destra del valore (suono o pausa) cui si riferisce, ne allunga di metà la durata, facendolo durare una volta e mezzo del valore originale.
Un eventuale punto aggiuntivo aumenterebbe di un ulteriore quarto e così via.

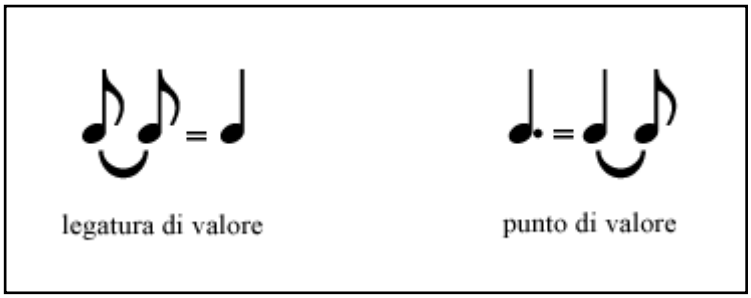



**Figura 14 - legature e punti di valore**

A questo punto, il *tempo* o *indicazione di misura*, che viene segnato immediatamente dopo la chiave ed alla sua armatura se presente, determina la suddivisione di una singola battuta.
E' indicato con una frazione. Ad esempio 4/4.
Con questo tempo, una battuta viene divisa in 4 movimenti o tempi da ¼. O in qualsiasi altro modo, purché la somma delle durate[7] presenti in battuta sia sempre 4/4.
Questo valore dell'indicazione di misura è molto comune, così come lo è la divisione ternaria. Ad esempio ¾. Spesso usata nei walzer.

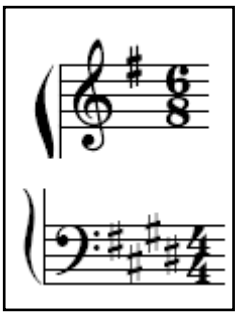


**Figura 15 - durate**

Ma una battuta non è solo un'unità di durata: essa è anche un sistema di *accentazione*.

[7] Ci riferiamo sempre a note e pause

Ogni misura contiene un accento principale sul primo movimento (cioè, come si dice, sul *tempo forte* o *in battere*). Ad esempio se dividiamo una successione di suoni isocroni in battute di due avremo un accento ricorrente ogni due suoni, collocato ad inizio di ogni battuta. Ugualmente per le battute di tre movimenti.
Parleremo rispettivamente di accentazione binaria e ternaria, le più usate nella nostra tradizione musicale. Infatti, anche nel tempo ordinario 4/4 viene ricondotto all'accentazione binaria. Il più forte dei due accenti, come sempre, ad inizio battuta (primo movimento), il secondo a metà (terzo movimento).
Questa organizzazione degli accenti all'interno della battuta rimane sempre costante anche quando le misure contengono suoni e pause non isocroni ma di diverso valore.

Esistono poi delle eccezioni, riconducibili nelle *figure irregolari*.
Se volessimo ottenere tre crome nello spazio occupato da due, le contrassegniamo con un simbolo di legatura ed il numero "3" ed otteniamo una *terzina*.

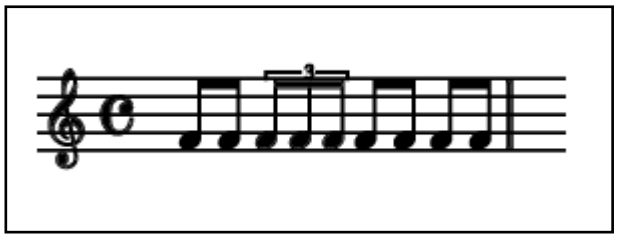


**Figura 16 - figura irregolare, la terzina**

La "C" raffigurata è un acronimo di 4/4. La stessa lettera con un taglio perpendicolare indica il tempo 2/2. Oltre ai tempi *regolari* comunque, ne esistono anche di *irregolari*. Questi ultimi risultano dall'unione di due misure una pari e una dispari o viceversa. Ad esempio 5/4.
A prescindere da questo però, ciò che è importante rimarcare riguardo la durata, è, come detto, che essa ci suddivide la battuta ma non ci indica assolutamente la durata in unità di tempo fisico. Quest'ultimo è assolutamente arbitrario.
Vedremo nel prossimo paragrafo quanti altri segni grafici ci diano una mano ad interpretare l'andamento ritmico ed espressivo di un brano musicale.

### *2.3.3 – abbellimenti e segni d'espressione*

- **Legature**

Ad eccezione della legatura di valore, vista precedentemente, tutte le altre (*di fraseggio*, *di portamento*, *d'attacco* e *di smorso*) rientrano nei segni d'espressione.
La prima, se unisce più suoni o un'intera frase indica che i suoni si debbono eseguire senza interruzioni.
La seconda, unendo due suoni ascendenti o discendenti, tende a far marcare leggermente il primo rispetto al secondo.
La legatura d'attacco, posta prima di un suono di un accordo, ne rende più dolce l'esecuzione.
Infine la legatura di smorso, posta dopo un suono, porta ad avere un risultato molto sfumato.

- **Punto d'espressione.**

Da non confondere col punto di valore, il *punto d'espressione*, normale o allungato, è posto sopra la nota tende a diminuire il valore della nota, rendendo il suono *staccato*.

- **Punto coronato**

Posto sopra una nota o una pausa, ne prolunga il suono o il silenzio per un valore indeterminato, a completa discrezione dell'esecutore.

- **Segni dinamici**

I *segni dinamici* riguardano le diverse gradazioni di sonorità di un pezzo.
Abbiamo *p* (piano), *pp* (pianissimo), *f* (forte), *ff* (fortissimo), *mp* (mezzo piano), *mf* (mezzo forte), *sf* (sforzato).
Per variazioni graduali si usano i segni *rinf* (rinforzando), *cresc* (crescendo), e *dim* (diminuendo). Questi ultimi si possono segnare anche con linee divergenti e convergenti dette *forchette*.

- **Andamento**

E' il grado di velocità che si vuole impiegare nell'esecuzione. Si segna all'inizio delle composizione con i termini *Lento, Grave, Largo, Adagio, Andantino, Andante, Moderato , Allegro, Presto* e dura per tutta la composizione se non vi sono altre indicazioni. Al più vi possono essere dei rallentare o affrettare che modificano momentaneamente o parzialmente l'andamento iniziale.
Tuttavia, spesso il termine dell'andamento è seguito da un aggettivo : solenne, brioso, con fuoco, doloroso ecc.; questi termini servono a determinare il carattere dell'andamento stesso. Per molto tempo, e ancora oggi, questi termini sono scritti prevalentemente in lingua italiana in Europa e in Occidente in generale.

E' bene aggiungere che accanto l'andamento, è consuetudine trovare l'indicazione del metronomo, con una figura ed una cifra. La figura corrisponde all'unità di misura di divisione o di suddivisione. La cifra indica il numero di oscillazioni che il pendolo del metronomo compie in un minuto[8].

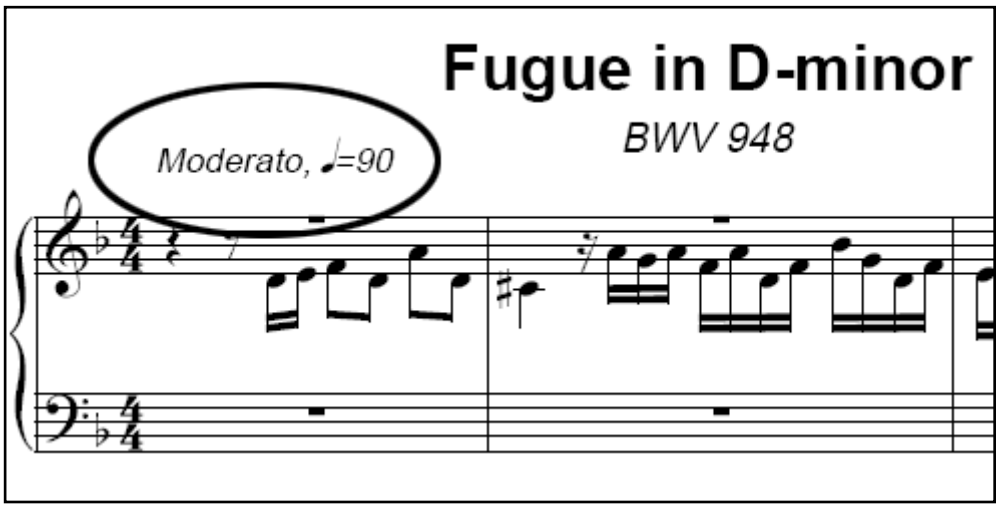



**Figura 17 - agogica e metronomo**

- **Abbellimenti**

[8] Questo espediente ovvia il discorso dell'arbitrarietà del tempo. Ma raramente un compositore lo indica. Ed un interprete spesso, se presente, ignora questo dato.
E' un'indicazione perlopiù usata a scopo didattico.

Sono gruppi di *notine* (in opposizione alle note *reali*) che arricchiscono ed ornano la melodia. Gli abbellimenti per se stessi non hanno valore, ma lo acquistano dalle note reali se essi sono seguiti in sottrazione, e dalle note che precedono le note reali se esse sono eseguite in anticipazione.

Le più usate attualmente sono l'acciaccatura, l'appoggiatura, il mordente, il gruppetto e il trillo.

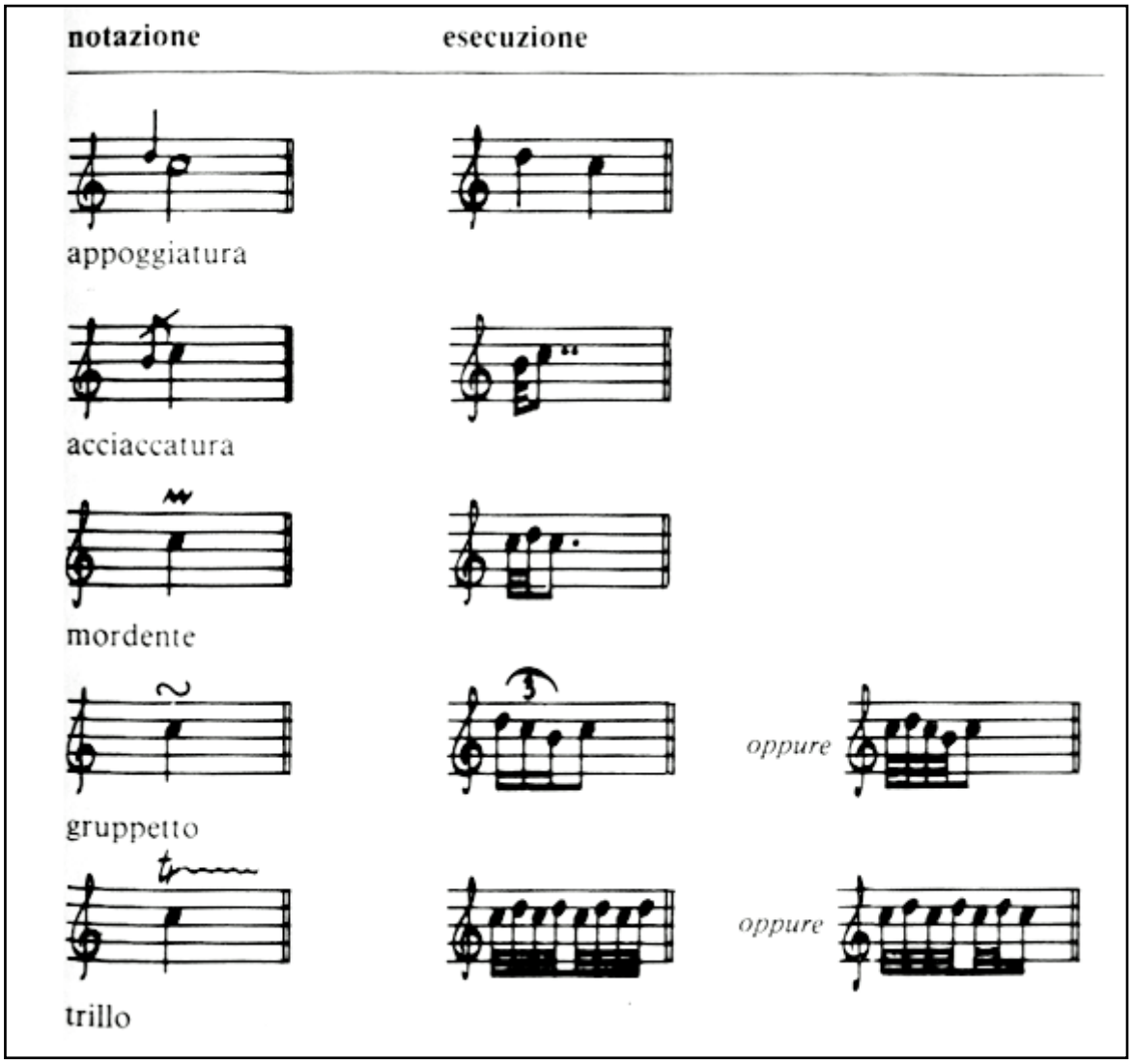



**Figura 18 - abbellimenti**

### *2.3.4 – tonalità*

Chiudiamo la questa sezione sulla teoria musicale parlando della tonalità.
Le note della scala, e gli accordi costruiti su di esse, obbediscono a delle leggi che li pongono necessariamente in relazione rispetto alla *tonica* (tonica è la nota che da il nome alla tonalità). Ciascuna nota o accordo della scala si trova quindi ad essere in qualche modo subordinata alla nota (o accordo) principale che appunto è la tonica. Tuttavia questa "subordinazione" non è univoca, ma vi sono gradi della scala paradossalmente più armonicamente forti della tonica stessa: la *dominante* (il V grado della scala) è capace di determinare o ribadire la tonalità in maniera assai più efficace della tonica. La successione dominante-tonica nella cadenza è considerata la successione armonica più importante di tutto il periodo tonale.
La tonalità è scelta liberamente dal compositore del brano, spesso tenendo conto della difficoltà esecutiva o delle caratteristiche peculiari dello strumento; si distingue in tono maggiore e tono minore. La tonalità prende il nome dal grado della scala cromatica che assume la massima importanza (centro tonale) nell'ambito della composizione.

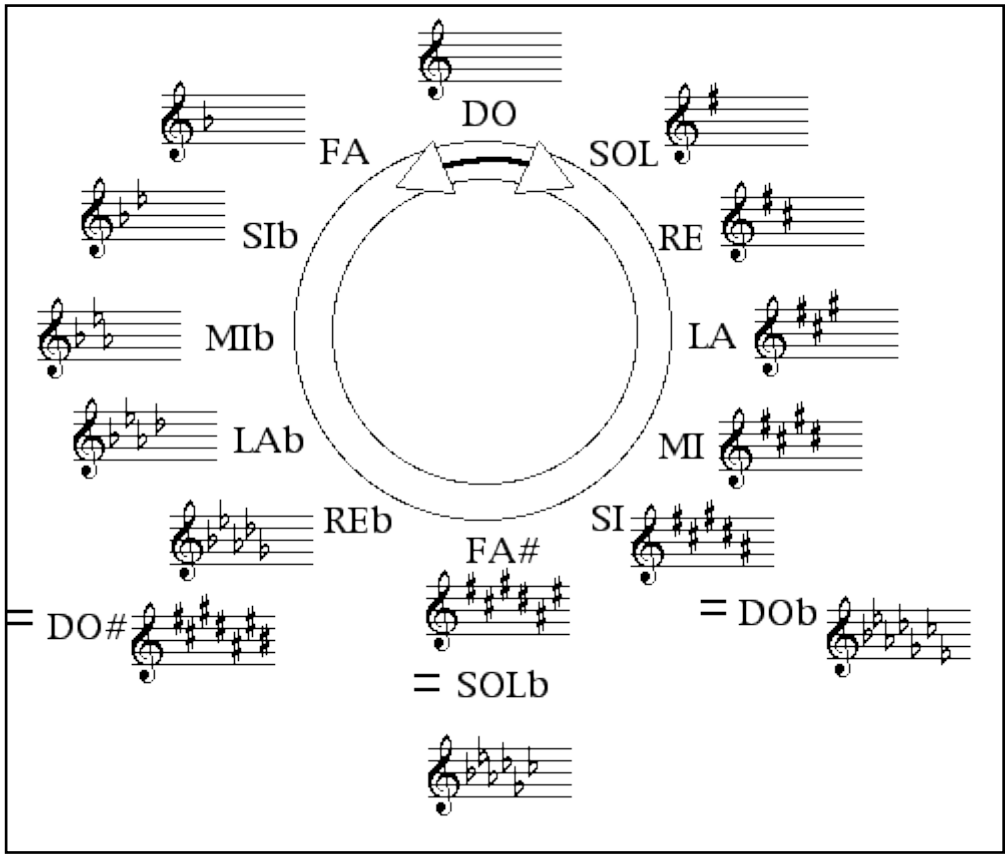



**Figura 19 - le tonalità**

Tono maggiore e tono minore possono essere applicati a partire da ognuna delle note della scala, generando così 24 tonalità: quando si utilizza (come è la norma nelle composizioni moderne) la scala temperata, che divide l'ottava in 12 semitoni uguali, alcune tonalità assumono lo stesso nome (tonalità enarmoniche), com'è ad esempio il caso di Do# e Reb. All'interno del brano la tonalità può cambiare e questo processo prende il nome di modulazione. I cambi tonali sono normalmente temporanei: così, nella composizione tonale, il trattamento da parte dell'autore dell'allontanamento del brano dal centro tonale e il suo ritorno ad esso costituisce gran parte del carattere del brano.

# Capitolo 3

# XML e MX

Nel capitolo precedente si è posta attenzione fra le altre cose, su un aspetto molto importante della musica scritta. Essa è molto articolata, rigorosa nella forma, ma altamente interpretabile in fase di esecuzione.
Cercheremo adesso, di ottenere lo stesso grado di precisione e descrizione in formato elettronico. Useremo cioè il linguaggio XML, per scrivere un file che in tutto e per tutto rappresenti una partitura e che in più, possa portare con sé qualsiasi informazione legata ad una traccia audio digitale. Questo particolare file XML che useremo nel corso di questa tesi, viene definito *MX*. E' una grammatica di XML, definita presso l'Università Statale di Milano dal LIM, Laboratorio di Informatica Musicale.
In ogni caso, non è l'unica. La descrizione dell'informazione musicale attraverso l'XML è uno studio internazionale che coinvolge molti studiosi e centri di ricerca dal 2001.
Molte sono le grammatiche e le soluzioni adottate, e illustreremo brevemente anche alcune di queste.

## *3.1 – XML*

### *3.1.1 – introduzione al linguaggio [10]*

eXtensible Markup Language, nato nel 1998, è un linguaggio che ha come obiettivo la rappresentazione di informazioni strutturate. Non ha importanza il tipo, purché riconducibili a precise regole. Possiamo riassumere alcune di queste regole nel concetto di mark-up (etichetta). Tale termine indica il fatto che i contenuti e la struttura del linguaggio sono definiti da simboli che identificano con esattezza le aree informative. Generalmente queste aree danno indicazioni all'interno di un testo atti ad istruire un compositore o un dattilografo su come deve essere graficamente distribuito un particolare passaggio.
L'XML infatti è un sottoinsieme di SGML, il metalinguaggio[1] che definisce i linguaggi di mark-up.

[1] Ovvero, insieme di regole generalizzate usate per creare molteplici linguaggi particolari. Nel nostro caso, linguaggi di mark-up come appunto, l'XML.

Dal punto di vista di SGML, un documento è una struttura gerarchica di elementi annidati; tale linguaggio non ha i mezzi per specificare aspetti di presentazione di tali elementi, né è in grado di fornire informazioni sul ruolo di alcun elemento. Queste informazioni sono sottintese dal creatore dell'applicazione SGML e sono solitamente auto-esplicative, o fornite a commento, oppure presenti nella documentazione che accompagna la specifica formale. L'SGML ci da un'unica informazione sulla natura di un elemento, e cioè in quali contesti e livelli di gerarchia esso possa o debba apparire.

Un noto linguaggio di mark-up è senza dubbio l'HTML, usato per la costruzione di pagine web e formattazione di ipertesti. Qui il concetto di etichettatura è molto chiaro. Testi in grassetto, giustificazione, inserimento di immagini e link sono tutte operazioni che vengono fatto contrassegnando secondo le regole del linguaggio, alcune porzioni di testo. Il browser si occuperà di interpretare tali informazioni e renderle in output nella maniera richiesta dall'autore.

```
<html>
<head>
<title>Storia del Personal Computer</title>
</head>

<body>
<font SIZE="4">

<p align="center"></font><font size="5"><b>STORIA DEL PERSONAL COMPUTER</b></font>
<font SIZE="4">
</p>

<p align="center">
<img src="IBM.jpg" width="544" height="459" alt="IBM.jpg (74208 bytes)">
</p>

<p ALIGN="JUSTIFY">
Il 12 agosto 1981 veniva presentato ufficialmente alla stampa specializzata
il personal computer di <b>IBM.</b> Dopo anni di ostracismo verso quegli oggetti,
la multinazionale presenta una macchina dalle dimensioni ridotte e con prestazioni
piuttosto modeste, indicata più genericamente come <i>microcomputer</i>.
</p>
```


L'XML ha un funzionamento analogo ma delle particolarità in più. Questo perché mentre l'HTML codifica come *visualizzare* i dati, l'XML si occupa di *descriverli*. Quindi, ne da un esatto significato.

Una differenza importante ad esempio, è il concetto di validità.

Infatti, se scriviamo un file XML, oltre a dovere rispettare quelle che sono le regole per così dire "grammaticali" (well formed[2]), c'è da rispettare la "semantica".

[2] In realtà l'HTML non è così rigoroso rispetto alle sue regole del *well formed.*

Una semantica che nel caso dell'HTML è intrinseca nel linguaggio e non modificabile dall'utente; con XML invece, del tutto personalizzabile. Essa risiede in altro file, il DTD (Document Type Definition), scritto anch'esso in XML e che contiene tutta la struttura e i mark-up che vogliamo usare per i nostri scopi. E' grazie al DTD che possiamo ottenere un particolare formato file che tratti la scrittura musicale. Ed è qui che definiamo quali sono i nostri mark-up.

### *3.1.2 – elementi e attributi [11]*

Gli *elementi* rappresentano i mattoni costitutivi di XML. Essi consentono di attribuire un preciso significato ad una parte di documento.
Ciascun tipo di elemento è rappresentato da un contrassegno (*mark-up*), detto anche etichetta (*tag*).
La sintassi dei tag è <element_name>…</element_name>, ove element_name rappresenta una stringa alfanumerica arbitraria. Dunque il singolo elemento viene delimitato in realtà da una coppia di etichette, dette rispettivamente etichetta di apertura (*start tag*) e di chiusura (*end tag*). Si noti il carattere "/" anteposto all'identificativo nel tag di chiusura.
All'interno di tale coppia si trova il contenuto del tag, che – mutuando la terminologia dei linguaggi di programmazione – si può considerare un'istanza dell'elemento.
I linguaggi di mark-up consentono l'annidamento di elementi a qualsiasi livello. In altre parole, ogni elemento potenzialmente contiene altri elementi, detti **sottoelementi** (*sub-elements*).
Nell'esempio seguente l'elemento `<address>` ha annidati al proprio interno gli elementi `<street>` e `<city>`, mentre questi ultimi racchiudono del semplice testo.

```
<address>
     <street>Via Chopin, 111</street>
     <city>Milano</city>
</address>
```

Analogamente, la stessa informazione potrebbe essere rappresentata in questo modo

```
<address>Via Comelico, 39 - Milano</address>
```

Evidentemente però l'informazione è meno strutturata. Di conseguenza, sarebbe più difficile estrapolare per via automatica la parte di indirizzo relativa alla città.
Tralasciando per ora la possibilità di definire dei vincoli sulla semantica dei contenuti di un elemento, riscontriamo quanto meno i seguenti limiti:

- non è certo che l'istanza di `<address>` contenga l'informazione richiesta (in questo caso, il nome della città)
- nel caso l'informazione sia effettivamente presente, non si ha garanzia di trovarla nella parte terminale della stringa, e separata dal resto con un trattino seguito da uno spazio

Vale anche il discorso inverso. Ad esempio potevamo inserire, nel primo esempio, un ulteriore elemento contenente il numero civico.

Ovviamente un'analisi "a mano" rende subito evidente la presenza o meno dell'informazione richiesta, e la sua ubicazione. Non avremmo alcuna difficoltà a estrapolare la città in un indirizzo (peraltro contenente un altro nome di città) quale:

Viale Roma, 74
20100 Milano (MI)
Italia

La nostra capacità di interpretare l'informazione ci permetterebbe di leggere senza alcuna difficoltà anche un indirizzo scritto in modo completamente diverso, quale ad esempio:

Milano – v.le Roma n°74

in cui il nome della città viene premesso a quello della via, si usa un trattino per separare le parti relative a città e via, la parola "Viale" viene abbreviata in "v.le" minuscolo, il numero civico viene espresso in modo differente, manca l'indicazione di provincia e via dicendo.
In definitiva, l'uomo sa come interpretare un certo tipo di informazione, alle volte addirittura

la integra grazie alle proprie conoscenze, ad esempio ricavando la sigla della provincia anche se non esplicitamente espressa, come nella seconda versione dell'indirizzo.
In ogni caso, tutto questo è totalmente estraneo a una macchina e a un sistema automatico (ossia privo di intervento da parte di esperti umani) per l'estrazione dei dati.
Questi e altri problemi si risolvono con una forte strutturazione dell'informazione, al prezzo di una maggiore rigidità (la struttura deve rispettare certe regole e non ammette varianti) e di una maggiore occupazione di risorse (i 74 byte di **Errore. L'origine riferimento non è stata trovata.** contro i 44 byte di).

Riassumendo un elemento può contenere

- Altri elementi (*sub-elements*)

```
<address>
     <street>Via Chopin, 111</street>
     <city>Milano</city>
</address>
```

- testo (*data content*)

```
<street>Via Chopin, 111</street>
```

- una loro combinazione (*mixed content*)

```
<name>Rosario Squillante</name> abita in <street>Via
Chopin</street> presso il civico <number>111</number> a
<city>Milano</city>…
```

Tra i diversi elementi, uno ricopre una particolare importanza. Si tratta dell'elemento ***document***, detto anche **etichetta radice** (*root tag*): è l'elemento più esterno, di conseguenza è quello che racchiude tutti gli altri elementi del documento. Esso deve esistere sempre.

Un discorso a parte meritano gli elementi vuoti (*empty*), ossia quelli senza contenuto. Essi

non presentano un *end tag*, per cui lo *start tag* ha una forma particolare:

```
<element_name … />.
```

Si noti la particolare posizione del carattere "/", che precede la chiusura dello *start tag*. In pratica, da un punto di vista sintattico si può immaginare che i tag di inizio e fine collassino in un'unica etichetta. Il vantaggio è che si occupa meno spazio.

Nella precedente scrittura, i puntini di sospensione in realtà rappresentano un segnaposto per qualcos'altro. Si tratta dei cosiddetti **attributi**, che andremo a definire tra breve.
Prima, ci sia consentita una piccola divagazione: se all'interno di un elemento vuoto non fossero presenti neppure degli attributi, il suo contenuto informativo sarebbe davvero scarso, e presumibilmente l'informazione potrebbe essere veicolata in modo più efficace ed efficiente utilizzando altri stratagemmi. Fa eccezione il caso in cui la semplice presenza dell'elemento costituisce tutto il significato informativo.
Ne vediamo subito un'applicazione nell'esempio sottostante, che rappresenta la descrizione di una nota, così come definita nel precedente capitolo.

```
<nota>
     <nome>Do</nome>
     <ottava>3</ottava>
     <alterazione>
          <diesis/>
     </alterazione>
</nota>
```

Anche se `<diesis>` è un elemento vuoto e privo di attributi, la sua sola presenza è sufficiente a veicolare l'informazione desiderata.

Gli *attributi* vengono utilizzati per aggiungere informazione ad un elemento.
Sono sempre associati allo *start tag*:

```
<element_name attribute_1="value_1" … attribute_N="value_N" >
…
```

```
</element_name>
```

Un elemento può avere un numero qualsiasi di attributi distinti.
Facciamo riferimento all'esempio sottostante. Quanto segue il nome dei tag all'interno delle parentesi angolari rappresenta un attributo. Dunque quantita è un attributo dell'elemento <ingrediente>, mentre luogo e minuti sono attributi di <cottura>.
Ogni attributo presenta un nome (che precede il segno "=") e un valore (scritto tra virgolette dopo il segno "="), ed è separato dal nome dell'elemento e dai successivi e/o precedenti attributi da uno spazio. Gli attributi sono posti all'interno delle parentesi angolari dello *start tag*.

```
<ricetta>
     <nome>Torta di mele</nome>
     <autore>Nonna Linda</autore>
     <ingredienti>
          <ingrediente quantita="200g">Margarina</ingrediente>
          <ingrediente quantita="1000g">Farina</ingrediente>
          <ingrediente quantita="4">Uova</ingrediente>
          …
     </ingredienti>
     <cottura luogo="forno" minuti="15"/>
     …
</ricetta>
```

Il problema di utilizzare elementi o piuttosto attributi per rappresentare l'informazione è sottile, e comunque non riguarda chi genera un file XML bensì chi progetta il corrispondente DTD. Nell'esempio della ricetta una volta stabilito che il DTD delle ricette prevede che minuti sia un attributo dell'elemento <cottura>, la persona (o il software) che codifica ricette secondo tale schema non ha possibilità di scelta.
In ogni caso, per completezza e chiarezza, vediamo quando nella progettazione dei DTD si tende a privilegiare un elemento e quando un attributo.

Si usa un **elemento** quando

- Necessito di una veloce ricerca
- Deve essere visibile a tutti
- E' importante per il significato del documento
- E' debolmente tipato

Si predilige un **attributo** quando

- E' una scelta
- E' visibile solo per il sistema
- Non e' importante per il significato del documento
- E' fortemente tipato

Evidenziamo infine che tra attributi esiste un rapporto gerarchico di parità. Dunque, se a un elemento si vogliono associare svariate informazioni di pari livello di importanza, queste saranno tutte attributi e nessuna rappresenterà il vero e proprio contenuto dell'elemento. Riprendendo l'esempio della ricetta, `<cottura>` è un elemento vuoto perché non esiste una caratteristica di cottura più importante delle altre; non sarebbe comprensibile il motivo per cui

```
<cottura minuti="15"> forno </cottura>.
```

Accettare (in un DTD) una scrittura del genere significa ammettere implicitamente che, nel definire la cottura, il luogo (contenuto dell'elemento) ha un'importanza preminente, mentre il tempo (attributo minuti) è un'informazione accessoria.
Diverso sarebbe il caso di

```
<luogo_di_cottura tempo="15">forno</luogo_di_cottura>
```

### *3.1.3 – altro*

In XML esistono altri "oggetti" tra cui ricordiamo:

- *processing instructions*, utilizzate principalmente per propositi di estensibilità e denotate con `<?target data?>`

- commenti, sintatticamente racchiusi dai caratteri <!-- e -->
- riferimenti a caratteri, quali £
- entità: file esterni o parti del documento cui si può fare riferimento ricorsivamente o da sezioni diverse nel documento stesso

### *3.1.4 – il DTD*

Come detto il *Document Type Definition* detta il tipo di un documento, cioè:

- i tag ammessi
- le regole di annidamento dei tag

Ció significa descrivere:

- Quali (sub-)elementi puó contenere un elemento
- Se puó contenere del testo o no
- Quali attributi contiene
- Tipizzazione e defaultizzazione degli attributi

Per far ciò, esso è diviso in due parti.

1. L'*Element Type Definition* specifica la struttura del documento, i contenuti consentiti (content model) e gli attributi consentiti (dal significato delle dichiarazioni delle liste di attributi).

2. L'*Attribute List Declaration* è la lista degli attributi permessi per ogni elemento. Ogni attributo è specificato da: *name, type*, e altre informazioni.

```
<!ELEMENT A (B*, C, D?)>
      "," = A può contenere gli elementi B, C e D, in questo ordine e con
la corrispondente
           cardinalità; significa sequence of
      cardinalità di default (non specificata) = 1 e solo 1
      cardinalità "?" = 0 o 1
```

```
      cardinalità "*" = 0 o più
      cardinalità "+" = 1 o più
<!ELEMENT A (B | C+)>
      "|" = A contiene o l'elemento B o l'elemento C; significa choice of
<!ELEMENT A (#PCDATA)>
      elemento di testo
<!ELEMENT A  EMPTY>
      elemento vuoto
<!ELEMENT A (#PCDATA| B | C)*>
      elemento misto
```

*Esempio di struttura di Element Type Definition*

```
<!ATTLIST ELEM attrib1 CDATA #IMPLIED>
      attrib1 è di tipo testuale e non è richiesto

<!ATTLIST ELEM attrib1 CDATA #IMPLIED attrib2 CDATA #REQUIRED>
      attrib1 e attrib2 sono di tipo testuale; il primo non è richiesto,
il secondo sì

<!ATTLIST ELEM attrib1 CDATA #IMPLIED "aaa">
      attrib1 è di tipo testuale, non è richiesto e ha valore di default
"aaa"

<!ATTLIST ELEM attrib1 CDATA #REQUIRED "aaa">
      attrib1 è di tipo testuale, è richiesto e ha valore di default "aaa"

<!ATTLIST ELEM attrib1 CDATA #FIXED "aaa">
      attrib1 è di tipo testuale e ha valore costante "aaa"

<!ATTLIST ELEM attrib1 (aaa|bbb) #IMPLIED "aaa">
      attrib1 è di tipo enumerato (valori elencati tra parentesi) e ha
valore di default "aaa"

<!ATTLIST ELEM id ID #REQUIRED>

      attrib1 è di tipo ID (assegnazione di un valore univoco)

<!ATTLIST ELEM ref IDREF #IMPLIED>
      attrib1 deve essere un riferimento a un ID
```

*Esempio di struttura di Attribute List Declaration*

I tipi di dato ammissibili per gli attributi sono i seguenti:

- `CDATA`: stringa
- `ID`: identificatore
- `IDREF, IDREFS`: valore di un attributo di tipo `ID` nel documento (o insieme di valori)
- `ENTITY, ENTITIES`: nome (nomi) di entità
- `NMTOKEN, NMTOKENS`: caso ristretto di `CDATA` (una sola parola o insieme di parole)

Sugli Attributi sono presenti i seguenti vincoli

- `#REQUIRED`: il valore deve essere specificato
- `#IMPLIED`: il valore può mancare
- `#FIXED` "valore": se presente deve coincidere con "valore"
- "valore": il valore può non essere specificato, nel qual caso si assume "valore" come default

Una volta specificato il DTD un file scritto in XML dovrà rispettarne la struttura (in questo caso il documento viene definito *valid*).
Alla validità si aggiunge la corretta forma (*well formed*) come già accennato in precedenza. Un documento XML la cui sintassi è corretta (cioè soddisfa le regole di scrittura di documenti XML) è detto ben formato.
Questo rigore è necessario in quanto il documento prodotto non può presentare ambiguità di interpretazione, soprattutto per un elaboratore!
In ogni caso esistono software specifici (*parser XML*) che controllano automaticamente le proprietà di validità e di giusta forma di un documento.

```
<!ELEMENT ELENCO (PRODOTTO+)>
<!ELEMENT PRODOTTO (DESCRIZIONE, PREZZO?)>
<!ELEMENT DESCRIZIONE #PCDATA>
<!ELEMENT PREZZO #PCDATA>
```

*Esempio di file DTD*

### *3.1.5 – XML Schema (XSD)*

Recentemente al DTD si preferendo l'*XSD* (*XML Schema Definition*) per la validazione dei documenti.
L'XSD non è altro che una particolare grammatica di XML, introdotta da Microsoft e definita nel 2001 dal W3C, pensata per descrivere la struttura ed il contenuto dei file XML.
Uno dei vantaggi principali, è che anch'esso un file XML (al contrario del DTD).
Quindi possiamo sia controllarne la validità con i parser disponibili, sia inserire porzioni di XSD nel documento XML stesso.

I vantaggi di XML Schema possono essere riassunti di seguito

– Sono estendibili (per future aggiunte)
– Sono anch'essi files XML
– Sono più ricchi e completi di DTD
– Supportano tipi di dati
– Gestiscono namespaces

Nel nostro lavoro di tesi tuttavia, useremo il classico DTD come riferimento dei files prodotti.

## *3.2 – La Musica e l'XML*

La flessibilità sopra descritta dell'XML, è una della caratteristiche che ha portato i ricercatori a chiedersi se fosse possibile usare tale linguaggio per descrivere la musica.
Sin dal 1991, l'IEEE[3] si occupa della computer music attraverso la sua sezione *Technical Committee on Computer Generated Music*. E dal 2001, il TC on CGM propose al SAB[4] un nuovo standard per le applicazioni musicali con l'uso del linguaggio XML.
L'idea era proprio quella di sfruttare la flessibilità e la portabilità dell'XML per standardizzare musica, simboli, suoni e partitura e renderli così fruibili per il mondo di internet e i nuovi media.
Inoltre, si cerava di metter fine agli annosi problemi degli attuali standard di fatto come MIDI, NIFF, WAV o MP3.
Problemi principalmente di incompletezza, inadeguatezza, traduzione e diritti d'autore[5].
Le pubblicazioni cominciate proprio nel 2001 miravano a dare linee guida comuni per le future descrizioni musicali attraverso l'XML. I vari formati ideati, le problematiche di ognuno, i punti di interesse, le criticità e così via.
Il risultato fu un gran numero di linguaggi XML musicali, ognuno con i suoi tag ma di solito strutturalmente simili. Ognuno di essi, si proponeva o risolveva solo però solo alcuni aspetti dell'informazione musicale.
Ad esempio molti di questi strutturavano la musica intesa come partitura. Altri, si occupavano di segmentare le parti audio di un pezzo musicale. Altri ancora, indicizzavano un brano musicale a seconda dell'autore, il genere e così via.
Di seguito ne vedermo alcuni, prima di concentrarci sul formato MX.

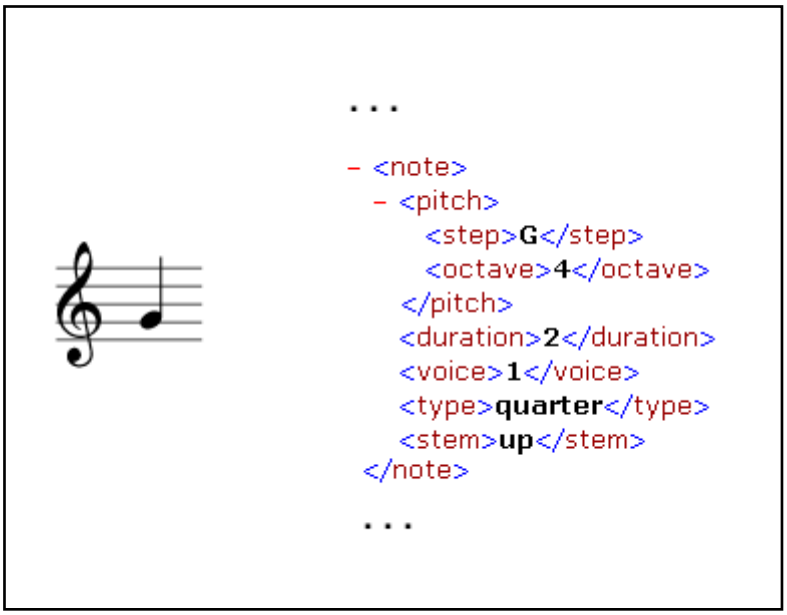
In ogni caso, i vantaggi di poter usare XML per la descrizione musicale sono evidenti e legati a doppio filo con l'obiettivo di XML stesso:

- E' un linguaggio di mark-up pensato per scrivere linguaggi di mark-up.
- Permette di definire una grammatica per descrivere dati associati ad una qualsiasi applicazione per computer.

[3] Insitute of Electrical and Electronic Enginers
[4] Standard Activity Board
[5] Ci riferiamo ovviamente alla descrizione dell'informazione musicale e non alla veicolazione del messaggio sonoro.

- E' studiato per l'interscambio.

A questo punto, la descrizione musicale si sposa bene con XML grazie alla sua struttura gerarchica ed alla possibilità di passare facilmente fra livelli di monoblocchi informativi a più fini e granulari collezioni di elementi descrittivi.
Per rendere l'idea [16], possiamo descrivere un brano musicale come l'opera X dell'autore Y e nello stesso tempo ricavare la seconda nota del flauto appartenente alla quinta battuta del terzo movimento. Che è poi la classica struttura ad albero. O se vogliamo essere più moderni, una struttura *object-oriented*.
Ciò permette un'altra operazione fondamentale. Quella di poterci muovere all'interno della struttura musicale e rappresentare la semantica di ogni elemento, sia esso semplice o complesso.
Se infatti apriamo un file XML con un qualsiasi browser, otterremo una visualizzazione del nostro documento fatta di elementi racchiusi tra parentesi "< >" che potranno contenere l'informazione o altri elementi, secondo la semantica e l'ordine gerarchico descritto nel DTD. Con la possibilità di poter espandere o comprimere i vari livelli di annidamento semplicemente cliccando sulla voce selezionata.




l'esempio mostra la codifica della nota sol, quarta ottava secondo *MusicXML*.

Il fatto di avere due tag per indicare la durata può sembrare eccessivo, invece è un esempio della flessibilità offerta dal linguaggio. Distinguiamo infatti, la parte di rappresentazione grafica (nel caso del valore, denotata dal tag `<type>`) da quella dell'esecuzione. Nessuno infatti, ci può impedire una minima ad una diversa velocità.
In quest'ultimo caso il valore da modificare è quello contenuto nel tag `<duration>`.
Ciò non toglie che la rappresentazione presentata nell'esempio in figura, possa apparire verbosa. Diciamo subito che possiamo ovviare in certi casi, ricorrendo all'uso degli attributi.
Ma ciò che più conta è che al contrario di altri formati, uno su tutti il MIDI, che vedremo nel prossimo capitolo, l'XML resta un formato leggibile anche dall'essere umano, a scapito di qualche riga di testo in più.
Ultimo, ma non meno importante, il poter contare *sul totally royalty-free*.
L'XML è un linguaggio free, può essere adottato da chiunque, le sue grammatiche restano a disposizione di tutti favorendone il costante e disinteressato ampliamento.
Chiaramente, una volta gettato il sasso, le soluzione proposte sono state e sono tuttora ancora svariate.
La codifica musicale con XML è stata fatta in tanti modi diversi.
Un elenco di tali linguaggi è facilmente reperibile in rete a questo indirizzo web:

www.oasis-open.org/cover/xmlMusic.html

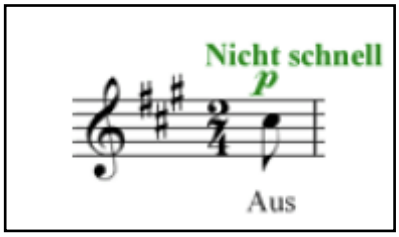
La mancanza tuttavia di uno standard consolidato porta all'inevitabile rovescio della medaglia. Fra i tanti sforzi che sono andati oltre al semplice esercizio accademico, molti di essi hanno davvero portato elementi di innovazione o acquisito visibilità commerciale ma rifacendosi ognuno al proprio schema o DTD, veniva a mancare una della caratteristiche basilari della scelta XML. Ovvero la portabilità e l'interscambio.
In ogni caso, possiamo di certo identificare alcuni linguaggi che più di altri si sono affermato in informatica musicale. Vediamone in una breve panoramica.

### *3.2.1 – Music XML [13]*

Si tratta di una delle soluzione più affermate. Sviluppato in ambito accademico, il suo sviluppatore, Michale Good, prese spunto dai formati di MuseData e Humdrum.
La partitura è vista da due punti di vista. La *Part-wise* e la *Time-wise*.

Ciò permette di controllare sia l'aspetto grafico dello spartito che l'esecuzione effettiva del brano.
Uno dei punti di forza di questo linguaggio, è stato il suo successo commerciale. In particolare, essere stato adottato da *MakeMusic Finale* e da *Sibelius*.
Questi due potenti software infatti, permettono di esportare i propri files in MusicXML, che diventa così una sorta di "Traduttore Universale".
La versione più recente è la 1.1 (2006).

Riportiamo un esempio completo di codifica MusicXML. L'immagine rappresenta ciò che viene rappresentato nel codice seguente.



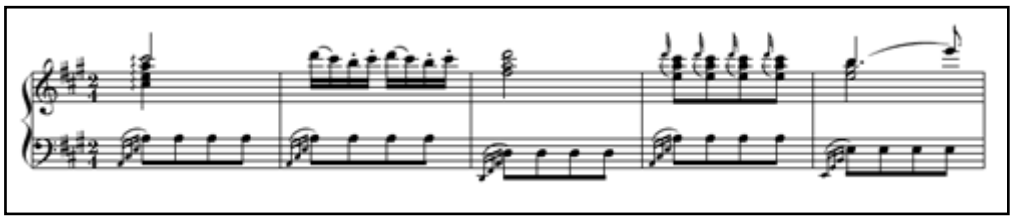

```
<part id="P1">
     <measure number="1" width="179">
          <attributes>
               <divisions>24</divisions>
               <key>
                    <fifths>3</fifths>
                    <mode>major</mode>
               </key>
               <time>
                    <beats>2</beats>
                    <beat-type>4</beat-type>
               </time>
               <clef>
                    <sign>G</sign>
                    <line>2</line>
               </clef>
          </attributes>
          <direction placement="above">
```

```
            <direction-type>
                <words default-y="25" font-size="10.5"
            font-weight="bold" relative-x="-42">Nicht
            schnell</words>
        </direction-type>
        <sound tempo="42"/>
        </direction>
        <direction placement="above">
            <direction-type>
                <dynamics default-y="10" relative-x="-5">
                    <p/>
                </dynamics>
            </direction-type>
            <sound dynamics="54"/>
        </direction>
        <note default-x="141">
            <pitch>
                <step>C</step>
                <alter>1</alter>
                <octave>5</octave>
            </pitch>
            <duration>12</duration>
            <voice>1</voice>
            <type>eighth</type>
            <stem default-y="-50">down</stem>
            <lyric default-y="-80" number="1">
                <syllabic>single</syllabic>
                <text>Aus</text>
            </lyric>
        </note>
    </measure>
```

Come si evince facilmente, le informazioni che servono a descrivere una singola nota, sono molteplici. Un documento che volesse descrivere un'intera opera lirica sarebbe di certo molto lungo. Volendo, potrebbero anche essere omesse alcune informazioni. Ma per il grado di astrazione sopra mostrato, (codificare la singola nota all'interno della partitura) esse ci danno una panoramica completa di ciò che può essere rappresentato. Notiamo che la nota viene descritta all'interno del tag *measure* (la battuta). Salendo ulteriormente questo livello di codifica, le battute saranno presenti all'interno dell'elemento *part*. Possiamo pensare ad esempio, ai due pentagrammi che compongono la partitura di pianoforte. Mano destra e mano sinistra sono considerate due part separate. Queste sono a loro volta contenute nella *part list* e così via, fino ad arrivare ad esempio al titolo della composizione.
Come si può notare, questo è esattamente il grado di scalabilità e precisione gerarchica che viene richiesto dagli studi sull'XML e la codifica musicale e si sposa perfettamente con quelle che sono le regole "classiche" della teoria musicale.

### *3.2.2 – MEI [14]*

Sviluppato da Perry Roland presso l'Università della Virginia il Music Encoding Iniziative deriva direttamente dal *TEI* (Text Encoding Initiative), un affermato standard per la codifica e la trasmissione di testi in formato elettronico.
Il MEI, tenendo fede alle sue origini, si propone come un formato mirato alla descrizione della musica nella sua forma scritta. Questo aspetto lo porta ad essere molto efficace per la catalogazione di brani musicali. Questo anche grazie alla possibilità di inserire nel documento analisi critiche e storiche dei pezzi rappresentati.
*MusiCAT* e *MDL* sono due formati nati seguendo le regole dettate nel MEI atti a questo scopo.
Nello stesso tempo, ciò lo limita un po' rispetto ad altre proposte più complete. Il vantaggio è quello di essere molto più leggibile ed asciutto rispetto ad esempio al MusicXML.
La mancanza inoltre di scopi commerciali lo rende, secondo l'autore, più propenso ad essere sviluppato con maggiore lungimiranza.
Proponiamo un esempio di codifica MEI per un estratto della sonata per pianoforte di Mozart in LA maggiore, op. K. 331.



```
- <mei version="1.7b">
  - <meihead>
      <meiid/>
    - <filedesc>
      - <titlestmt>
          <title>Piano Sonata in A Major, K. 331</title>
        - <respstmt>
            <agent type="composer">Wolfgang Amadeus Mozart</agent>
          </respstmt>
        </titlestmt>
        <pubstmt/>
      + <notesstmt></notesstmt>
      </filedesc>
    - <profiledesc>
      - <langusage>
          <language id="it"/>
        </langusage>
      </profiledesc>
    + <revisiondesc></revisiondesc>
    </meihead>
  - <work>
    - <music>
      - <mdiv>
        - <score>
          - <scoredef>
            + <staffgrp></staffgrp>
            </scoredef>
          - <section>
              <scoredef meter.count="2" meter.unit="4" key.sig="3s" key.mode="major"/>
            - <measure n="1" id="d1e31">
              - <staff def="1">
                - <layer def="1">
                    <note id="d1e58" tstamp="0" pname="c" oct="6" dur="2" dur.ges="8" stem.dir="up" acci.ges="s"/>
                  </layer>
                - <layer def="2">
                  - <chord id="d43e1" tstamp="0" dur="4" dur.ges="4" stem.dir="down">
                      <note id="d1e81" pname="c" oct="5" acci.ges="s"/>
                      <note id="d1e101" pname="e" oct="5"/>
                      <note id="d1e120" pname="a" oct="5"/>
                    </chord>
                    <space id="d1e139" tstamp="4" dur.ges="4"/>
                  </layer>
                </staff>
              + <staff def="2"></staff>
                <phrase tstamp="0" place="above" start="d1e149" dur="0m+0" end="d1e211" staff="2"/>
                <arpeg tstamp="0" plist="d1e58" staff="1" layer="1"/>
                <arpeg tstamp="0" plist="d1e81 d1e101 d1e120" staff="1" layer="2"/>
              </measure>
            + <measure n="2" id="d1e285"></measure>
            + <measure n="3" id="d1e614"></measure>
            + <measure n="4" id="d1e806"></measure>
            + <measure n="5" id="d1e1240"></measure>
            </section>
          </score>
        </mdiv>
      </music>
    </work>
  </mei>
```

### *3.2.3 – Altri linguaggi*

- MML (Music Markup Language)

Sviluppato da Jacques Steyn, esso comprende 12 moduli per rappresentare la musica nel suo più ampio spettro. I moduli del Tempo e della Frequenza racchiudono gran parte delle informazioni, gli altri vanno aggiunti in base alle necessità.
L'intento è quello di dare non solo una visione accurata di qualsiasi aspetto musicale, ma anche di permettere un controllo "remoto" dell'informazione musicale che si intende manipolare.
L'ambizioso progetto crea però non poche difficoltà. Sia per la sua complessità (qualcuno ha definito MML come l'SMDL musicale) che per le sue velleità di voler essere non uno standard per l'XML ma per la musica stessa.

- MPeG 7

Non ci riferiamo al formato di compressione audio, ma ai metadati presenti che sono rappresentati in XML. Più precisamente, attraverso XML Schema.
Vengono definiti vari aspetti dell'informazione multimediale e a vari livelli di astrazione.
Usando questo metodo è possibile ottenere informazioni su di un brano trasmettendo i pochi bytes che ne tengono traccia.
Tale metodo è usato soprattutto per scopi di Music Information Retrieval[6].

- Altri

- ChordML
- MusicML
- FlowML
- 4ML
- MusiXML
- MNML Musical Notational Markup Language

[6] consiste nell'interrogare archivi elettronici musicali attraverso domande focalizzate sul contenuto ("query by content"), con il ricorso a criteri non più basati su parole chiave (per esempio: titolo del brano musicale, nome del compositore, etc.), ma su descrizioni o astrazioni relative al contenuto musicale vero e proprio, quali la struttura armonica, la melodia o lo stile dell'esecutore.

## *3.3 – MX*

Se abbiamo attentamente letto la prima parte di questo capitolo, ci siamo resi conto di una cosa.
Non vi è ancora una grammatica XML che raccoglie nella maniere più efficace e completa l'informazione musicale, e quindi non è stato ancora ufficializzato uno standard.
Inoltre le grammatiche definite sono tante e ognuna con alterne fortune.

L'MX è stato sviluppato (e lo è tuttora!) da Maurizio Longari e Luca Ludovico presso il LIM del prof. Goffredo Haus dell'Università Statale di Milano.
L'intento è quello di rappresentare l'informazione musicale nella sua interezza, soffermandosi su i vari livelli di astrazione che possiamo dare alla musica.
Suddividendo quindi un ipotetico brano in più strati, possiamo concentrarci su ognuno di questi laddove ve ne sia bisogno e possibilità, e mantenendo il più possibile una struttura semplice, scalabile e modulare.
Per far questo, useremo dei *layers*, ognuno dei quali definirà un livello di astrazione.
Tale struttura è anche la base di partenza della nostra discussione.
La versione del DTD a cui ci riferiremo è la 1.7

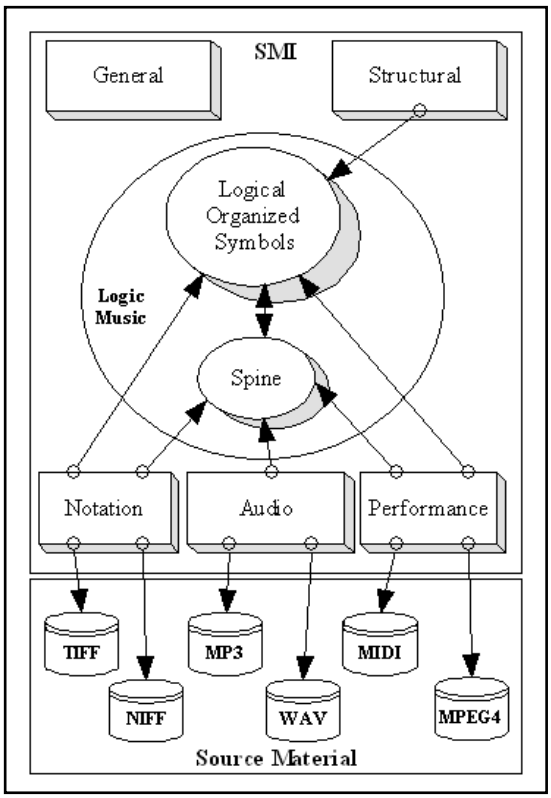
### *3.3.1 – layers MX*

Suddividiamo l'informazione musicale in sei strati.

1. **general**
2. **music logic**
    - **spine**
    - **los (logically organized symbols)**
    - **layout**
3. **structural**
4. **notation**
5. **performance**
6. **audio**

Essi rappresentano i possibili gradi di astrazione di un componimento. A livello informatico, esistono vari formati per ognuno di questi aspetti (mp3 per *audio*, MIDI per *performance*, NIFF per *notation* e così via).
Chiameremo *layer* ognuno di questi strati.
Il layer logic è composta a sua volta da 3 *sub-layers*.

La struttura che rappresenta la relazione spazio-temporale implicita nella musica si trova nello *spine*. Lo spine può essere visto come una funzione di mappatura bidimensionale tra i domini dello spazio e del tempo, e rappresenta in pratica un collante tra i diversi strati. Esso si compone di eventi, ciascuno dei quali presenta un riferimento nel dominio dello spazio e del tempo. Grazie all'uso dello spine, i differenti formati di file possono essere messi in relazione per ottenere una descrizione completa dell'informazione musicale.
Lo spine è il cuore dell'MX: tutti i layer fanno riferimento ad esso, in una sorta di struttura a stella, a eccezione del layer *general*.
Un brano musicale può dunque essere descritto nella sua interezza (oggetti musicali, scansioni della partitura, esecuzioni sintetizzate, registrazioni,…), e questo porta indubbi benefici nel campo dell'analisi e dell'organizzazione dell'informazione, in quanto costringe a una sua visione strutturata.



Da un punto di vista tecnico, ogni layer rappresenta un elemento figlio del tag radice.
Nel DTD, troviamo:

```
<!ELEMENT mx (general, logic, structural?, notational?,
performance?, audio?)
```

Quindi il nostro file .MX sarà strutturato nel seguente modo

```
<mx>
     <general>…</general>
     <logic>…</logic>
     <structural>…</structural>
     …
</mx>
```

Notiamo dalle cardinalità che non tutti gli strati sono richiesti: solo l'identificazione generale e i dati sulla "logica" del brano devono essere necessariamente presenti. Per quanto concerne gli altri layer, essi o sono presenti o non lo sono; la loro cardinalità non è mai superiore a 1.

#### *3.3.1.1 – il layer general*

Il layer *general* prende in considerazione il brano musicale nella sua interezza, dandone un inquadramento generale e raggruppando le informazioni sulle relative istanze.

```
<!ELEMENT general (description, casting?, related_files?,
analog_media?, notes?, rights?)>
```

La parte più importante ai fini dell'identificazione di un brano è quella relativa all'elemento *description*, di cui ora ci occuperemo in dettaglio.

```
<!ELEMENT description (work_title?, work_number?,
movement_title, movement_number?, genre?, author+)>
```

```
<!ELEMENT work_title (#PCDATA)>
```

è un elemento testuale che contiene il titolo dell'opera complessiva (ad esempio, il nome della sinfonia di cui l'MX rappresenta un solo movimento, o il nome dell'album da cui è tratta la canzone considerata).

```
<!ELEMENT work_number (#PCDATA)>
```

consente di attribuire un numero d'ordine o di catalogo all'opera complessiva da cui è tratto il brano considerato.

```
<!ELEMENT movement_title (#PCDATA)>
<!ELEMENT movement_number (#PCDATA)>
```

sono l'equivalente di *work_title* e *work_number*, ma volti alla descrizione del brano in esame e non alla raccolta da cui il brano è tratto.

```
<!ELEMENT author (#PCDATA)>
<!ATTLIST author type CDATA #IMPLIED>
```

sono destinati a contenere il nome o i nomi degli autori. La cardinalità è +, il che significa che ci deve essere almeno un autore ma se ne possono indicare più d'uno. L'attributo type consente di specificare il tipo di autore per il brano considerato. Ad esempio, sono ammessi type = "compositore", type = "librettista", type = "revisore" e via dicendo…

```
<!ELEMENT genre (#PCDATA)>
```

è un elemento testuale riservato alla descrizione del genere musicale.

Per quanto riguarda gli altri elementi di general, a parte description troviamo:

```
<!ELEMENT casting EMPTY>
```

è un elemento – al momento non definito – riservato alle informazioni sul cast e sugli esecutori. Può essere presente o meno.

```
<!ELEMENT related_files EMPTY>
```

è un elemento – al momento non definito – riservato alla descrizione dei file dati collegati alla musica, in riferimento a tutti i layer. Rappresenta l'elenco dei file multimediali coinvolti a corredo dell'informazione musicale (immagini, video,…). La sua presenza è opzionale.

```
<!ELEMENT analog_media EMPTY>
```

comprenderà le informazioni relative ai media analogici, compresa l'importazione/esportazione, il restauro, la catalogazione ecc. ecc. Non è ancora stato implementato e la sua presenza non è richiesta.

```
<!ELEMENT notes (#PCDATA)>
```

è un elemento opzionale di tipo testo riservato ad annotazioni generiche.

```
<!ELEMENT rights (#PCDATA)>
```

costituisce con tutta probabilità l'aspetto più complesso da trattare. Si tratta di specificare i diritti di proprietà intellettuale e le informazioni sulla sicurezza dell'intero pezzo. Questi aspetti devono essere dettagliati non solo sul brano in generale, ma anche sui singoli file origine referenziati nell'MX. Elemento attualmente in fase di sviluppo.

Per fissare le idee, mostriamo alcuni esempi di layer general all'interno di file MX:

Es. 1

*Ludwig van Beethoven*
*Lied n° 5 – "Gottes Macht und Vorsehung"*
*su testo di Christian Fürchtegott Gellert*

```
<mx>
     <general>
          <description>
               <movement_title>Gottes Macht und
Vorsehung</movement_title>
               <movement_number>5</movement_number>
               <genre>classical lied</genre>
               <author type="composer">Ludwig van
Beethoven</author>
               <author type="poet">Christian Fürchtegott
Gellert</author>
          </description>
```

```
     </general>
     …
</mx>
```

Es. 2

*The Beatles*
*"The long and winding road" (Writer, lead vocal: Paul McCartney)*
*traccia 4 di "Let it be… naked"*

```
<mx>
     <general>
          <description>
               <work_title>Let it be… naked</work_title>
               <movement_title>The long and winding
road</movement_title>
               <movement_number>4</movement_number>
               <genre>pop</genre>
               <author type="performers">The Beatles</author>
               <author type="writer">Paul McCartney</author>
          </description>
     </general>
     …
</mx>
```

E' possibile, come sappiamo, che una canzone possa comparire su più album (pensiamo a tutte le compilation che ogni anno vengono stampate sotto le feste…). Ciò viene considerato nel layer *audio*, di cui si parlerà più avanti.
A livello general, se si vuole includere l'informazione work_title sarebbe bene considerare l'opera originale di cui la canzone fa parte.
Comunque, la compilazione degli elementi dal punto di vista dell'MX è libera: in pratica, è necessario rispettare la sintassi, ma non viene effettuato alcun controllo sulla semantica. Dunque, scrivere One come work_title non darebbe errore nell'MX (il file risulterebbe comunque valido); purtroppo, anche scrivere "The dark side of the moon" o "Sinfonia n° 3

eroica" come work_title, o invertire le informazioni sul nome della canzone e il nome dell'album non avrebbe generato errore: si sarebbe trattato di una compilazione errata dal punto di vista del significato, ma sintatticamente corretta.

#### *3.3.1.2 – il layer Logic*

Il layer *logic* presenta tre sottolivelli (elementi figli):

1. *spine* contiene la funzione di mappatura spazio-temporale
2. *los* descrive gli oggetti in partitura
3. *layout* si occupa di rappresentare il layout del brano

Chiaramente, spine e los sono fondamentali per un file MX: la loro presenza è dunque richiesta.
In particolare, los contiene le informazioni ricavabili dalla partitura intesa come insieme di oggetti musicali (note, pause, segni di articolazione,...), mentre ignora gli aspetti di layout (margini, impaginazione,...) cui è dedicato un elemento apposito.
Il layer spine è fondamentale data la natura multi-livello dell'MX: senza lo spine, che funge da collante tra i vari livelli di rappresentazione, la codifica MX stessa non avrebbe significato.

- **Lo spine**

L'elemento spine rappresenta la struttura logica che implementa l'integrazione dei layer *notational*, *performance* e *audio* con il *music logic*. Il suo obiettivo è costruire una struttura astratta cui fanno riferimento tutti gli strati che descrivono le proprietà del materiale originario.
L'utilità dello spine deriva dalle differenti codifiche adottate per l'informazione musicale che rappresenta il materiale di base; si rende dunque necessario un punto di riferimento unico per tutte le istanze appartenenti a layer diversi (ad esempio, file MIDI, TIFF e WAVE) o anche allo stesso layer (ad esempio, gli MP3 con esecuzioni differenti della stessa partitura).
Due eventi fisici diversi (quali due note) in due istanze distinte (quali due file che contengono la registrazione dello stesso pezzo) possono così far riferimento allo stesso

evento logico, ossia allo stesso simbolo in partitura. Analogamente, l'immagine scansionata di una porzione di partitura e l'audio della sua esecuzione possono essere emanazioni degli stessi eventi logici.

Nello spine è necessario identificare ed etichettare tutti gli eventi "significativi" in partitura. Ma cosa si intende per evento significativo? Lo definiremo come qualsiasi segno di cui si voglia tener traccia nell'MX. La scelta più banale è quella di prendere in considerazione tutte le note e le pause, in quanto si tratta degli elementi atomici del linguaggio musicale. Ma questa scelta, per quanto sensata, non è l'unica.

Innanzi tutto, il campo di elementi da contrassegnare può essere arbitrariamente esteso. Ad esempio, segnature di chiave e di tempo, armature di chiave e cambiamenti di tonalità, segni di agonica e di articolazione e via dicendo possono essere ritenuti significativi e dunque aggiunti allo spine.

In modo analogo, lo spine può anche essere arbitrariamente ridotto. Se di ogni pezzo interessano solo alcune note (ad esempio le prime di ogni battuta), o solo il testo cantato e via dicendo, lo spine non dovrà necessariamente includere le informazioni ritenute inutili.

In conclusione, tutto quello cui si vuole far riferimento nella codifica MX deve avere una riga corrispondente nello spine.

Vediamo come si presenta lo spine in generale:

```
...
     <general>
          …
     </general>
     <logic>
          <spine>
               <event id="evento0" timing="NULL" hpos="NULL"/>
               <event id="evento1" timing="0" hpos="6"/>
               <event id="evento2" timing="1000" hpos="0"/>
               <event id="evento3" timing="0" hpos="0"/>
               <event id="evento4" timing="2000" hpos="6"/>
          </spine>
          …
     </logic>
...
```

Lo spine definisce un ordinamento stretto tra eventi. Nel seguito della codifica MX, la descrizione degli eventi può anche essere disordinata (ad esempio, la nota 25 di una voce può essere descritta prima della nota 1 nella partitura), perché comunque lo spine ricrea l'ordinamento. Si tratta ovviamente di una pratica deprecabile, ma consentita dalla sintassi.
L'attributo *id* assegna un identificativo univoco all'evento: id duplicati costituiscono errore e il file MX non risulta valido. Gli id possono essere scelti arbitrariamente: "pippo", "pluto" e "topolino" sono nomi validi esattamente quanto "evento_0", "evento_1" ed "evento_2". E' però consigliabile effettuare scelte di chiara decifrazione, per cui una numerazione degli eventi e una loro migliore organizzazione anche a livello di nomi è auspicabile. Ad esempio, avendo a che fare con un pezzo polifonico complesso, si potrebbe decidere di etichettare gli eventi con nomi che non solo li distinguano temporalmente ma anche li denotino voce per voce: ad esempio "tromba_ev0",…, "tromba_ev100", "flauto_ev0",…,"flauto_ev38", e via dicendo.
L'attributo *timing* costituisce una temporizzazione virtuale in *VTU* (virtual timing units). Esistono delle norme per scegliere l'unità di misura, che non è fissata: i VTU non sono millisecondi, decimi o quarti d'ora, ma l'utente sceglie che significato temporale attribuire ai VTU di volta in volta. Nella scelta del VTU di riferimento è consigliabile utilizzare valori grandi, divisibili per molti divisori e potenze di due per i valori più lunghi. Ad esempio, in un corale scritto sostanzialmente a quarti si può assegnare il valore 1024 al quarto. Le potenze di due consentono di rappresentare molti valori con numeri interi (1024 = semiminima, 512 = croma, 256 = semicroma,…); un valore base divisibile per 3, per 5, per 7 consente di rappresentare con numeri interi le note che costituiscono terzine, quintine, settimine. In ogni caso, si tratta di una scelta del tutto arbitraria. La valutazione si deve basare anche su considerazioni di granularità degli eventi da descrivere. Se abbiamo a che fare con un corale di Bach in cui si usano solo metà, quarti e ottavi, si può effettuare l'assegnamento:

croma = 1, semiminima = 2, minima = 4.

La granularità risultante è molto grezza: nessun valore intero potrebbe rappresentare ad esempio un ottavo con il punto, in quanto la sua durata sarebbe di 1,5. Però in certi casi semplici potrebbe essere adeguata anche una scelta così rinunciataria.

Un discorso analogo a quello fatto per il timing vale per l'*hpos*, che però tiene conto della spazializzazione orizzontale (sempre virtuale) anziché della temporizzazione. La partitura va concepita come nella rappresentazione *Score view* di Finale, ossia tutta di seguito e senza ritorni a capo. Ancora una volta, l'unità di misura può essere arbitrariamente stabilita[7].

Entrambi gli attributi timing e hpos supportano i valori numerici nulli, interi positivi e il valore speciale NULL, che va inserito quando non ha senso parlare di temporizzazione o di spazializzazione. Ad esempio, la temporizzazione dell'armatura di chiave non è in generale determinabile né significativa!
Precisiamo che gli attributi timing e hpos sono relativi e non assoluti. Quindi ogni valore fa riferimento a quello immediatamente precedente. La contemporaneità spaziale e/o temporale viene denotata dal valore 0. Nell'esempio riportato precedentemente, `evento0` non ha spazializzazione né temporizzazione; `evento1` occorre nell'istante 0, `evento2` dopo 1000 VTU, `evento3` è contemporaneo a `evento2`, ed `evento4` dopo 2000 VTU rispetto a `evento2` ed `evento3`. Un discorso analogo vale per lo spazio.
Si noti che il valore di timing e hpos non dipende dalla figura musicale stessa, ma da quella precedente. Si consideri la sequenza A-B-C-D, con A, C e D ottavi e B quarto, attribuendo agli ottavi 200 VTU e ai quarti 400 VTU: $timing_A = 0$, anche se la nota A è un ottavo; $timing_B = 200$, anche se la nota B è un quarto (infatti, la nota A che precede B è di un ottavo); $timing_C = 400$, dato che B è un quarto; $timing_D = 200$, dato che C è un ottavo. Stesso discorso vale per gli hpos.

- **los (Logical Organized Symbols)**

In questo sub-layer si trovano le informazioni simboliche di partitura. In altre parole, il contenuto informativo simbolico del brano (note, pause, segni di articolazione, dinamiche,…) è interamente racchiuso nell'elemento.
Data l'importanza e la vastità dell'argomento, esso sarà trattato in modo completo nel paragrafo 3.3.2.

[7] talvolta è comodo adottare la cosiddetta spaziatura in base al tempo, in cui ad esempio valori doppi occupano il doppio dello spazio. Ciò consente infatti di poter copiare quasi sempre i valori della colonna timing nella colonna hpos, sebbene dal punto di vista semantico il significato della temporizzazione sia ben diverso da quello della spazializzazione orizzontale. Perché "quasi" sempre? Perché in determinati casi la spaziatura orizzontale non rispetta comunque quella temporale: un caso tipico si verifica dopo le stanghette di battuta, che comportano una maggiore distanza della nota da quella precedente

- **layout**

L'informazione musicale codificata nel modo finora descritto consente grande flessibilità anche nel rendering visuale. Infatti, è facile definire elementi e proprietà che, sulla base dell'informazione contenuta nello spine e nel logic, organizzino i simboli musicali su diversi media, a partire dalla carta stampata nei vari formati alle differenti risoluzioni degli schermi dei PC. Questo perché lo spine funge da ponte consentendo riferimenti indiretti dal layout (o meglio, dalle diverse istanze in layout) ai simboli in partitura. Simboli e segni non convenzionali possono essere modellati facendo uso dell'SVG (Support Vector Graphics).

#### *3.3.1.3 – layer structural*

Il layer *structural* si occupa di descrivere le relazioni interne al brano in esame: temi musicali, soggetti, sequenze o segmenti che si ripetono, o che presentano un particolare interesse.
Le informazioni qui racchiuse sono il frutto arbitrario di analisi di carattere musicologico, svolte manualmente o automaticamente.
Nel layer structural troviamo in pratica il risultato di una segmentazione, o la rappresentazione dell'informazione musicale basata ad esempio su *Reti di Petri*.
Attualmente i formalismi consentiti, come si evince dal DTD sono i seguenti:

```
<!ELEMENT structural (chord_grid*, analysis*, petri_nets*,
mir*)>
```

#### *3.3.1.4 – layer notational*

Il layer *notational* si riferisce alle istanze "visive" di un pezzo musicale.
Sono state individuate due modalità di scrittura e lettura della musica: quella notazionale (*notational*) e quella grafica (*graphical*). Un'istanza notazionale è spesso in formato binario, come in NIFF o in ENGIMA, e rappresenta informazione simbolica. Altre istanze di questo tipo in formati XML alternativi sono: MusicXML di M. Good, Music Encoding Initiative (MEI) di P. Roland e Music Markup Language di J. Steyn. Un'istanza grafica, invece, contiene immagini che rappresentano la partitura. Anch'essa solitamente si presenta in formato binario (ad esempio, file JPEG, TIFF o PDF), ma si può anche trattare di immagini

vettoriali codificate in SVG (e quindi in formato testuale senza riferimenti espliciti alla musica). L'informazione di questo layer si lega alla parte spaziale dello spine, il che ne consente la corretta localizzazione.
Riportiamo qui sotto quanto si trova nel DTD dell'MX RC1. Si nota una forte similitudine tra quanto riguarda un'istanza notazionale e una grafica (stessi attributi, stessi elementi figli). Pertanto, la differenza nell'uso dell'una o dell'altra è puramente logica.

```
<!ELEMENT notational (graphic_instance_group |
notation_instance_group)+>

<!ELEMENT graphic_instance_group (graphic_instance+)>
<!ATTLIST graphic_instance_group
     description CDATA #REQUIRED>

<!ELEMENT graphic_instance (graphic_event+, rights?)>
<!ATTLIST graphic_instance
  description CDATA #IMPLIED
     position_in_group CDATA #REQUIRED
     file_name CDATA #REQUIRED
     file_format %formats; #REQUIRED
     encoding_format %formats; #REQUIRED
     measurement_unit (centimeters | millimeters | inches |
decimal_inches | points | picas | pixels | twips) #REQUIRED>

<!ELEMENT graphic_event EMPTY>
<!ATTLIST graphic_event
     %spine_ref;
     upper_left_x CDATA #REQUIRED
     upper_left_y CDATA #REQUIRED
     lower_right_x CDATA #REQUIRED
     lower_right_y CDATA #REQUIRED
     highlight_color CDATA #IMPLIED
  description CDATA #IMPLIED>
```

```
<!ELEMENT notation_instance_group (notation_instance+)>
<!ATTLIST notation_instance_group
     description CDATA #REQUIRED>

<!ELEMENT notation_instance (notation_event+, rights?)>
<!ATTLIST notation_instance
  description CDATA #IMPLIED
     position_in_group CDATA #REQUIRED
     file_name CDATA #REQUIRED
     format CDATA #REQUIRED
  measurement_unit CDATA #REQUIRED>

<!ELEMENT notation_event EMPTY>
<!ATTLIST notation_event
  %spine_ref;
  start_position CDATA #REQUIRED
  end_position CDATA #REQUIRED
  description CDATA #IMPLIED>
```

Cominciamo dicendo che possiamo avere diverse istanze dei due elementi. Che saranno raggruppate da `graphic_instance_group` e `notation_instance_group`. La descrizione fungerà da identificativo, così come la sua posizione all'interno del proprio gruppo di appartenenza.
Notiamo che, come è abbastanza ovvio, sia necessario specificare il nome del file e il suo formato (ad esempio, file_name = "c:\partiture\bach.jpg" format = "JPEG").
Nell'elemento singolo, è necessario il riferimento allo spine.
Questo perché ogni file di notazione potrebbe riferirsi non all'intera partitura ma a una sua porzione. Pensiamo alla scansione delle usuali partiture, la cui lunghezza supera la pagina: ogni foglio, in genere, viene salvato in formato digitalizzato in un file differente. Questi attributi consentono anche di linkare all'MX non partiture complete ma singole parti strumentali. Ancora una volta notiamo la centralità del layer spine, cui si fa riferimento per

individuare sezioni e sottosezioni della composizione e per ottenerne una corretta collocazione spazio-temporale.
Da notare infine, la presenza del campo `rights`.

#### *3.3.1.5 – layer performance*

Il layer *performance* si colloca tra gli strati notational e audio. I formati di file che sono supportati da questo layer codificano i parametri delle note da eseguire (così come avviene nel precedente layer notational) e i parametri dei suoni da creare (così come avviene nel successivo layer audio), ma sempre al fine di una produzione musicale sintetica, ossia da parte dell'elaboratore.
Performance abbraccia i formati MIDI, CSound, SASL/SAOL (MPEG4) e potrebbe essere ulteriormente esteso.

Analizziamolo nel DTD

```
<!ELEMENT performance (midi_instance | csound_instance |
mpeg4_instance)+>

<!ELEMENT midi_instance (midi_mapping+, rights?)>
<!ATTLIST midi_instance
  file_name CDATA #REQUIRED
     format (0 | 1 | 2) #REQUIRED>
…
```

Il layer in questione dunque si compone di una o più rappresentazioni in forma subsimbolica. La trattazione del MIDI merita un discorso a parte, dato che l'elemento è più ricco di informazioni rispetto al CSound e all'MPEG. Visto che si tratta di attributi ed elementi figli estremamente intuitivi per chi conosca la codifica MIDI, verranno riportati senza commenti.

```
<!ELEMENT midi_mapping (midi_event_sequence+)>
<!ATTLIST midi_mapping
     part_ref IDREF #REQUIRED
     voice_ref IDREF #IMPLIED
```

```
     track CDATA #REQUIRED
     channel CDATA #REQUIRED>

<!ELEMENT midi_event_sequence (midi_event | sys_ex)+ >
<!ATTLIST midi_event_sequence
     division_type  (metrical | timecode) #REQUIRED
     division_value NMTOKEN #REQUIRED
  measurement_unit (ticks | sec) #REQUIRED>

<!ELEMENT midi_event (%MIDIChannelMessage;)*>
<!ATTLIST midi_event
     timing CDATA #REQUIRED
  %spine_ref;>

<!ELEMENT sys_ex (SysEx)>
<!ATTLIST sys_ex
     %spine_ref;>
```

#### *3.3.1.6 – layer audio*

Il layer audio descrive le proprietà del materiale musicale audio. Si tratta del livello più basso nella nostra visione multi-strato.
I formati per la rappresentazione dell'informazione audio si possono suddividere in due categorie: compressi e non compressi. Tra questi ultimi troviamo PCM/WAV, AIFF e MuLaw. A loro volta, i formati compressi possono presentare perdita informativa (lossy) o meno (lossless). Tra i formati compressi con perdita ricordiamo MPEG e DOLBY, tra quelli senza perdita ADPCM e SHN.
Per poter collegare un generico file audio allo spine, è necessario estrarre le caratteristiche degli eventi musicali relative alla loro localizzazione temporale. Si tratta di informazioni che non dipendono dal particolare formato di codifica o dall'eventuale compressione.
Questo layer ha subito profondi cambiamenti nel corso delle varie versioni del DTD.
Attualmente è così definito:

```
<!ELEMENT audio (track+)>
```

```
<!ELEMENT track (track_general?, track_indexing?, rights?)>
<!ATTLIST track
     file_name CDATA #REQUIRED
     file_format %formats; #REQUIRED
     encoding_format %formats; #REQUIRED
  md5 CDATA #IMPLIED>
```

Il quale a sua volta contiene due sub layer

- **General**

```
<!ELEMENT track_general (recordings?, genres?, albums?,
performers?, notes?)>
<!ATTLIST track_general
     geographical_region CDATA #IMPLIED
     lyrics_language CDATA #IMPLIED>

<!ELEMENT recordings (recording+)>

<!ELEMENT recording EMPTY>
<!ATTLIST recording
     date CDATA #REQUIRED
     recorded_part CDATA #IMPLIED
     studio_name CDATA #IMPLIED
     studio_address CDATA #IMPLIED>

<!ELEMENT albums (album+)>

<!ELEMENT album EMPTY>
<!ATTLIST album
     title CDATA #REQUIRED
  track_number CDATA #IMPLIED
     carrier_type CDATA #IMPLIED
```

```
     catalogue_number CDATA #IMPLIED
     number_of_tracks CDATA #IMPLIED
     publication_date CDATA #IMPLIED
     label CDATA #IMPLIED>

<!ELEMENT performers (performer+)>

<!ELEMENT performer EMPTY>
<!ATTLIST performer
     name CDATA #REQUIRED
     type CDATA #REQUIRED>
```

- **Indexing**

```
<!ELEMENT track_indexing (track_region*, track_event+)>
<!ATTLIST track_indexing
     timing_type (samples | time | seconds | time_frames |
frames | measures | smpte_24 | smpte_25 | smpte_2997 |
smpte_30) #REQUIRED>

<!ELEMENT track_region EMPTY>
<!ATTLIST track_region
     name CDATA #REQUIRED
     description CDATA #IMPLIED
  %spine_start_end_ref;>

<!ELEMENT track_event EMPTY>
<!ATTLIST track_event
     start_time CDATA #REQUIRED
     end_time CDATA #IMPLIED
  %spine_ref;
     description CDATA #IMPLIED>
```

Da come si evince, è un layer molto più ricco di quanto possa sembrare. Questo perché gli ultimi formati di file audio portano con sé molte informazioni aggiuntive oltre alla classica ”forma d’onda”.

### *3.3.2 – header file MX*

Ogni file MX ben formato deve cominciare con le seguenti righe

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mx SYSTEM "http:\www.lim.dico.unimi.it\mx\mx.dtd">
```

la prima riga definisce la versione XML adottata e il tipo di codifica dei caratteri Unicode. La seconda linea consente di far riferimento al DTD ufficiale dell’MX, costantemente aggiornato e depositato sul server del LIM all’indirizzo:

http://www.lim.dico.unimi.it/MX/mx.dtd

Ovviamente, se si lavorasse in locale, basterebbe copiare il DTD in una qualsiasi cartella e cambiare l’indirizzi dell’instestazione.

Inoltre, deve essere presente il numero della versione del DTD usata.

```
<mx version="1.7">
```

Il file MX è un file di testo, quindi può essere visualizzato con un qualsiasi lettore di documenti.
Ancor meglio sarebbe usare un browser quale Internet Explorer o Mozilla Firefox per poter gestire le strutture nidificate.

### *3.3.3 – partiture in MX*

Come detto in precedenza la partitura vera e propria viene codificata all’interno del sub-layer los.

```
<general>
…
</general>
<logic>
     <spine>
     …
     </spine>
     <los>

          PUNTO DI INSERIMENTO DELLA PARTITURA

     </los>
</logic>
```

Da ora in poi e per tutta la durata del paragrafo in questione, ci occuperemo solo di questo livello e dei suoi elementi figli.

#### *3.3.3.1 - strutture gerarchiche*

Come detto, l'XML ben si presta a rappresentare dati strutturati; potremmo dire che addirittura intrinsecamente richieda la strutturazione dell'informazione. Perciò è necessario stabilire una gerarchia delle informazioni nella partitura. Sembra logico affermare che il livello più alto (ossia il più astratto) sia quello costituito dall'intera partitura, mentre quello più basso (ossia il più elementare) sia rappresentato dai simboli quali note e pause. E in mezzo come strutturiamo l'informazione?
La risposta dell'MX è la seguente:

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.

Riportiamo ora alcune osservazioni su quanto detto.

- La gerarchia pentagramma > parte > voce in generale si presta alle situazioni riscontrabili in letteratura. Un caso semplice è quello in cui ciascun pentagramma contiene esattamente una parte e la parte si compone di un'unica voce. Ad esempio, la linea vocale del solista è associata a un ben preciso pentagramma e confinata ad esso. Per venire a casi più complessi, pensiamo a una situazione in cui un pentagramma contiene più parti: l'esempio che sovviene è quello dei tre corni in "Pierino e il lupo" di *Prokofiev*. In un pentagramma possono poi trovare posto più voci, tutte riconducibili ad un'unica parte: la parte del pianoforte in una fuga per tastiera può contemplare tre o più voci. Fin qui abbiamo riportato casi supportati dall'MX. Purtroppo, non mancano in letteratura controesempi problematici: ad esempio, la migrazione di una voce da un pentagramma a un altro (come avviene in numerosi pezzi pianistici per le voci interne); o il ribaltamento della gerarchia sopra descritta, nel senso che non è il pentagramma a contenere più parti bensì una parte ad abbracciare più pentagrammi (come è consueto nella musica per tastiera, arpa, ecc.). Vedremo nel seguito come gestire tali casi particolari.
- Le battute sono descritte parte per parte: prima i contenuti delle misure da 1 a n della parte 1 (voce 1, voce 2,…), poi quelli delle misure da 1 a n della parte 2 (voce 1, voce 2,…), e così via per le altre parti… Dunque i rapporti di sovrapposizione verticale, o di sincronizzazione temporale, a livello di partitura non sono evidenti "a occhio nudo", al contrario di quanto avviene in una partitura stampata o manoscritta. Ma questo non dovrebbe rappresentare un problema, dato che le finalità dell'MX sono di codifica dell'informazione e non di presentazione. Sarà poi un eventuale editor o viewer a ri-trasformare l'informazione simbolica in modo fruibile per il musicista.
- Per via della posizione gerarchica in cui il termine compare, per accordo si intende la sovrapposizione di note all'interno di una parte e di una voce specifica. Perciò in MX è un accordo quello che esegue la mano sinistra del pianoforte come conclusione di molti pezzi, e non quello che scaturisce dall'esecuzione contemporanea di linee monodiche differenti, ad esempio in un mottetto o in un corale.

- Si osservi che la singola nota, ossia la nota che in una data voce non si sovrappone verticalmente ad alcun altro suono appartenente alla voce stessa, viene vista e codificata come accordo (degenere). Si tratta di una situazione assai comune nella musica.

Quanto detto genera il seguente scheletro di MX

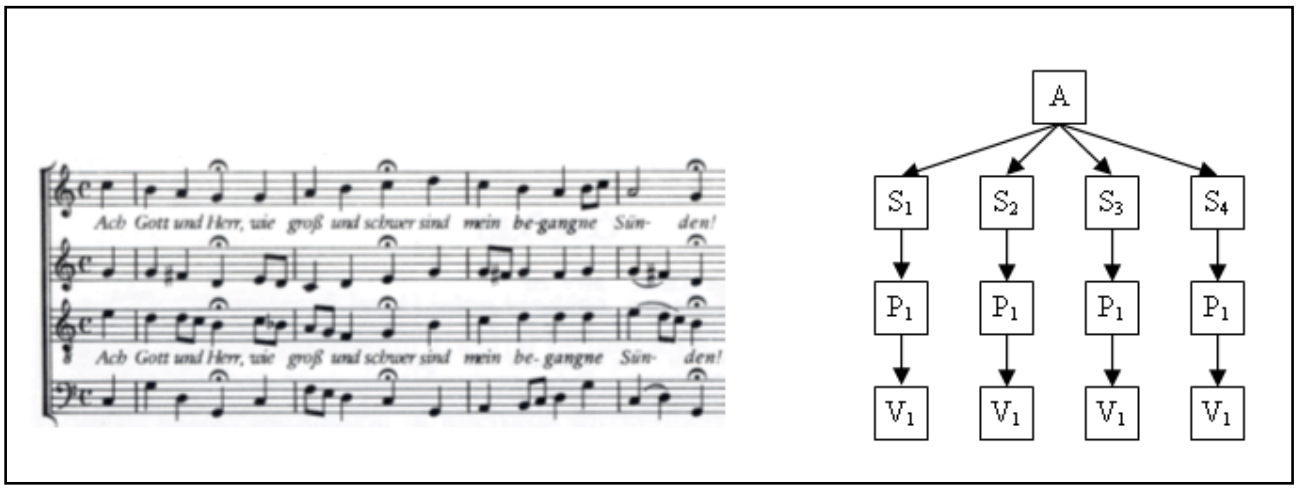
```
<los>
     <!-- descrizione dell'accollatura -->
     <staff_list>
          <!-- descrizione delle proprietà dei pentagrammi -->
          <staff id="staff_1">     … </staff>
          <staff id="staff_2">     … </staff>
     </staff_list>
     <!-- descrizione delle singole parti -->
     <part id="flauto" dfstaff_ref="staff_1">
          <!--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" dfstaff_ref="staff_2">
          <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>
```

ove *staff_list* è l'accollatura (ossia l'elenco degli staff), *staff* è il pentagramma, *part* è la parte, *voice_list* è l'elenco delle voci in cui si divide la parte e *voice_item* è la singola voce. Notiamo una differenza rispetto a quanto detto: prendendo in considerazione il livello di indentazione delle parti (part), scopriamo che è pari a quello dell'accollatura (staff_list). Questo significa che le parti, pur essendo in teoria associate ai pentagrammi e trovandosi a livello gerarchico più basso, non vengono indentate al di sotto di staff. In ogni caso, la corrispondenza tra pentagrammi e parti viene ricostruita immediatamente, grazie all'uso dell'attributo *dfstaff_ref*, il cui valore è l'identificatore precedentemente definito per una delle parti.
Altra cosa degna di nota è proprio la costante introduzione di identificatori (attributi id) per ciascun nuovo elemento gerarchico. Ciò consente di poter fare precisi riferimenti nei livelli inferiori. Ad esempio, come già detto, questo permette di associare le part agli staff tramite l'attributo dfstaff_ref di part; o i voice in una measure ai voice_item della voice_list; o ancora i *notehead* che compongono gli accordi agli staff tramite l'attributo dfstaff_ref (quest'ultimo punto non è ancora stato introdotto e non è evidente dallo schema sopra riportato).

Tale schema si semplifica notevolmente nel caso in cui si abbia a che fare con parti strumentali monodiche: in tal caso, lo staff_list si compone di un unico staff, che a sua volta contiene un'unica part, che a sua volta contiene un'unica voice.

Vediamo qualche esempio pratico:



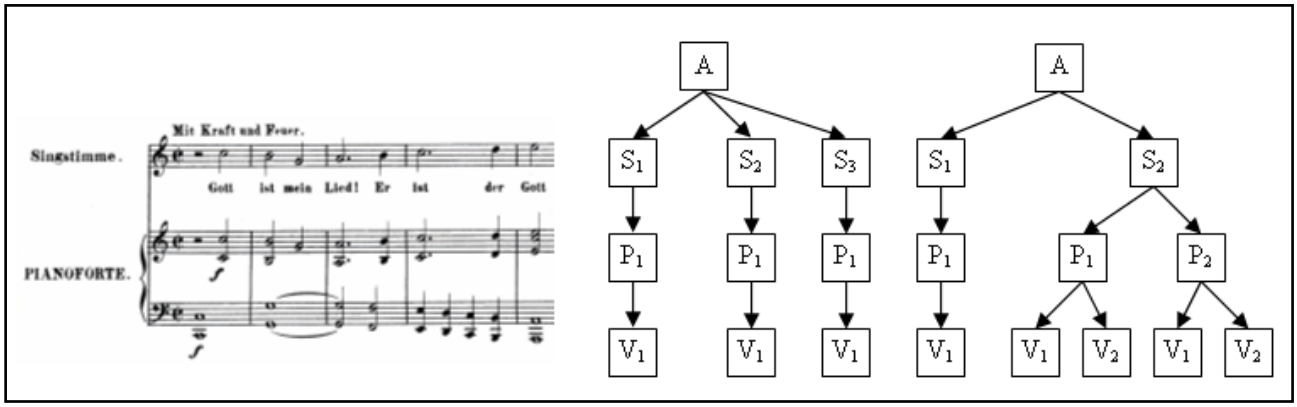

ogni accollatura contiene 4 pentagrammi, ciascuno dei quali contiene una parte monofonica e dunque una sola voce.

Nello schema ad albero sulla destra **A** sta per Accollatura, **S** per staff, **P** per Parte e **V** per voce.

Di seguito riportiamo anche lo scheletro del file MX risultante

```
<los>
     <staff_list>
          <staff id="staff_soprano"> … </staff>
          <staff id="staff_contralto"> … </staff>
          <staff id="staff_tenore"> … </staff>
          <staff id="staff_basso"> … </staff>
     </staff_list>
     <part id="soprano" dfstaff_ref="staff_soprano">
          <voice_list>
               <voice_item id="voce_soprano"/>
          </voice_list>
          <measure number="1"> … </measure>
          <measure number="2"> … </measure>
          …
     </part>
     <part id="contralto" dfstaff_ref="staff_contralto">
          <voice_list>
```

```
            <voice_item id="voce_contralto"/>
        </voice_list>
        <measure number="1"> … </measure>
        <measure number="2"> … </measure>
        …
    </part>
    <part id="tenore" dfstaff_ref="staff_tenore">
        <voice_list>
            <voice_item id="voce_tenore"/>
        </voice_list>
        <measure number="1"> … </measure>
        <measure number="2"> … </measure>
        …
    </part>
    <part id="basso" dfstaff_ref="staff_basso">
        <voice_list>
            <voice_item id="voce_basso"/>
        </voice_list>
        <measure number="1"> … </measure>
        <measure number="2"> … </measure>
        …
    </part>
</los>
```




L'accollatura contiene 2 pentagrammi (volendo considerare uno solo il pentagramma composto del pianoforte) o in alternativa 3 (il che porta a una gestione più semplice della

partitura in MX). Inoltre, possiamo considerare i raddoppi del pianoforte come voci a sé stanti o voci singole che procedono ad accordi.

Sono presentati due tipi di albero a seconda della scelta effettuata. La scelta è arbitraria in quanto non esiste un metodo "ottimale".

Mostriamo nello scheletro MX la seconda struttura

```
<los>
     <staff_list>
          <staff id="staff_voce">  … </staff>
          <staff id="staff_pianoforte"> … </staff>
     </staff_list>
     <part id="part_voce" dfstaff_ref="staff_voce">
          <voice_list>
               <voice_item id="soprano"/>
          </voice_list>
          <measure number="1"> … </measure>
          <measure number="2"> … </measure>
          …
     </part>
     <part id="part_pianoforte_dx"
dfstaff_ref="staff_pianoforte">
          <voice_list>
               <voice_item id="voce_dx_sup"/>
               <voice_item id="voce_dx_inf"/>
          </voice_list>
          <measure number="1"> … </measure>
          <measure number="2"> … </measure>
          …
     </part>
     <part id="part_pianoforte_sx"
dfstaff_ref="staff_pianoforte">
          <voice_list>
               <voice_item id="voce_sx_sup"/>
               <voice_item id="voce_sx_inf"/>
          </voice_list>
```

```
          <measure number="1"> … </measure>
          <measure number="2"> … </measure>
          …
     </part>
</los>
```

### *3.3.3.2 – elementi di Los*

Nel DTD Los è così definito

```
<!ELEMENT los (agogics*, text_field*, metronomic_indication*,
staff_list, part+, horizontal_symbols?, lyrics*)>
```

L'asterisco indica una voce non del tutto definita. Tuttavia l'elemento *agogics* è il primo che troviamo all'interno del layer los.
Esso rappresenta appunto l'agogica e conterrà fra gli attributi un nome, un'identificatore per lo spine più altre caratteristiche come ad esempio il tipo di font.
Stesso dicasi per *text-field* e *metronomic_indication* cui il significato è ovvio.
Un possibile esempio di agogics nel file mx potrebbe essere il seguente:

```
<los>
     <agogics event_ref="timesig_1"
          font_type="Times New Roman"
          font_size="12">Allegretto</agogics>
     …
</los>
```

### *3.3.3.3 – staff*

```
<!ELEMENT staff_list (brackets | staff)+>

<!ELEMENT brackets EMPTY>
<!ATTLIST brackets
```

```
     marker (start_of_staff_group | end_of_staff_group)
#REQUIRED
     group_number CDATA #REQUIRED
     type (square | rounded_square | brace | vertical_bar |
none) #REQUIRED>

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

Il DTD evidenzia che uno staff_list (accollatura) deve contenere almeno un pentagramma, come è logico attendersi.
Ciascun pentagramma contiene le informazioni su:

- clef (la chiave)
- key_signature (armatura di chiave)
- time_signature (indicazione della divisione temporale della battuta)
- barline (stanghetta di fine battuta)
- staff_hiding (elemento usato per l'attributo "ossia")

impostare il pentagramma come *ossia* genera un pentagramma "flottante" per i passaggi musicali alternativi, ad esempio per la realizzazione di abbellimenti o cadenze.

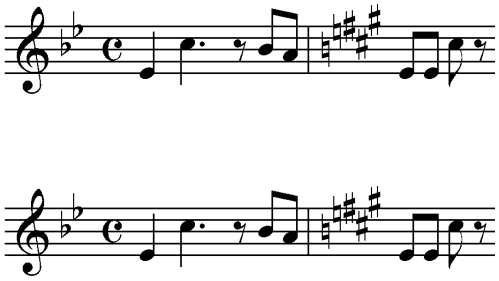
Senza soffermarci sulla definizione e la spiegazione di tutti i figli di staff, vediamo alcuni esempi per fissare le idee.

Canto 

```
<clef type="G" staff_step="2"
event_ref="clef_1"/>
```

Basso 

```
<clef type="F" staff_step="6"
event_ref="clef_1"/>
```



```
<key_signature
event_ref="keysig_1">
     <flat_num number="2"/>
</key_signature>
<key_signature
event_ref="keysig_2">
     <natural_num
number="2"/>
     <sharp_num number="3"/>
</key_signature>
```

| | |
|---|---|
|  | `<time_signature event_ref="timesig_1" num="2" den="4" vtu_amount="4000"/>` |

### *3.3.3.4 – measure (battua)*

*Measure* è elemento figlio di part, ed è allo stesso livello di *voice_list*.
Esso è il contenitore di tutte le informazione relative alla battuta. Quindi note e pause principalmente.
Essendo figlio di part, sappiamo già a quale strumento appartengono. Al suo interno poi, vengono specificate le eventuali voci che lo compongono.
Dobbiamo solo identificarlo con un numero, che identificherà anche l'eventuale concorrenza con altre *part* o altre *voice*.

All'interno di measure vi è il *voice_ref*, il nostro riferimento alla voce (pensiamo ad esempio alla mano destra e la mano sinistra del pianoforte).

Schematizzando

```
<los>
     <staff_list> … </staff_list>
     <part id="part_organo" dfstaff_ref="staff_organo">
          <voice_list>
               <voice_item id="mano_dx"/>
               <voice_item id="mano_sx"/>
          </voice_list>
          <measure number="1">
               <voice ref=”mano_dx”> … </voice>
               <voice ref=”mano_sx”> … </voice>
          </measure>
          <measure number="2">
          ...
</los>
```

### *3.3.3.5 – chord e rest*

Ci addentriamo sempre più nella struttura per apprestarci ad incontrare gli elementi nucleici di MX
Una volta identificata la singola battuta ne esaminiamo il contenuto che potrà essere formato da note e da pause (più tutta la grammatica musicale che ne consegue).

L’elemento padre è il *chord*. In esso sono definite le note e le pause per intervallo di tempo. Più esattamente il chord contiene prima di tutto il suo riferimento allo spine (event_ref) che mappa tutte le referenze musicali. Quindi la durata.
Sarà poi compito dell’elemento figlio *notehead* specificare il tipo di nota e tutte le eventuali alterazioni di tonalità e valore.
Se non vi sono note ma pause, chord conterrà solo il riferimento allo spine. In questo caso al posto di notehead avremo l’elemnto chiuso rest (la pausa appunto) con l’attributo di durata.

Vediamone un esempio e illustriamolo

```
...
```

```
<measure number="7">

    <voice ref="voce_Grand Piano_dx">
         <chord event_ref="evento_nota_Grand Piano_01_000012">
               <rest duration num="2" den="4"/>
         </chord>
         <chord event_ref="evento_nota_Grand Piano_01_000285"
duration num="2" den="4">
               <notehead>
                     ...
               </notehead>
          </chord>
…
```

Ci troviamo all'interno della battuta 7 di un dato strumento definito più in alto da part.
Più precisamente, la voce cantata dalla mano destra di un pianoforte.
Notiamo nel primo *chord* il riferimento allo spine, ma nessuna *duration*. Questo perché c'è una pausa definita dall'elemento chiuso rest che reca con sé l'attributo di durata. Nel nostro caso una pausa minima.
Il resto della battuta è composto da note (una singola o un accordo) di durata sempre minima.
Quali che siano e quali elementi tipografici contengano saranno specificati nel notehead che vedremo fra breve.
E' importante rimarcare che tutte le informazioni operative sono contenute nello spine.
E'grazie ad esso che riusciamo a ricostruire la partitura nella sua corretta funzionalità esecutiva e il suo ordinamento spazio-temporale.
Inoltre non ci interessa a questo livello specificare armatura di chiave, di tempo o altro in quanto basta risalire a gli elementi padri per sapere esattamente dove ci troviamo e quali informazioni porta la battuta corrente in esame.
Risulta chiaro come questo struttura sia utile potendo usufruire di software adeguati, per accedere a qualsiasi informazione musicale di un brano in maniera dettagliata, rapida e profonda.

#### *3.3.3.6 – il notehead*

Se diamo un'occhiata alla definizione di *chord* nel DTD ci accorgiamo come esso possa e debba essere ricco.

```
<!ELEMENT chord (duration, aug_dot?, notehead+, articulation?,
arpeggio?)>
<!ATTLIST chord
     id ID #IMPLIED
     %spine_ref;
     stem_direction (up | down | none) #IMPLIED
     stem_length CDATA #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>
```

Pensiamo ad esempio all'elemento *articulation*. Esso porta con sé numerose descrizioni così come abbiamo visto nel secondo capitolo (legato, staccato, tenuto ecc.). Stesso dicasi per *l'arpeggio.*

Notiamo anche la ATTLIST composta da informazioni prevalentemente tipografiche su gambo e testa della nota. La completezza dell'MX schematizza tutte le informazioni della teoria musicale.

Noi ci concentreremo sul notehead, rimandando alla consultazione del DTD e dell'ampia letteratura presente sull'MX per i dettagli.

Cominciamo dando un'occhiata alla definizione di notehead

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

<!ELEMENT pitch EMPTY>
```

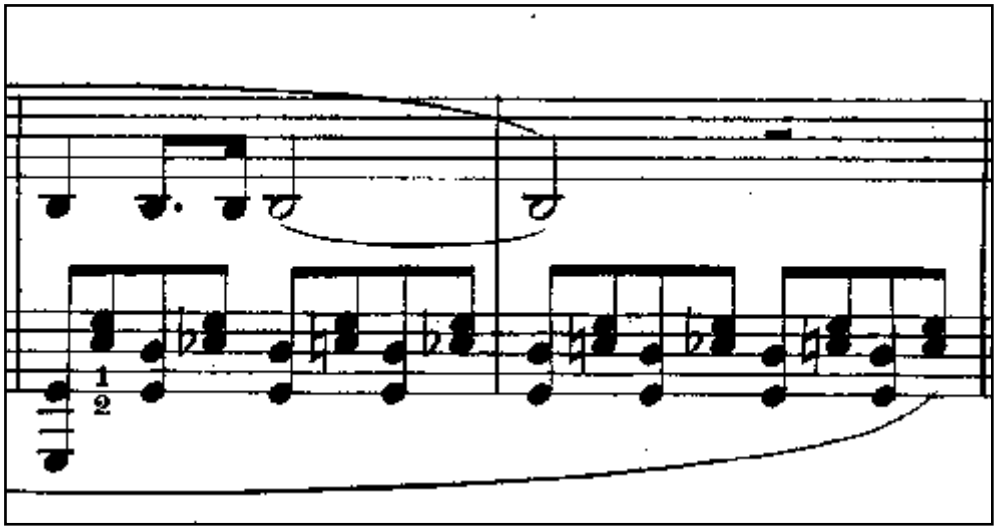
```
<!ATTLIST pitch
     step (A | B | C | D | E | F | G | none) #REQUIRED
     octave CDATA #REQUIRED
     actual_accidental %accidental; #IMPLIED>

<!ELEMENT printed_accidentals (sharp | flat | natural)+>
<!ATTLIST printed_accidentals
     size (normal | small) "normal">

<!ELEMENT sharp EMPTY>
<!ATTLIST sharp
     parenthesis (yes | no) "no">

<!ELEMENT flat EMPTY>
<!ATTLIST flat
     parenthesis (yes | no) "no">

<!ELEMENT natural EMPTY>
<!ATTLIST natural
     parenthesis (yes | no) "no">

<!ELEMENT augmentation_dots EMPTY>
<!ATTLIST augmentation_dots
     number CDATA "1">

<!ELEMENT tie EMPTY>
```

Esso contiene l'altezza della nota (pitch), le alterazioni (in accidental) e la legatura di valore (tie).

La dicitura delle note è data secondo la codifica anglosassone (A = LA, B = SI ecc.).
La combinazione pitch + ottava ci da l'altezza della nota.
Le tre alterazioni sono rispettivamente il diesis (sharp), il bemolle, (flat) ed il bequadro (natural).

La legatura è elemento chiuso ed è associato al notehead da cui la legatura di valore ha inizio. Il notehead di arrivo non viene segnato, a meno che non vi sia un'ulteriore legatura.

Vediamo come viene codificato questo estratto dal secondo preludio di Chopin.
Per semplicità ci occuperemo della sola mano destra.



```
<measure number="6">
     <voice ref="voce_ Grand Piano_01">
         <chord event_ref="evento_nota_ Grand Piano_01_000008"
duration="480">
            <notehead>
               <pitch step="B" octave="4"/>
            </notehead>
         </chord>
         <chord event_ref="evento_nota_ Grand Piano_01_000009"
duration="360">
            <notehead>
               <pitch step="B" octave="4"/>
            </notehead>
         </chord>
         <chord event_ref="evento_nota_ Grand Piano_01_000010"
duration="120">
            <notehead>
               <pitch step="B" octave="4"/>
```

```
            </notehead>
         </chord>
         <chord event_ref="evento_nota_ Grand Piano_01_000011"
duration="960">
            <notehead>
               <pitch step="B" octave="4"/>
               <tie/>
            </notehead>
         </chord>
     </voice>
<measure number="7">
     <voice ref="voce_Acoustic Grand Piano     _01">
        <chord event_ref="evento_nota_Grand Piano_01_000012">
           <rest duration="960"/>
        </chord>
        <chord event_ref="evento_nota_Grand Piano _01_000285"
duration="960">
           <notehead>
              <pitch step="B" octave="4"/>
           </notehead>
        </chord>
     </voice>
```

Si noti come nella measure “7” segnamo prima la pausa e poi la nota. Ciò non è rilevante in quanto è attraverso l’event_ref che ricostruiamo l’ordine temporale degli eventi.
Ciò ci dimostra l’importanza dello spine.

## *3.4 – Scelte MX*

Il formato MX è stato illustrato nella sua completa funzionalità.
Come visto, esso ci permette di effettuare delle scelte su come strutturarlo, e di conseguenza su come strutturare la codifica della musica (si vedano a tal proposito gli esempi del paragrafo 3.3.3.1).
Per scrivere il programma MIDI2MX di cui ci occuperemo nel quinto capitolo, sono state effettuate delle scelte che privilegiano delle strutture rispetto ad altre. Così come alcuni elementi ed alcuni layer non sono stati implementati o presentano della caratteristiche leggermente diverse.
Ciò è dipeso, oltre che da difficoltà oggettive, dalla particolarità delle informazioni veicolate dal formato MIDI, che esamineremo nel dettaglio nel prossimo capitolo.

In particolare nel file MX generato automaticamente dall'applicazione si è ritenuto opportuno compiere la seguenti scelte:

- Ogni *part* rappresenta un singolo strumento
- Nella staff list saranno presenti tanti *clef, time segnature e key segnature* quante sono le voci che compongono lo strumento definito in part
- Le voci dello strumento sono definite nella *voice list*
- I raddoppi non sono visti come voci separate, ma come *chord* multipli all'interno della singola voce
- Le durate sia delle note che della pause vengono espresse con cifre intere (VTU) e non con frazioni

In ogni caso, su queste differenze torneremo più dettagliatamente quando ci occuperemo del programma MIDI2MX.

# Capitolo 4

# IL MIDI

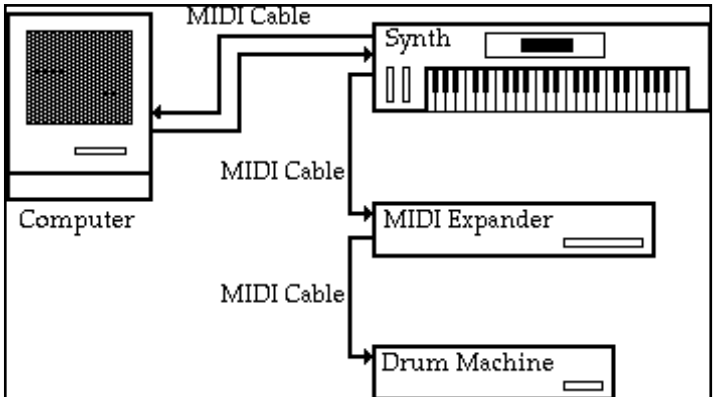
La formalizzazione di una partitura musicale attraverso l'MX sarà il nostro risultato finale.
E' il momento quindi di analizzare la sorgente sonora da cui far partire la nostra conversione. O meglio ancora, la nostra interpretazione.

Esistono molti formati musicali per la musica digitalizzata. Dal popolare *MP3*, ai vetusti e gloriosi *MOD* e *Xm* originari della macchina *Commodore Amiga*. Arrivando ai moderni formati proprietari di software di editing e composing professionali quali *Reason*, *Live*, *Cubase* ed altri ancora.
Ed ovviamente ci sono i basilari formati di campionamento di forma d'onda. Come il *WAV*.
E lo standard MIDI, di cui ci occuperemo diffusamente in questo capitolo.
La particolarità di tutti questi tipi di file, è che ognuno di essi rappresenta la musica secondo determinati criteri. Diversi fra loro e con diverse caratteristiche. Questo è dovuto dal fatto che ognuno di questi formati doveva rispondere a determinate esigenze. Siano esse tecnologiche (hardware e software) che operative e funzionali.
Il primo passo è stato certamente la digitalizzazione della forma d'onda.
Attraverso la campionatura e la quantizzazione, il segnale analogico in ingresso viene discretizzato secondo i principi fisici - matematici di Fourier e reso disponibile quindi in forma numerica.
Tutti i formati che rappresentano la forma d'onda si basano sul *PCM* (Pulse Code Modulation).

Al contrario i files MIDI, veicolano istruzioni da dare ad uno specifico hardware, il quale a sua volta conterrà i sintetizzatori necessari a riprodurre i suoni richiesti. Ne risulta in primis un'occupazione di memoria molto scarsa. Ma soprattutto, queste istruzioni di cui ora parleremo, ci permettono di impostare un modello interpretativo molto vicino alla teoria musicale tradizionale.

## *4.1 – Cenni storici*

IL MIDI (Musical Instrument Digital Interface), fu definito per la prima volta da Dave Smith e Chet Wood nel 1980, ed in origine era chiamato USI (Universal Synthesizer Interface).
Lo scopo era quello di ovviare ai problemi di comunicazione tra i sintetizzatori.
Alla fine del 1981, l'interfaccia USI fu sottoposta alla Audio Engeneering Society. Dopo due anni di integrazioni e modifiche (con la partecipazione di tutti i maggiori produttori di strumenti musicali elettronici), nel 1983 fu definito uno standard dal nome MIDI (Musical Instrument Digital Interface).
Il MIDI fu subito accettato dai più grandi produttori per il suo basso costo di implementazione e per la ormai sentita esigenza di uno standard in questo settore. In breve tempo ci fu un grande positivo riscontro da parte degli utenti che finalmente potevano far comunicare tra loro gli strumenti musicali elettronici di ogni marca.
Possiamo definire lo standard MIDI come un insieme di specifiche hardware e software che rende possibile lo scambio di informazioni (note, modifiche di configurazione, controllo dell'espressione etc.) tra strumenti musicali elettronici o altri dispositivi elettronici come computers, sequencer, centraline di controllo luci, mixer etc.

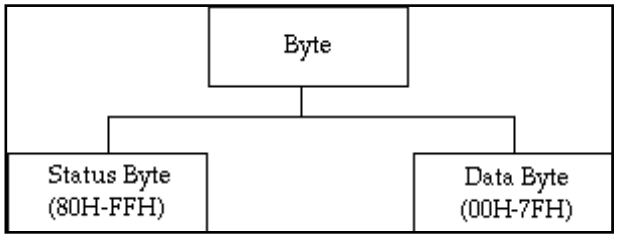


## *4.2 - Specifiche MIDI*

Per una prima panoramica delle specifiche MIDI ci riferiamo a [17] e [18]

•L'interfaccia hardware MIDI opera a 31.25 (±1%) Kbaud, è asincrona, ha uno start bit, 8 data bit e uno stop bit. Ciò significa che per trasmettere serialmente un byte formato da 8 + 2 bit sono necessari circa 320 microsecondi.
Il circuito è percorso da una corrente di 5mA. Un output sarà collegato a uno e un solo input. La circuiteria del trasmittente e del ricevente sarà separata internamente da un opto-isolatore. Nelle specifiche viene fornito un dettagliato schema del circuito.
I connettori montati sui dispositivi sono pentapolari (180 gradi) femmina. I connettori devono essere chiamati "MIDI IN" e "MIDI OUT". I pin 1 e 3 (i più esterni) non sono usati e non devono essere connessi.
I cavi devono avere una lunghezza massima di 15 metri e le due estremità devono essere terminate con un connettore pentapolare maschio.
Sui dispositivi può essere installata una porta di output aggiuntiva chiamata "MIDI Thru" che fornisce una copia esatta dei dati in arrivo sulla porta di input. Per catene di collegamento lunghe (più di tre strumenti), si devono usare optoisolatori ad alta velocità per limitare i ritardi di trasmissione dei dati.

Più importanti per i nostri fini sono le specifiche software.
Il sistema di comunicazione dei MIDI si basa messaggi formati da uno *Status byte* (che identifica il messaggio) seguito da uno o due *Data byte* (che descrivono il messaggio).



Il primo bit a sinistra (il bit più significativo o anche MSB, Most Significant Bit) identifica il tipo di byte. 1 per lo Status Bye e 0 per il Data byte. Quindi, abbiamo a disposizione 7 bit per rappresentare i nostri dati. Ci riferiremo ad essi attraverso l'acronimo LSB, Less Significant Bit.

I messaggi MIDI si dividono sostanzialmente in due categorie. *Channel Message* e *System Message.*

## *4.3 – Messaggi MIDI*

- Channel Message

Lo Status Byte dei Channel Message contiene nei primi 4 bit il codice di identificazione del messaggio e nei secondi 4 bit il numero di canale MIDI (0..15) a cui il Channel Message appartiene. Per chiarire questo concetto, ammettiamo sia possibile impostare un device MIDI in modo che presti attenzione solo ai messaggi che appartengono ad un particolare canale o insieme di canali. Allora il ricevente potrà usare questo codice come un indirizzo che viene scritto sui dati in partenza in modo tale che ogni device che è in condizioni di riceverlo fisicamente possa decidere o meno di considerarlo. In questo modo, all'interno del sistema MIDI si possono identificare degli insiemi di dispositivi che reagiscono solo a certi tipi di messaggi, ignorando automaticamente gli altri.

Ci sono due tipi di Channel Message: *Voice* e *Mode*.

I Channel Voice Message controllano in generale le voci dello strumento (cioé che cosa deve suonare lo strumento).

I Channel Mode Message controllano come lo strumento deve gestire i Channel Voice Message (cioé il comportamento dello strumento al momento della ricezione di un Voice Message). I Channel Mode Message vengono trasmessi sul "Basic Channel" dello strumento che vogliono controllare, cioè sul canale che viene "ascoltato" dallo strumento.

- System Message

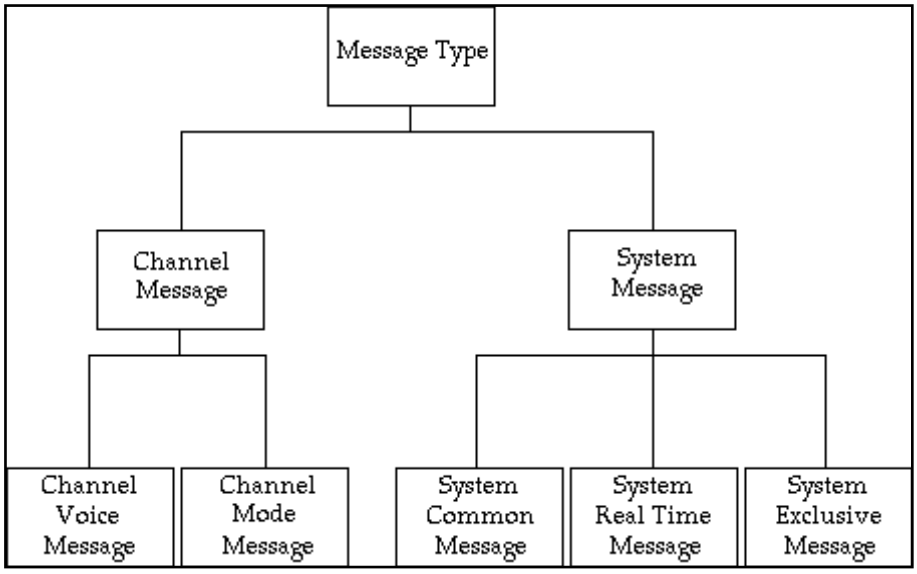
I System Message non hanno un numero di canale specificato nello Status Byte: lo Status Byte descrive solamente il codice di identificazione del messaggio.

Esistono tre tipi di System Message: *Common*, *Real Time*, *Exclusive*.

I System Common Message sono indirizzati a tutti i dispositivi presenti nel sistema MIDI, indipendentemente dal loro "Basic Channel".

I System Real Time Message sono usati per la sicronizzazione dell'intero sistema MIDI e sono indirizzati a tutti i dispositivi che sono predisposti ad accettare un codice di sincronizzazione (sono in "time slave"). I Real Time sono costituiti da un solo Status Byte che può e deve essere posizionato in mezzo a qualunque messaggio MIDI. Quindi si può verificare il caso in cui un Channel Message venga "rotto" da un Real Time sia in trasmissione che in ricezione.

I System Exclusive Message sono usati per comunicare ai dispositivi del sistema messaggi particolari, che dipendono dal dispositivo stesso e che non sono significativi per altri. I System Exclusive sono costituiti dallo Status Byte di header e da un qualsiasi numero di Data byte; il System Exclusive può finire o con uno specifico messaggio di End of Exclusive (*EOX*) o con un qualsiasi altro Status Byte (tranne lo Status Byte dei Real Time Message). Un EOX dovrebbe comunque essere mandato. Questi messaggi includono un ID del costruttore, in modo da consentire al singolo dispositivo di saltare o esaminare completamente il messaggio.




Nei prossimi paragrafi esamineremo attentamente i messaggi che più di tutti utilizzeremo nella nostra applicazione

### *4.3.1 – channel message*

**Il messaggio NOTE ON (Channel Voice Message).**

Questo messaggio viene inviato da una tastiera, batteria elettronica etc. quando una nota viene attivata (tasto per una tastiera, pad per una batteria elettronica etc.).

Il messaggio MIDI di *NOTE ON* è composto da uno Status Byte e due Data Byte

| | |
|---|---|
| Status & Channel Number (0-15)[1]<br>1001 CCCC | I primi 4 bit identificano il messaggio<br>I secondi 4 bit identificano il canale |
| Note Number<br>0NNN NNNN | Il numero della nota (0..127) |
| Velocity<br>0VVV VVVV | Identifica quanta "forza" viene impressa alla nota. Un NOTE ON con velocità pari a 0 è da considerarsi un NOTE OFF |

**Il messaggio NOTE OFF (Channel Voice Message).**

Questo messaggio viene inviato quando una nota viene disattivata e, come il messaggio di NOTE ON è composto da uno Status Byte e due Data Byte

| | |
|---|---|
| Status & Channel Number (0-15)<br>1000 CCCC | |
| Note Number<br>0NNN NNNN | Vedi NOTE ON |
| Velocity<br>0VVV VVVV | In questo caso identifica la velocità del rilascio della nota. |

**Il messaggio Program Change (Channel Voice Message).**

[1] E' la definizione di Status byte e non verrà più segnalata

I dispositivi MIDI dedicati alla produzione del suono come per esempio i sintetizzatori MIDI, sono multitimbrici. Questo messaggio ci consente di trasmettere un segnale per cambiare il timbro di un sintetizzatore MIDI.

| | |
|---|---|
| Status & Channel Number (0-15)<br>1100 CCCC | |
| Program Number<br>0PPP PPPP | Il numero identificativo dello strumento. |

L'assegnazione dell'identificativo dello strumento non è univoca e dipende dal supporto hardware. Tuttavia, è possibile usare la tabella General MIDI, la quale è così composta

**General MIDI Instrument Families**

| PC# | Family | PC# | Family |
|---|---|---|---|
| 1-8 | Piano | 65-72 | Reed |
| 9-16 | Chromatic Percussion | 73-80 | Pipe |
| 17-24 | Organ | 81-88 | Synth Lead |
| 25-32 | Guitar | 89-96 | Synth Pad |
| 33-40 | Bass | 97-104 | Synth Effects |
| 41-48 | Strings | 105-112 | Ethnic |
| 49-56 | Ensemble | 113-120 | Percussive |
| 57-64 | Brass | 121-128 | Sound Effects |

**Il messaggio Poliphonic Key Pressure (Poliphonic After Touch) (Channel Voice Message).**

Questo messaggio viene trasmesso quando il dispositivo MIDI è in grado di rilevare e trasmettere cambi di pressione su ogni singolo tasto dopo l'attivazione della nota.

| | |
|---|---|
| Status & Channel Number (0-15)<br>1010 CCCC | |
| Note Number<br>0NNN NNNN | |
| Pressure Value | Il valore della pressione |

| | |
|---|---|
| 0VVV VVVV | |

Nel caso in cui ci siano tre messaggi di NOTE ON "aperti", con il Poliphonic After Touch si riescono a trasmettere i cambi di pressione relativi ad ogni tasto abbassato sulla tastiera.

**Il messaggio Channel Pressure (After Touch) (Channel Voice Message).**

Questo messaggio viene trasmesso quando il dispositivo MIDI è in grado di rilevare e trasmettere cambi di pressione su ogni singolo tasto dopo l'attivazione della nota. A differenza del messaggio di Poliphonic Key Pressure, qui non troviamo il note value proprio perché tutte le note suonate, indipendentemente dalla loro specifica pressione, vengono coinvolte in modo uniforme alle modifiche timbriche che possono venire associate a questo messaggio (vibrato, phase, loudness etc.). Le modifiche timbriche che possono venire associate ai messaggi di after touch in genere, dipendono strettamente dal dispositivo MIDI che si sta usando. Il MIDI si limita a fornire la trasmissione dell'evento di pressione sul tasto ma non il suo significato!.
Il messaggio Channel Pressure è così composto:

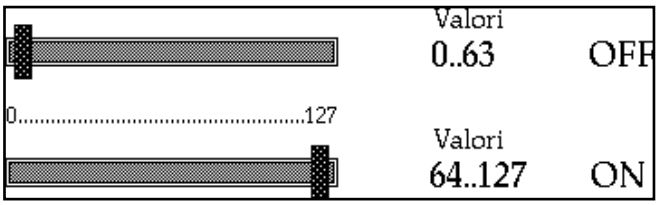
| | |
|---|---|
| Status & Channel Number (0-15)<br>1101 CCCC | |
| Pressure Value<br>0VVV VVVV | Il valore della pressione, riferito a tutte le note aperte |

**I messaggi Control Change (Channel Voice Message).**

Questi messaggi consentono di gestire in real time i parametri di controllo dei device MIDI.

| | |
|---|---|
| Status & Channel Number (0-15)<br>1011 CCCC | |
| Controller ID Number<br>0NNN NNNN | Specifica il tipo di controller MIDI (volumi, pedali, switch ecc.) |
| Controller Value<br>0VVV VVVV | Valore specifico per ogni controller |

Dobbiamo subito far notare che i controller con un ID number compreso tra 120 e 127 servono per identificare i Channel Mode Message.
Altri si dividono principalmente in due categorie: continuous controller (da 0 a 63) e switches (da 64 a 95). I rimanenti non sono definiti.
E' chiaro che la generazione di questi messaggi è strettamente legata allo strumento MIDI che si sta usando.
Pensiamo ad esempio, alla rotellina per la regolazione del panning




I Channel Mode Message invece, specificano il comportamento dei device rispetto al flusso di dati MIDI. Il 121 ad esempio resetta tutti i controller. Il 123, sopprime tutte le note in esecuzione e così via.
Per una completa panoramica dei messaggi MIDI si consulti l'appendice A.

### *4.3.2. – system message*

Molti dispositivi MIDI (tastiere, batterie elettroniche etc.) dispongono di un *sequencer* interno in grado di memorizzare delle sequenze di dati MIDI. Possiamo quindi pensare di avere una batteria elettronica con memorizzata la sequenza di batteria di una *MIDI song* (un brano in formato MIDI) e una tastiera con memorizzata la parte armonica e melodica della stessa song. La MIDI song è quindi residente su due distinti device fisici che dovranno in qualche modo cooperare per l'esecuzione della song stessa. E' quindi necessario che questi due device possano sincronizzarsi. Si dovrà definire un master e uno slave, ad esempio la batteria elettronica master e la tastiera slave e premendo il tasto di start song sulla batteria anche la tastiera, se opportunamente collegata via MIDI, comincerà ad eseguire la sua parte di song.
Vedremo come il protocollo MIDI consente di realizzare questo tipo di cooperazione tra due o più MIDI device.

Relativamente ai nostri fini, queste informazioni ci permetteranno di posizionarci in un punto preciso della song e quindi estrapolare alcune importanti informazioni. Una su tutte, le durate delle note.
L'insieme delle funzioni necessario alla sincronizzazione comprenderà messaggi di tipo System Common e di tipo System Real Time

**Il messaggio Song Position Pointer (System Common Message)**

| Status |
|---|
| 1111 0010 |
| Least Significant |
| 0LLL LLLL |
| Most Significant |
| 0MMM MMMM |

Questo messaggio viene inviato per comunicare la posizione corrente della song. E' un singolo numero intero (max 14 bit di risoluzione, 7 LSB e 7 MSB) che rappresenta il numero di MIDI beats (1 beat = 6 MIDI clocks; 24 MIDI clocks rappresentano un nota da un quarto) dall'inizio della song. Viene generalmente trasmesso quando la song parte e quando si ferma, permettendo così al device in slave di posizionarsi al punto della song specificato.

**Il messaggio Song Select (System Common Message)**

| Status |
|---|
| 1111 0011 |
| Song Number |
| 0SSS SSSS |

I MIDI device possono disporre allo stesso momento di altre song, oltre a quella corrente. In questo modo si può richiedere ad un device di cambiare la song corrente con quella specificata nel messaggio di Song Select.

### Il messaggio Timing Clock (System Real Time Message)

| Status |
| --- |
| 1111 1000 |

sincronizza l'intero sistema MIDI. Viene generato dal device master 24 volte per ogni nota da un quarto e controlla l'avanzamento dei device slave. Può essere trasmesso in ogni momento, anche spezzando qualunque altro messaggio.

### I messaggi System Exclusive (System Message).

Questo messaggio consente di comunicare ai MIDI device dei messaggi "custom", dedicati esclusivamente ad un particolare device. Questo serve di solito per elaborare parametri che dipendono fortemente dal device in uso: per esempio i timbri. Ogni sintetizzatore definisce un suono in base alla attribuzione di valori specifici a tutte variabili che contribuiscono a creare un timbro. Questi valori sono, in un certo senso, le coordinate di un suono, che sono ovviamente fortemente variabili rispetto al tipo di sintesi usate. Possiamo anche citare i campionatori, il cui assetto di uso viene definito da un insieme di campioni e di parametri di gestione.
I messaggi di System Exclusive sono composti da uno Status Byte, da un codice di identificazione del costruttore dello strumento, da un numero qualsiasi di Data Byte e dallo Status Byte "EOX" (End Of System Exclusive). I messaggi Real Time possono anche essere trasmessi in mezzo ad un messaggio System Exclusive. Qualunque Status Byte diverso dai Real Time e da "EOX" termina il messaggio System Exclusive.

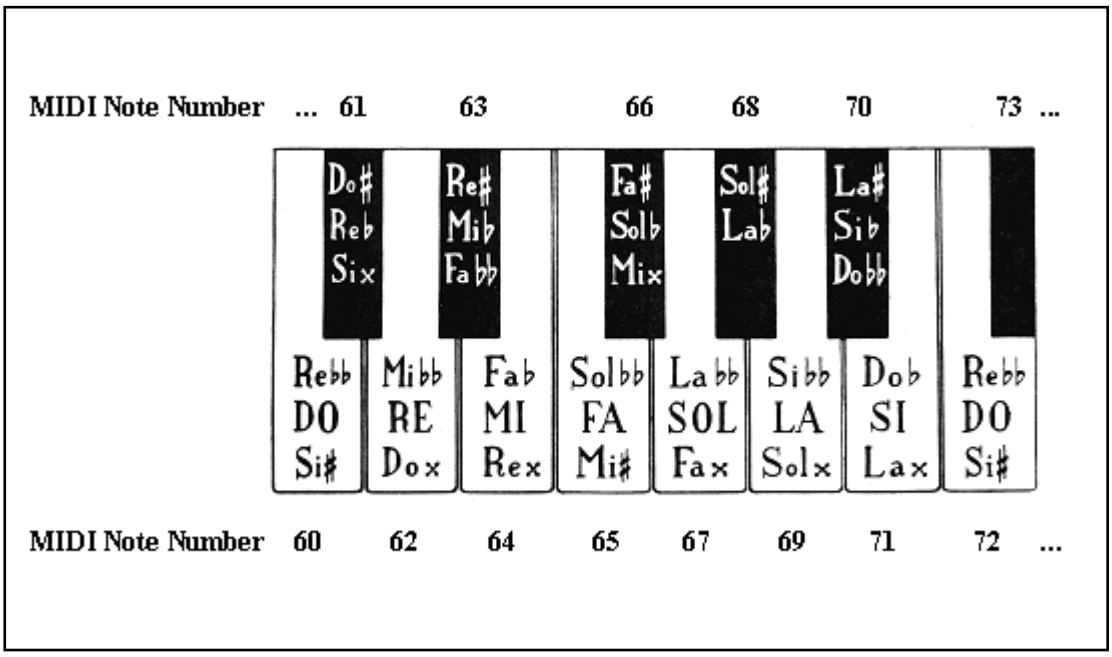
| Status |
| --- |
| 1111 0000 |
| Manifacturer's ID Byte |
| 0DDD DDDD |
| (Any Number of Data Bytes) |
| EOX |
| 1111 0111 |

## *4.3 – Notational vs Performance*

Prima di esaminare nel dettaglio le specifiche dei file, analizziamo le caratteristiche musicali del MIDI.
Nel nostro progetto, la scelta di tale formato non è stata casuale.
In primis, si tratta di un formato standard affermato e con un'ampia letteratura facilmente disponibile. Inoltre, nonostante i limiti e i problemi che possiamo trovarvi, è senza dubbio uno dei formati più ricchi dal punto di vista dell'informazione musicale. Rispetto ad esempio ad un qualsiasi formato di musica campionata.
In effetti i file MIDI non veicolano forma d'onda, ma solo le istruzioni necessarie affinché un synth MIDI possa eseguire una song.
Possiamo dire che tale sequenza di istruzioni è quanto di più vicino ci possa essere ad una partitura. Almeno dal punto di vista "funzionale". In realtà, come ora vedremo, vi è una differenza sostanziale potremmo dire all'antitesi, tra la grafia musicale e le istruzioni MIDI. Quest'ultime, danno all'elaboratore un'indicazione esatta e discreta di cosa e come suonare. Nella partitura, è l'uomo a decidere come e cosa suonare le note che legge sullo spartito. Questa differenza, è alla base della ricerca dei modelli di riconoscimento automatico, nonché di questo lavoro di tesi.
Esaminiamo nel dettaglio le principali differenze fra MIDI e teoria musicale.

- Le note

Partiamo da uno degli aspetti primari, le altezze delle note. Per indicarle, il MIDI ricorre ad una tastiera virtuale in cui ad ogni tasto è associato un numero. I 7 bit del messaggio NOTE ON. Quindi possiamo avere 128 altezze. In musica, la situazione è ben diversa.
La figura illustra molto chiaramente questa differenza




Quindi, trascrivere una nota MIDI in partitura presenta non poche ambiguità.

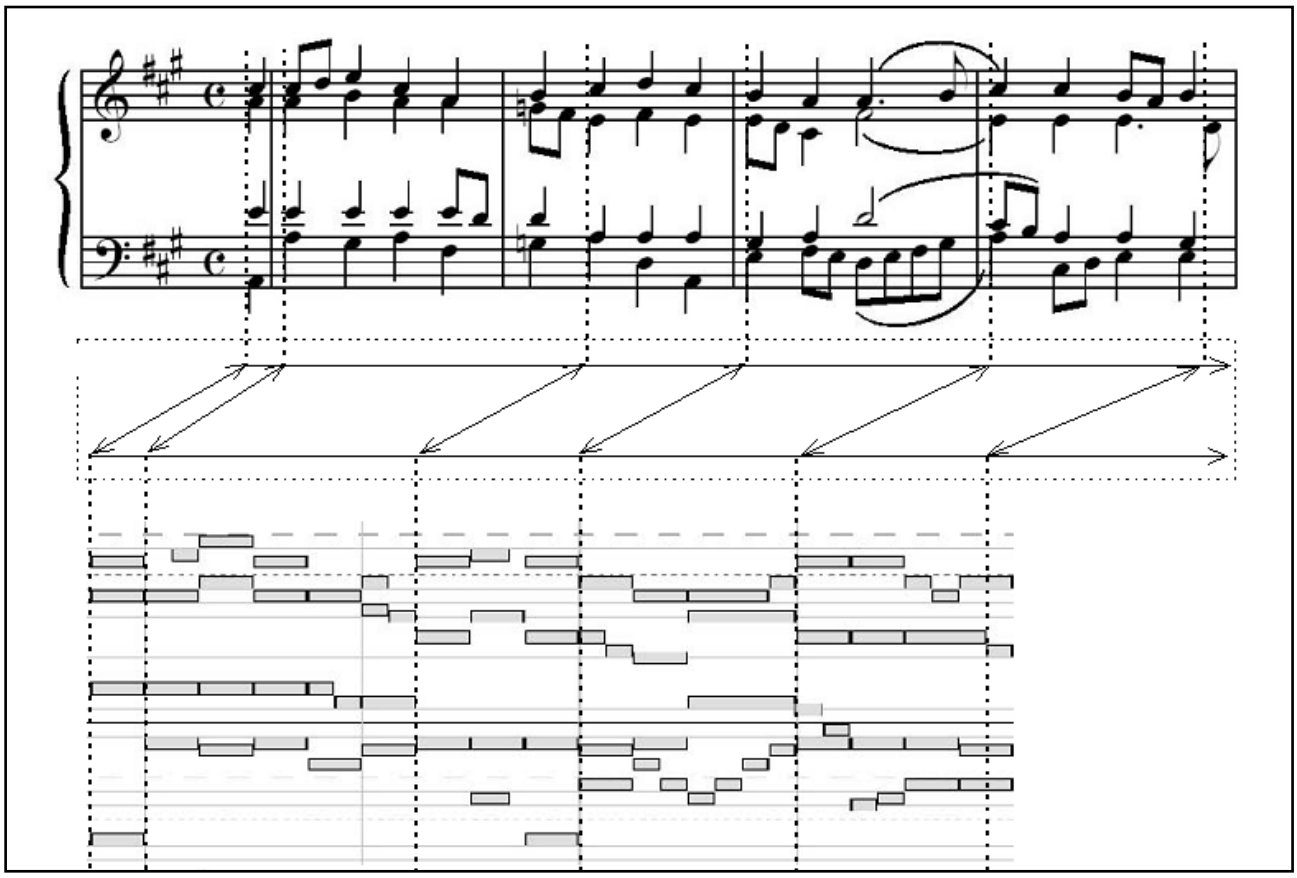
- Le pause e le note mute

Il MIDI non si preoccupa di segnare le pause. Abbiamo i NOTE OFF che spengono una note precedentemente attivata, ma non è sufficiente. Immaginiamo di avere ad esempio un pezzo musicale con due strumenti. Quando uno di questi non suona o semplicemente se attacca a suonare dopo il primo, nel MIDI non ci si preoccupa di segnarlo. Avremo il primo messaggio solo nel momento in cui questi sarà effettivamente attivo. Invece, in partitura, due strumenti anno i rispettivi pentagrammi uniti dall'accollatura e il numero delle battute ovviamente coincide. Semplicemente avremo delle pause per tutta la durate del "silenzio". Inoltre, un suono tenuto per più battute viene segnato in partitura con legature di valore. Nel MIDI avremmo semplicemente un NOTE ON lungo quanto la durata effettiva del suono.

- Le durate e le battute

Il MIDI, ovviamente, descrive in unità di tempo fisico le durate e la successione degli eventi nota. Questo aspetto è uno dei più critici, se confrontato con la divisione temporale musicale tradizionale.

Nel MIDI abbiamo prima di tutto, un tempo assoluto che scandisce la durata di un intero brano. Inoltre, ogni messaggio porta con sé l'intervallo di tempo che intercolle con l'evento immediatamente precedente. Questo intervallo di tempo è chiamato *delta time*.



.

Esiste tuttavia, anche un metodo di divisione. Ogni file MIDI racchiude la divisione in tick di una nota da un quarto. Questo numero è arbitrario e varia da file a file, tuttavia…

…….

In ogni caso, è da queste informazioni che in qualche modo ricaviamo le durate.

- Segnature in chiave

Consideriamo la chiave, l'indicazione del tempo e delle alterazioni.

Nel MIDI è possibile inserire queste informazioni, il problema è che a volte, laddove siano presenti, sono del tutto arbitrarie o comunque inesatte. Riguardo la chiave, possiamo contare sulla conoscenza dello strumento. Ad esempio un pianoforte ha tipicamente due voci (due canali MIDI) dove una è in chiave di violino e l'altro di basso.

Inoltre basandoci sull'altezza delle note, quando queste ultime superano certi limiti possiamo cambiare chiave.
Le segnatura di chiave e di tempo, possono essere ricavate dalla scansione delle note, considerando accenti, durate globali e successione di note particolari. In [x], c'è un interessante algoritmo che trova la tonalità di un brano partendo dall'analisi delle note che si susseguono in un brano.
Una volta individuata la tonalità, l'aggiunta delle alterazioni diventa automatica.
Nel nostro caso tuttavia, ci limiteremo a riscrivere un brano in una delle 15 scale a seconda della scelta dell'utente.
Per il tempo invece, poiché useremo VTU per le durate delle note, useremo sempre la divisione 4/4.

- La dinamica.

Attraverso il key velocity e il messaggio After Touch possiamo interpretare la pressione data all'attacco della nota e quindi all'indicazione di dinamica. Starà a noi preoccuparci di creare dei range di valori accettabili e soprattutto posizionarli solo quando serve e non per ogni evento come fa il MIDI.

- La spazializzazione

Andremo più nel dettaglio quando ci occuperemo degli Standard MIDI Files, ma già ora possiamo dire che il MIDI elenca gli eventi in modo sequenziale ed elencando un canale per volta.
In musica invece, lo scorrimento orizzontale è globale, per tutti gli strumenti e per tutte le voci ed è chiaramente definita la contemporaneità degli eventi.
Anche questo è un aspetto critico, ma di più facile risoluzione grazie alla precisione ed alla risoluzione dei tempi fisici.

- Abbellimenti

E' la sezione più ricca di segni grafici in musica e dove ci si può maggiormente sbizzarrire a livello interpretativo e di riconoscimento automatico. Staccati, terzine, acciaccature possono essere riconosciute prevalentemente attraverso gli intervalli di tempo fra gli eventi.

Riscontriamo tuttavia, anche uno dei limiti maggiori del MIDI. La risoluzione granulare del tempo. Eventi troppo vicini, possono non rientrare nel limite inferiore ed essere considerati come contemporanei. A livello pratico invece, capita che molte note suonate nello stesso tempo generino un alto traffico di dati con il risultato di avere un arpeggio.

- Informazioni generali sul brano

E' possibile avere meta eventi o comunque inserire stringhe di testo all'interno dei messaggi MIDI.
Quali, titolo, autore ecc. Tuttavia non sempre sono presenti, a volte sono incomplete ed in ogni caso non affidabili. Lasceremo all'utente il controllo di queste informazioni.

## *4.4 – Standard MIDI Files*

Prima della definizione dello *SMF* (Standard MIDI File), proposto per la prima volta nel 1986 dalla americana Opcode Systems, e adottato nel luglio 1988, i produttori di software musicale, per la memorizzazione di eventi MIDI, adottavano un loro particolare formato, con il risultato che, pur essendo i dati memorizzati, della stessa natura e su stessi supporti, i brani MIDI registrati dovevano essere letti dallo stesso tipo di software con cui erano stati creati.

L'implementazione e l'adozione dello SMF ha fatto in modo che, pur utilizzando software musicale (sequencer) diverso, la memorizzazione dei brani in formato MIDI avvenisse con la stessa procedura, consentendo così lo scambio e l'utilizzazione tra computer e sequencer diversi.

### *4.4.1 - struttura e formato*

- I chunk

Gli SMF sono composti da due blocchi principali definiti rispettivamente *Header-chunk* e *Track-chunk*. I primi 4 byte di ogni blocco, servono ad identificare il blocco stesso. L'Header-chunk (blocco d'intestazione) è il primo elemento inserito nel MIDI file, in quanto contiene le informazioni del formato, del numero di tracce e della temporizzazione. Inizia con quattro caratteri ASCII:MThd, che appunto identificano il blocco.

| **Dati esadecimali** | **Commento** |
|---|---|
| 4D 54 68 64 | ASCII=MThd |
| 00 00 00 06 | Lunghezza del blocco, in questo esempio 6 byte |
| 00 00 | Formato del MIDIfile (0,1,2) |
| 00 01 | Numero delle tracce (da 1 a 16) |
| 00 60 | Tick per quarto di nota, oppure<br>Tipo di sinc. SMPTE (24, 25, 29.97, 30 frame/s) |

Il quinto ed ultimo campo, *division*, specifica il significato dei delta-time, cioé dei valori temporali che separano tra loro gli eventi MIDI (come vedremo meglio descrivendo i track

chunk). Questo campo è sempre di 16 bit (integer/short) e può essere espresso in due formati: metrical time e time-code-based time.

Se il bit 15, il bit più alto, è uguale a zero, i bit dal 14 allo 0 rappresentano il numero di tick in cui viene divisa una nota da un quarto. Se division vale 96, per rappresentare un intervallo di un ottavo fra due eventi successivi dovrò scrivere nel delta-time la metà del numero dei tick che rappresentano una nota da un quarto, cioè 48.

Se il bit 15 è uguale a 1 i delta-time faranno riferimento a come si suddivide il secondo, inteso come unità di tempo, in modo consistente con SMPTE e MIDI Time Code.I bit da 14 a 8 contengono i valori -24, -25, -29, -30 (espressi in complemento a due) che corrispondono agli standard SMPTE e MTC (con il -29 si identifica il formato 30 drop frame) e rappresentano il numero di frame per secondo. I bit da 7 a 0 (espressi normalmente) rappresentano la risoluzione all'interno di un frame, per esempio 4 cioè la risoluzione del MIDI Time Code.

Schematizzando

```
MThd <length of header data>
    <header data>
    MTrk <length of track data>
    <track data>\
    MTrk <length of track data>
    <track data>
    ...
```

Il Track-chunk (blocco di traccia), contiene gli eventi MIDI (nota on, nota off, ecc.) relative ad una singola traccia. Prima di ogni evento viene inserito il riferimento temporale (Delta-time), che esprime il tempo trascorso tra ogni singolo evento.
Un track chunk contiene uno stream di dati MIDI relativo ad un massimo di 16 MIDI canali.
I quattro caratteri ASCII di identificazione del blocco sono: MTrk.

| **Dati esadecimali** | **Commento** |
|---|---|

| | |
|---|---|
| 4D 54 72 6B | ASCII=MTrk |
| 00 00 00 00 | Lunghezza del blocco, in questo esempio 0 byte |

La sintassi è la seguente

```
<track data> = <MTrk event>+
<MTrk event> = <delta-time> <event>
```

- Delta-time

La lunghezza del Delta-time è variabile per consentire un intervallo tra un evento e l'altro anche molto grande, senza pregiudicare la compattezza del file.
Il valore del Delta-time viene calcolato suddividendo il tempo trascorso tra ogni evento, in pacchetti da 7 bit a cui vengono aggiunti dei bit più significativi di valore 1, tranne che per l'ultimo pacchetto a cui viene aggiunto il valore 0.
La scelta di questa complessa modalità di codifica del Delta-time è stata fatta per limitare la memoria stessa del MIDIfile e potere esprimere nel contempo valori di tempo in teoria infiniti.

Se due eventi sono simultanei il delta-time è zero ma deve essere comunque scritto.
Oltre ai messaggi riferiti al canale come quelli di nota, Control change, Program change ecc. (eventi MIDI), nel blocco Track-chunk, vengono memorizzati anche messaggi di System Exclusive, ed i Meta-eventi.
La memorizzazione dei dati di System Exclusive (Sys-Ex) avviene come per gli eventi MIDI, viene inserito prima il Delta-time, poi il dato di inizio del Sys-Ex (F0), seguito dalla lunghezza in byte dell'intero messaggio e infine dai dati.
In figura è descritto il formato di memorizzazione del Sys-Ex relativo al Reset General MIDI (**F0,7E,7F,09,01,F7**).

| **Dati esadecimali** | **Commento** |
|---|---|
| 00 | Delta-time |
| F0 | Inizio dei dati di Sys-Ex |
| 05 | Lunghezza in byte del Sys-Ex |

| | |
|---|---|
| 7E 7F 09 01 | Dati Sys-Ex |
| F7 | Fine dei dati di Sys-Ex |

Vediamone un esempio

Le istruzioni saranno segnate in questo modo

```
F0 <length> <bytes to be transmitted after F0>
F7 <length> <all bytes to be transmitted>
```

Ad esempio il seguente codice:

```
F0 03 43 12 00
  81 48                        200-tick delta-time
F7 06 43 12 00 43 12 00
  64                           100-tick delta-time
F7 04 43 12 00 F7
```

Invia 3 byte di dati del tipo 43 12 00. Quindi attende, invia 6 byte, attendo la metà del tempo precedente ed invia 4 byte.

Il messaggio F7 di fine SYS-Ex deve essere sempre presente

- I Meta eventi

I meta-eventi rappresentano un complemento importante negli SMF. Con essi vengono memorizzati: i nomi delle tracce, la divisione del tempo, la velocità, la tonalità, il testo del brano, il copyright e altro. Il problema di questi dati è che spesso sono mancanti o, soprattutto per quanto riguarda tonalità, tempo e altre informazioni musicali, del tutto arbitrari e lontani dalla reale rappresentazione in partitura.

Di seguito vengono descritti i Meta-eventi più importanti.

| **Tipo di Meta-evento** | **Commento** |
|---|---|
| 01 | Evento di testo per descrivere il nome di una traccia o il nome della voce usata, può essere utilizzato anche per inserire il testo |
| 02 | Evento di copyright, il testo deve contenere la © seguita dall'anno |

| | |
|---|---|
| | di creazione e dal nome dell'autore. Dovrebbe essere inserito come primo evento sulla traccia 1 |
| 03 | Evento di testo per indicare il nome della traccia. Se è inserito in un MIDIfile in formato 0 o nella prima traccia di un MIDIfile in formato 1, indica il nome del brano |
| 05 | Evento di testo per inserire le liriche del brano |
| 2F | Evento che indica la fine di un blocco di traccia (Track-chunk) |
| 51 | Evento di velocità del brano, indica quanti microsecondi ci debbono essere in un quarto di nota |
| 58 | Evento per descrivere la divisione del brano |
| 59 | Evento che indica la tonalità |

I Meta-eventi vengono memorizzati nello SMF allo stesso modo dei Sys-Ex, ma iniziano con l'esadecimale **FF**.

- I tre formati degli SMF

**Formato 0**
Le tracce di un brano vengono mixate in una singola traccia che contiene però tutte le informazioni degli eventi di tutte le tracce del brano.
**Formato 1**
Le tracce vengono memorizzate in modo singolo e contengono gli stessi valori di tempo e metrica. La velocità del brano viene inserita nella prima traccia che fa da riferimento a tutte le altre.
**Formato 2**
Le tracce vengono gestite indipendenti l'una da l'altra con valori anche diversi di tempo e metrica.
Generalmente i più utilizzati sono il Formato 0 usato principalmente dai sequencer a lettura diretta, cioè che non devono caricare in memoria l'intera sequenza, ma prelevano ed eseguono i dati MIDI direttamente. Ed il Formato 1 per i sequencer che possono creare e/o modificare SMF, specie i software musicali, i quali hanno in genere la possibilità di gestione di entrambi i formati.
Nella nostra applicazione, useremo esclusivamente il Formato 1.

### *4.4.2 – un esempio di file MIDI*

Se proviamo ad aprire un MIDIfile con un qualsiasi editor esadecimale possiamo verificare con precisione quanto detto finora.
Leggiamo ad esempio questo esempio preso da [XX]

```
4D 54 68 64    MThd
 00 00 00 06    chunk length
 00 01          format 1
 00 04          four tracks
 00 60          96 per quarter-note
```

**Il primo chunk contiene le indicazioni sulla segnatura**

```
4D 54 72 6B    MTrk
 00 00 00 14    chunk length (20)
 Delta-time     Event                       Comments
 ---------------------------------------------------------
 00             FF 58 04 04 02 18 08        time signature
 00             FF 51 03 07 A1 20           tempo
 83 00          FF 2F 00                    end of track
```

**Quindi, la prima traccia con note musicali.**

```
4D 54 72 6B    MTrk
 00 00 00 10    chunk length (16)
 Delta-time     Event                       Comments
 ---------------------------------------------------------
 00             C0 05 81 40
                90 4C 20 81 40
                4C 00             running status: note on, vel
= 0
```

```
 00              FF 2F 00                    end of track
```

**la seconda traccia**

```
 4D 54 72 6B     MTrk
 00 00 00 0F     chunk length (15)
 Delta-time      Event                       Comments
 ---------------------------------------------------------
 00              C1 2E 60
                 91 43 40 82 20
                 43 00                       running status
 00              FF 2F 00                    end of track
```

**e la terza**

```
 4D 54 72 6B     MTrk
 00 00 00 15     chunk length (21)
 Delta-time      Event                       Comments
 ---------------------------------------------------------
 00              C2 46 00
                 92 30 60 00
                 3C 60                       running status
 83 00           30 00          two-byte delta-time, running
status
 00              3C 00                       running status
 00              FF 2F 00                    end of track
```

## *4.5. – Il MIDI Parser*

Il recupero dei dati presenti nel file MIDI di partenza è stato affidato alla libreria di funzioni in C++ scritta in origine nell'anno 2000 da Dario Belloli e denominata *DBClasseMidi*. Tale libreria è disponibile presso il L.I.M. ed è stata utilizzata in altri progetti di tesi, subendo vari ampliamenti e modifiche [20]. Attraverso questo parser è possibile importare solo file MIDI di tipo 1.

### *4.5.1 – DBClassMidi*

Per utilizzare questa classe è necessario:
- aggiungere al progetto tutti i file forniti;
- includere nel proprio file d'intestazione (*header*) il file DBClasseMidi.h;
- dichiarare all'interno del proprio codice un'istanza della classe DBClasseMidi;

A questo punto è sufficiente invocare la funzione `CaricaMidiFile`(nomefile) perchè in memoria si allochi tutta la struttura dati con le informazioni del file MIDI. Il caricamento successivo di un altro file sostituirà i dati in memoria in modo automatico. Si segnala la possibilità di ripulire la memoria allocata manualmente tramite la funzione pubblica `DistruggiArray`().
Durante la fase di caricamento vengono aggiornate le seguenti variabili con informazioni di carattere generale:

- tipo_SMF; // 0, 1, 2 (viene caricato solo se =1)
- numTracceTotali;
- numTracceConNote;
- TickPerQuarti;
- lunghezza_SMF; // in Tick dall'inizio del brano all'ultimo Evento
- lunghezzaNote_SMF; // in Tick dall'inizio del brano all'ultimo Note OFF
- lunghezzaBattute_SMF;// in battute fino all'ultimo evento
- numBeatTotali_SMF; // numero di beat forti fino all'ultimo Note OFF

I dati allocati da CaricaMidiFile(nomefile) non includono:

- la quantizzazione degli start time (chiamato allineamento)
- la quantizzazione delle durate

- il riempimento dei campi evento.battuta, evento.beat, evento.tick

Queste operazioni possono essere invocate tramite le funzioni pubbliche:

- `Allinea(…)`
- `Quantizza(…)`
- `AssegnaBattuteBeatTick()`

Prima di ogni nuovo allineamento o quantizzazione, se interessano i valori di delta time esatti, è necessario chiamare la funzione `RipristinaDeltaOriginale`().
Se viene eseguita qualche operazione di quelle appena elencate, è necessario chiamare `AssegnaBattuteBeatTick`() per ricostruire l'esatta collocazione delle note all'interno della battuta.

La struttura principale in cui vengono memorizzate le informazioni per ogni singola nota è la struct evento. In essa troviamo i seguenti campi:

- *partenza*: è lo start time in tick (cioè la somma di tutti i delta degli eventi precedenti);
- *stop*: è lo stop time in tick;
- *partenza_Q*: simile a partenza, ma il valore è allineato;
- *stop_Q*: simile a stop, ma il valore è allineato;
- *durata_Q*: è riempito con la durata della nota (TICK) calcolata considerando la differenza tra il tick time di un NOTE ON e quello del suo NOTE OFF. Con il caricamento, questo campo viene riempito quindi con la durata reale dell'evento, mentre la quantizzazione lo modifica. E' possibile riottenere il valore originale tramite la riga di codice durata = evento.stop-evento.partenza, in quanto questi sono valori che non vengono mai modificati.
- battuta: indica il numero di battuta di appartenenza, a partire da 1. Viene aggiornata da `AssegnaBatBeatTick`();
- *beat*: parte dal valore 1; viene aggiornato da AssegnaBatBeatTick();
- *tick*: parte da 000 (quindi il primo Beat della prima battuta è 1:1:000); viene aggiornato da AssegnaBatBeatTick();
- *nota*: il valore numerico intero identificativo della nota MIDI, che varia nel *range* (0-127);
- *on*: è 1 (cioè TRUE) se l'evento è NOTE ON oppure 0 (FALSE) in fase di costruzione delle struttura;
- delta: il delta time dall'evento precedente (TICK). Non viene mai modificato.
- delta_Q: il delta modificato da allineamento e quantizzazione.

E' opportuno evidenziare che tali informazioni sono puramente esecutive, ben lungi dal poter essere considerate informazioni musicali simboliche.

Per scorrere gli eventi della partitura si adotta il seguente frammento di codice:

```
DBClasseMidi DBCmidi; //istanza della classe
int i,j; //indice tracce, eventi

// scorro le tracce
for (i=0; i<=DBCmidi.arTracce.GetUpperBound(); i++)
{
     DBCmidi.p_arEventi=((CObArray*)(DBCmidi.arTracce.GetAt(i)));
     // scorro gli eventi della traccia
      for (j=0; j<=DBCmidi.p_arEventi->GetUpperBound(); j++)
     {
DBCmidi.pEvento=((DBClasseMidi::evento*)
((DBCmidi.p_arEventi)->GetAt(j)) );

          //Ora si possono raggiungere i campi cosi':
          // DBCmidi.pEvento->nota
          // DBCmidi.pEvento->durata_Q
          // ...
     }
}
```

### *4.5.2 – modifiche effettuate alla classe*

Sono state effettuate alcune modifiche alla classe in questione. Questo perché alcuni valori basilari per la riuscita della traduzione non venivano restituiti.
Queste modifiche sono facilmente identificabili all'interno del codice DBClassmidi.cpp e DBClassmidi.h grazie al comando `#IFDEF`.

Rispettivamente

```
#ifdef MODIFICA_NOTE_OFF

#ifdef MODIFICA_PRESSIONE_TASTO
```

Esse riguardano il comando NOTE_OFF che nella libreria veniva soppresso quindi non restituito, ed è stata aggiunta la possiblità di ottenere all'esterno del codice l pressione tasto che era presente ma non esportabile.

# Capitolo 5

# MIDI2MX

Il cuore operativi di questo lavoro di tesi è rappresentato dal programma per PC MIDI2MX. L'applicazione converte nel formato MX file musicali MIDI.
Il vantaggio di usare file MIDI ci permette di partire da un archivio musicale estremamente vasto e libero. E di conseguenza, di ottenere con gran semplicità un archivio altrettanto vasto di partiture codificate in MX.
Le difficoltà da affrontare sono le seguenti

- **La non completa definizione e quindi il continuo aggiornamento del DTD MX**
- **La mancanza e l'ambiguità di alcune informazioni musicali derivanti dal MIDI**
- **La necessità di rendere l'applicazione "aperta"**

In realtà l'ultima annotazione potrebbe racchiudere in sé la problematica di questo lavoro.

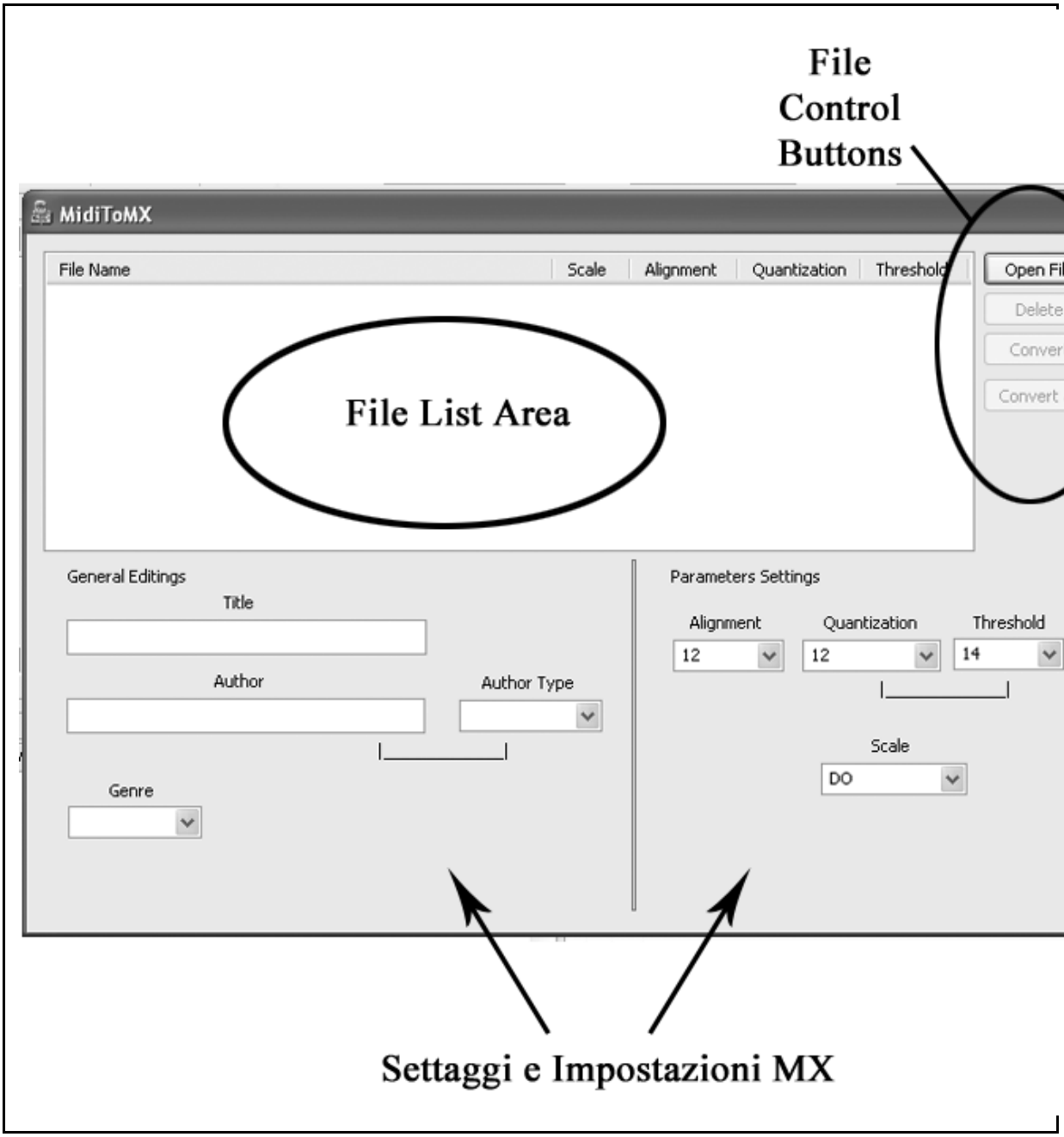
Dobbiamo avere la possibilità di aggiungere facilmente i nuovi layer che vengono definiti in MX o modificare quelli esistenti qualora vi siano soluzioni più funzionali.
E' opportuno non fossilizzarci sul file MIDI in quanto diversi sono i formati che raccolgono l'informazione musicale (si veda il cap. 4) quindi deve esser possibili aggiungere ulteriori parser oltre qullo preso in esame.
Inoltre, l'algoritmo di interpretazione deve essere certamente sviluppato ulteriormente andando di pari passo con la ricerca informatica musicale.

E’ da questi presupposti che nasce il progetto MIDI2MX, concepito attraverso una programmazione modulare, leggibile ed espandibile.

## *5.1 – L'applicazione MID2MX*

Illustriamo prima di tutto il programma ed il suo funzionamento

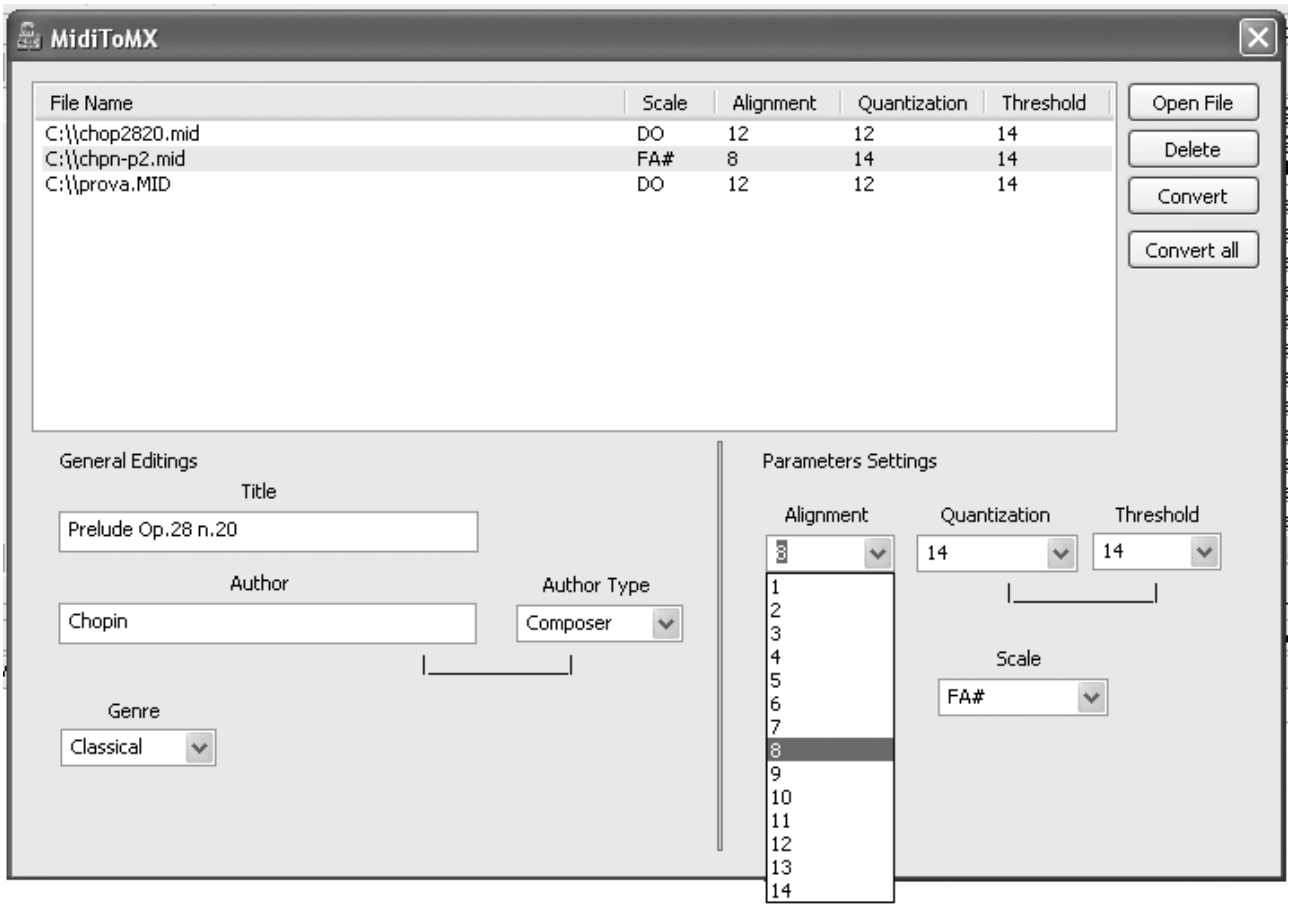
La finestra di lavoro è la seguente




La parte alta visualizza i file MIDI importati da convertire (File List)
In basso sono presenti le impostazioni di controllo MX (General Editings e Parameters Settings)

Infine sulla destra, i tasti per le operazioni di conversione o cancellazione dei file selezionati (File Control).

Cliccando su Open File si accede al menù esplora risorse, dove è possibile importare anche file multipli. E' possibile tuttavia caricare i file MIDI anche semplicemente trascinandoli in finestra con il mouse da una cartella qualsiasi.
Il numero di file massimo è di diverse migliaia.
Per depennare uno o più elementi dalla nostra lista, vi è il tasto *Delete*.
Una volta aperto i files possiamo selezionarli cliccandoci sopra o selezionandoli col mouse, esattamente come se ci trovassimo in una cartella di Windows.
Possiamo immediatamente convertire i file azionando il pulsante *Convert*. Mentre con *Convert All* traduciamo in formato MX tutti i MIDI caricati nell'applicazione.
Il programma usa le impostazioni di base per quanto riguarda i parametri (Parameters Settings). Essi sono Allineamento (alignment), Quantizzazione (quantization), Soglia (threshold) e Scala (scale). Per semplicità riferiamoci ad essi con la sigla *AQS*.
I dati lasciati in bianco nella parte destra (General Settings) portano al solo inserimento nel campo movement title di MX del nome del file importato.
Il significato di questo valori saranno esaminati più avanti.
In ogni caso, possiamo cambiare tali impostazioni selezionando i file interessati ed operando sulle combo box e le caselle di testo.
Nella File List saranno riportati i valori di tali settaggi.
Ora esaminiamo la figura




Possiamo notare che al secondo file (chpn-p2.mid) è stata assegnata una descrizione di titolo, autore e genere. Inoltre abbiamo scelto di convertirlo in MX seguendo la tonalità in Fa# e con 8,14,14 come valori di AQS.

Il file che produrremo verrà chiamato dall'applicazione

```
chpn-p2.mid_sFA#a8q14s14.mx
```

in cui come si vede vengono riportati i valori da noi selezionati.

Nulla ci vieta di importare molteplici istanze dello stesso file MIDI e convertirle ognuna con una diversa impostazione di Scale e AQS.
Il tempo di conversione di un file dipende dalla sua lunghezza ma possiamo dire che è pressoché immediata.

- General Editings

In questa sezione possiamo aggiungere delle informazioni sul brano che non riusciamo ad ottenere dal MIDI. Esse fanno tutte parte del layer General -> Description.
In primis, il titolo del pezzo musicale (Title). Questo dato sarà inserito all'interno di MX nell'elemento *movement_title*.

Quindi possiamo aggiungere l'autore e la sua specifica mansione (compositore, esecutore, arrangiatore o altro).
E' chiaro che trattandosi di file MIDI, l'esecutore materiale è la macchina, quindi nel caso raffigurato poc'anzi, potrebbe sembrare insulso aggiungere questa voce.
Tuttavia dobbiamo innanzitutto porci nell'ottica generale dove è possibile specificare questo dettaglio. Inoltre, vi sono varie eventualità dove questo fatto non è del tutto vero. Ad esempio potremmo descrivere un ipotetico "yesterday.mid" con "Beatles" come esecutori e "Lennon/McCartney" come compositori. Oppure considerare l'eventualità di una *cover*.
Ricordiamoci che non vogliamo semplicemente scrivere un *convertitore* da MIDI a MX, ma progettare un'applicazione che crei file MX, *sfruttando* nel caso specifico l'archivio MIDI.

L'altra voce presente è la casella *Genre*. Qui possiamo inserire il genere musicale del nostro brano.
Nella storia della musica e nella terminologia delle varie epoche sono state date molteplici descrizioni di un tipo di musica rispetto ad un altro. Soprattutto negli ultimi trent'anni possiamo dire di aver assistito ad una proliferazione, spesso arbitraria e confusionaria, di termini più o meno pertinenti.
Nel DTD non sono specificati quali e quanti generi possono essere categorizzati.
Certamente alcune distinzioni di massima possiamo farle, ed è quello che avviene nel nostro caso. Sono scelte volutamente generiche, tuttavia ci danno una prima indicazione sull'orientamento musicale dei pezzi che ci aggiungiamo a tradurre.

Possiamo quindi scegliere fra

- Antique
- Classical
- Rock

- Pop
- Jazz
- Blues
- Disco
- Electronic
- Metal
- Other

- Parameters Settings

Il lato destro dell'interfaccia è dedicato al controllo utente sulla qualità della conversione.
I parametri che possiamo variare sono la tonalità (scale) e i tre valori per stabilire le durate (AQS).

Il significato della tonalità è abbastanza immediato. Come spiegato nel capitolo 2, essa è l'insieme dei principi armonici e melodici che regolano i relativi legami tra accordi e/o note. Quindi, attraverso questo parametro possiamo ottenere il file MX del nostro brano in ciascuna delle 16 scale elencate.
Il file MIDI di origine è sempre in DO Maggiore, quindi il valore standard sarà appunto, "DO".

Più complesso è il significato della tripletta AQS.
Cominciamo col dire che essa è un opzione della libreria MIDIParser, che abbiamo illustrato nel precedente capitolo.
Attraverso di essa possiamo impostare la corretta configurazione delle durate.
Ricordiamoci che il criterio di base del MIDI è legato alla performance. Ciò vuol dire che al suo interno vi sono istruzioni su *come* suonare *cosa*. Ma al contrario della partitura tradizionale, i comandi sono univoci (per forza di cose…).
Caso classico è l'intervallo di tempo che intercorre tra le note (meglio dire eventi).
E' un tempo indicato in millisecondi, quindi assolutamente privo di interpretazione da parte del calcolatore. Per di più ogni file MIDI ha un suo Tick per Quarter Note, cioè l'unità di misura delle durate.

Dobbiamo invece cercare di ricondurci ad un tempo inteso come divisione, che come sappiamo non è affatto univoco.
I valori di AQS servono ad ottenere con miglior approssimazione a seconda dei casi, dei valori di durata riconducibili alla teoria musicale. Questo valore sarà espresso non come frazioni ma come VTU.

All'inerno del MIDIParser vi è un array che in base al valore del Tick Per Quarter Note tiene traccia di tutti i valori notazionali trattabili. E di conseguenza, ottiene il numero di tick per ciascun tipo di valore di una nota.
Questo array è denotato da un indice numerico (da 0 a 14) che racchiude i seguenti casi

- indice 0: 4 QUARTI
- indice 1: 3 Quarti
- indice 2: 2 QUARTI
- indice 3: 1 Quarto con Punto
- indice 4: 1 QUARTO
- indice 5: 1 Ottavo con Punto
- indice 6: 1 Quarto a terzina
- indice 7: 1 OTTAVO
- indice 8: 1 Sedicesimo con Punto
- indice 9: 1 Ottavo a Terzina
- indice 10: 1 SEDICESIMO
- indice 11: 1 Trentaduesimo con Punto
- indice 12: 1 TRENTADUESIMO
- indice 13: 1 Sessantaquattresimo con Punto
- indice 14: 1 SESSANTAQUATTRESIMO

Il parametro *allinea* ci permette di identificare uno di questo indici, all'interno dei quali opereremo la quantizzazione necessaria per ottenere l'esatto ( o quanto meno più vicino) grado di precisione.
Infatti attraverso *quantization* e *threshold* non facciamo altro che identificare il valore di durata massima e minima.

Facciamo un esempio:

se quantizziamo con Q=4, uno nota che dura ¼+1/8 sarà segnata con valore ¼ se S=4 e con valore 1/8 se S=7.
Nel primo caso l'ottavo "di troppo" viene troncato, nel secondo viene normalizzato ad ¼.

E' chiaro che in linea di massima sembrerebbe conveniente usare sempre valori non bassi. Tuttavia, vi sono molti esempi dove valori di un quarto sono più che sufficienti per descrivere le durate di note e pause.

In ogni caso nel nostro file MX risultante le durate saranno espresse in VTU.
Ciò significa che ci troveremo di fronte a valori interi dove multipli e sottomultipli della nota di valore minima ci indicheranno l'esatta divisione.

Così potremmo avere valori di 960 per la minima, 480 per la semiminima, 240 per la croma 720 per semiminima con punto e così via. Sempre a patto di aver usato dei valori pertinenti di AQS.
Si tratta quindi di un parametro molto importante per la buona riuscita della traduzione.

## *5.2 – La struttura dati*

Per scrivere l'applicazione è stato usato il linguaggio C++ nella versione .**NET 2002** della Microsoft. E sono state usate le Standard Library 1.1 (STD).

Nel descrivere la struttura dati usata, è bene dare un'occhiata al file MX prodotto.

```
- <mx version="1.7">
  - <general>
    - <description>
        <main_title>notturno </main_title>
        <author type="Performer">rubinstein </author>
        <genre>Jazz </genre>
      </description>
    </general>
  - <logic>
    + <spine></spine>
    - <los>
      - <staff_list>
        + <staff id="staff_-"></staff>
        </staff_list>
      - <part id="-" dfstaff_ref="staff_-">
        + <voice_list></voice_list>
        + <measure number="1"></measure>
        - <measure number="2">
          - <voice ref="voce_-_01">
            - <chord event_ref="evento_nota_-_01_000000" duration="48">
              - <notehead>
                  <pitch step="G" octave="5"/>
                </notehead>
              </chord>
            + <chord event_ref="evento_nota_-_01_000001" duration="0"></chord>
            </voice>
          + <voice ref="voce_-_02"></voice>
          </measure>
        + <measure number="3"></measure>
          ...
        </part>
      </los>
    </logic>
    <performance> </performance>
</mx>
```

E' una struttura gerarchica, in cui per tutti i layer ogni elemento ne racchiude altri. Alcuni elementi presentano degli attributi.

Scendendo in profondità arriviamo agli elementi nucleici, cioè che non si espandono a loro volta.

Vi sono elementi che si trovano sullo stesso livello di gerarchia di altri. Ogni layer ha i suoi specifici elementi.

In poche parole, abbiamo descritto la struttura dati di un *albero*.
Risulterebbe naturale a questo punto, implementare questa soluzione. In realtà la scelta della struttura dati è senza dubbio cruciale per i nostri fini. Sono state ponderate varie soluzioni prima di arrivare a quello che si spera sia il metodo più performante ed allo stesso tempo, chiaro.
Scartata quasi subito l'ipotesi di creare un *record*, l'obiettivo posto era quello di metter su una struttura che fosse una "fotocopia" di quella realizzata attraverso la grammatica MX.
In pratica ricreare in C++, layer, elementi e attributi con le gerarchie e i riferimenti illustrati nel capitolo 3.
Questa scelta permette due vantaggi

- poter definire con semplicità gli elementi di MX
- manipolare efficacemente e intuitivamente i nostri dati (cioè le informazioni musicali che andiamo a gestire)

Per ottenerla il primo passo è stato quello di costruire tante classi quanti sono gli elementi ed i layer di MX.

### *5.2.1 – Costruzione delle classi*

Ogni classe (oggetto) risiede su un singolo file .cpp con relativo header .h
Ogni file prende il nome dell'elemento MX rappresentato. La dicitura sarà MX*Nome_Elemento.*
Inoltre, per le classi, useremo il nome del file preceduto dal suffisso "C".
Ad esempio la classe CMXMovement_title[1] è così dichiarata nel file MXMovement_title.cpp:

```
#include "StdAfx.h"
using namespace std;
#include "mxmovement_title.h"


CMXMovement_title::CMXMovement_title(void)
{
}
```

[1] Nella versione definitiva, questa classe rappresenta l'elemento *main title* di MX

```
void CMXMovement_title::SetValues(CString newMovement_title)
{
	movement_title=newMovement_title;
}
```

```
class CMXMovement_title
{
	CString movement_title;
public:
	CMXMovement_title(void);
	~CMXMovement_title(void);

	void SetValues(CString newMovement_title);
};
```

In essa viene definito l'elemento MX **movement title**, figlio del layer **Description**, contenuto a sua volta nel layer **General**. Come si può notare è una semplice stringa, in quanto è tutto quello che ci occorre per questo elemento, non avendo attributi e potendo comparire una sola ed unica volta all'interno del nostro file.
Si noti anche nei file di libreria, l'uso delle *Standard Libraries* attraverso il comando `namespace` e l'inclusione dell'header mxmovement_title.

L'elemento padre come detto è Description, ed è una classe vuota nel file .cpp

```
#include "mxgenre.h"
#include "mxwork_number.h"
#include "mxwork_title.h"
#include "mxmovement_number.h"
#include "mxmovement_title.h"
#include "mxauthor.h"


CMXDescription::CMXDescription(void)
{
}
```

Infatti contiene unicamente le chiamate alle librerie degli elementi da noi definiti, elencati nel file MXDescription.h

```
class CMXDescription
{

public:
	CMXGenre genre;
	CMXAuthor author;
	CMXWork_number work_number;
	CMXWork_title work_title;
	CMXMovement_number movement_number;
	CMXMovement_title movement_title;
```

Questi sono gli elementi contenuti nel layer Description.
Possiamo subito evincere che l'aggiunta e la modifica degli oggetti MX di questo tipo è estremamente semplice attraverso questa implementazione.
Fin qui tuttavia, abbiamo visto solo elementi per così dire "statici". Guardiamo la classe CMXEvent

```
class CMXEvent
{
	CString id;
public:
	int timing,hpos;
	~CMXEvent(void);

	void SetValues(CString newId, int newTiming=MX_NULL, int
newHpos=MX_NULL);

	CString GetID()
	{
		return this->id;
	};
}
```

In MX **Event** è composto dall'id[2] e dai valori di hpos e timing. Ve ne possono essere in numero imprecisato all'interno dello spine, di elementi event. Quindi all'interno di CMXSpine l'istanza CMXEvent verrà creata come segue

```
#include "MXEvent.h"

class CMXSpine
{
	vector <CMXEvent*> m_Events;
public:
	CMXSpine(void);
	~CMXSpine(void);

	void AddEvent(CMXEvent* newEvent)
	{
		this->m_Events.push_back(newEvent);
	}

};
```

L'oggetto event è quindi un vettore. Un vettore di puntatori nel caso specifico.
Rispetto all'array, i vettori hanno il vantaggio di adattare la dimensione della loro cella alla lunghezza del dato.
Nel corpo del codice vi è anche la funzione di inserimento di un evento (AddEvent) che viene inserito nella coda del vettore attraverso la chiamata `push_back`. Ci occuperemo tra poco di questo aspetto. Per ora basti dire che useremo una funzione `AddOggetto` anche in altre classi per aggiungere tutti gli elementi di cui avremo bisogno nella fase di riempimento del nostro albero.
Già nella classe CMXGeneral possiamo vederne un'applicazione

```
CMXGeneral::CMXGeneral(void)
{
}

void CMXGeneral::AddMovement_title (CString title)
{
```

[2] Id è definito come *private*. La funzione getID ne permette la manipolazione al di fuori della sua classe di origine. Useremo la funzione GetXX ogniqualvolta vorremo che XX sia visibile anche all'esterno

```
	this->m_Description.movement_title.SetValues(title);
}

void CMXGeneral::AddAuthor(CString type, CString name)
{
	this->m_Description.author.SetValues(type,name);
}
…
```

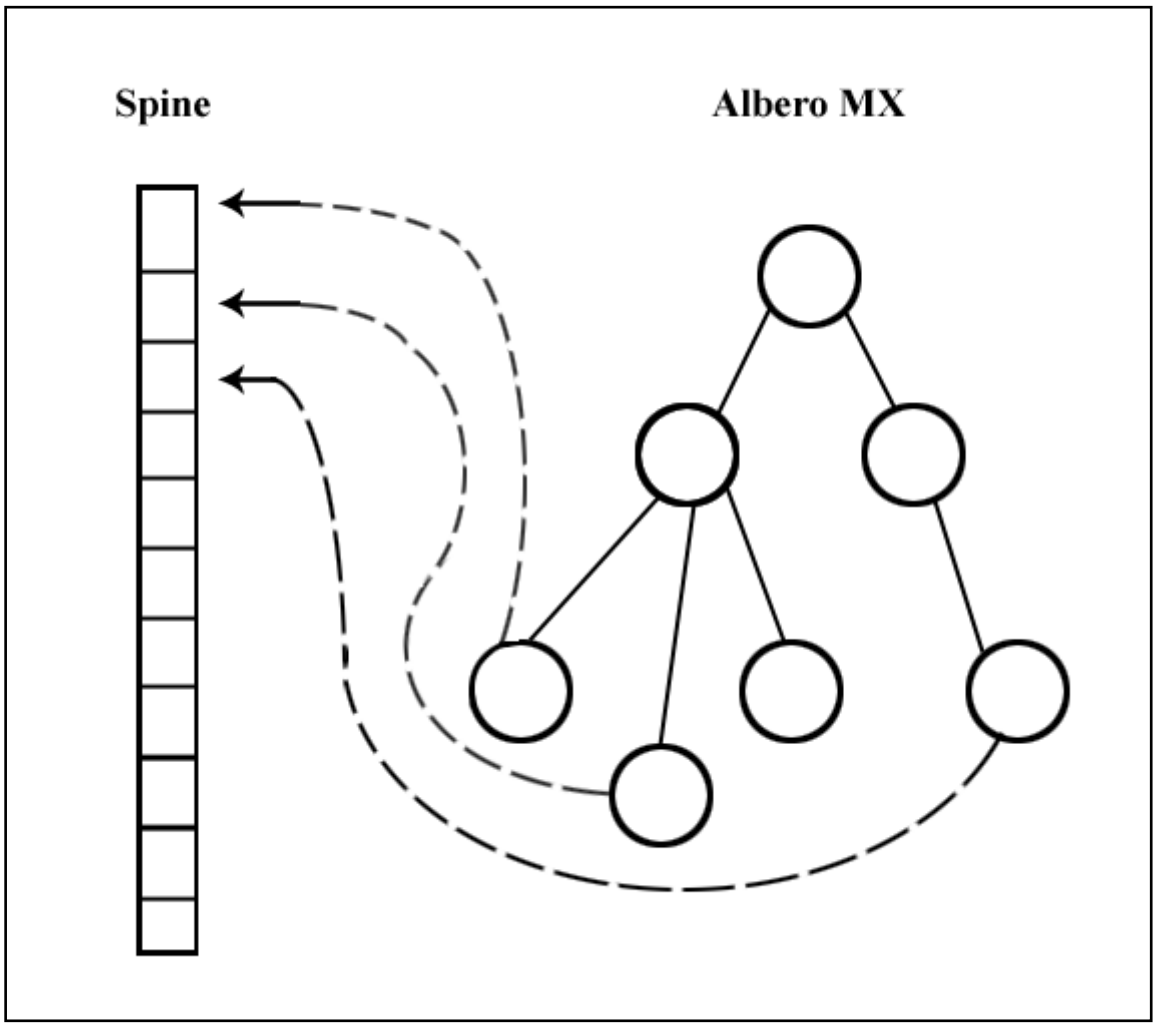
dove aggiungiamo all'interno di Description, gli elementi Main_Title, Author e così via.
In ogni modo, scriviamo tanti file e tante classi quanti sono gli oggetti che ci servono.
L'alta modularità di questa scelta ci permette di creare quanti elementi di MX vogliamo.
Nel momento in cui nella grammatica di MX viene aggiunta una voce, o ampliata, o anche solo semplicemente cambiata una dicitura, non dovremo far altro che aprire o creare la classe relativa e apporre le relative modifiche o aggiunte. La programmazione modulare ci permette questa semplicità di procedura.

### *5.2.2 – tipi di dati in C*

Una volta definite le classi, dobbiamo preoccuparci di come gestire il loro contenuto. Quindi la scelta del tipo di dato da usare, in base alle differenti possibilità.
In linea di massima, ci occuperemo principalmente di numeri interi e stringhe. Anche perché useremo i nostri tipi di dati sopra definiti (oggetti di tipo staff, di tipo voice, di tipo notehead ecc.). Tuttavia ciò su cui dobbiamo ragionare non è tanto se rappresentare un identificatore con un numero progressivo piuttosto che con un nome. Quanto al numero di occorrenze del dato in sé, alla sua dimensione ed ai possibili collegamenti con altri dati. Non dimentichiamoci che all'interno di MX note, pause e segni in genere sono rigorosamente mappati nello Spine. Ciò significa che nel momento in cui noi aggiungiamo una nota, perché il file MIDI ci dice che ad un certo punto lì c'è una nota, essa va non solo specificata con tutti i suoi attributi grafici all'interno di **Los**; dovremo anche occuparci di segnarla come evento all'interno di **Spine** e soprattutto, darle la sua giusta collocazione spazio-temporale.
Se per la descrizione del titolo del brano o del genere musicale, ci limitavamo a creare un'istanza singola di CMXMain_Title piuttosto che di CMXGenre, ciò non è più sufficiente per elemento di altro tipo. Come appunto `event`, o ad esempio `measure`.

Abbiamo precedentemente accennato ai vettori. Ci riferiamo alla libreria `<vector>` di STD. L'idea è quella di creare vettori di oggetti ogniqualvolta *non conosciamo a priori* l'esatto dimensionamento di un elemento. Non tanto come grandezza del singolo oggetto, quanto a numero di occorrenze.
Infatti, non ci è dato sapere da quante battute è composto un brano e da quante note. Così come non possiamo sapere se ci sarà un unico evento o nessuno. O mille.
Mentre sappiamo che il titolo sarà sempre e solo uno[3].
E' chiaro che in questa situazione di alta variabilità, l'uso del vettore ben si sposa con le sue caratteristiche. Anche perché rispetto ad un array possiamo contare su una gestione automatica della memoria. Inoltre lo scorrimento delle celle del vettore viene fatto fatta attraverso iteratori, con tutti i vantaggi che ne conseguono.
Abbiamo risolto il problema di creare dinamicamente più istanze dello stesso oggetto.
Tuttavia non basta. Se ripensiamo per un momento alla struttura dell'MX possiamo notare che essa è sì un albero, ma nello stesso tempo esso contiene al suo interno una struttura come lo Spine che è di tipo sequenziale e per di più strettamente collegata con tutti i nodi foglia al di fuori dello Spine stesso.
Stiamo parlando del meccanismo di referenze fra gli eventi e la loro caratterizzazione.
Tutto ciò che appare in partitura, MX lo mappa nello Spine secondo l'esatto ordine spazio-temporale dettato dalla lettura musicale. Inoltre, tutti questi eventi saranno associati alla loro precisa descrizione presente nell'elemento di appartenenza.
Quindi il primo evento presente nello spine (mettiamo che sia la chiave di violino del primo pentagramma dell'accollatura del pianoforte) conterrà l'identificatore del tipo clef*_numero di clef_nome staff_numero voce*, la sua posizione sul pentagramma *hpos* e la sua durata di esecuzione *timing*. Che per quanto riguarda gli eventi diversi dalla note e dalle pause è NULL.
A sua volta l'elemento **Staff**, contenuto in **Staff_list**, conterrà un'istanza di **Clef** con attributi che identificheranno la forma della chiave (*type*), la sua posizione sul pentagramma (*staff_step*) e soprattutto il riferimento univoco con l'evento nello Spine (clef*_numero di clef_nome staff_numero voce)*. E' attraverso questo metodo che noi riusciamo a ricondurre la chiave di violino descritta dall'elemento Clef con *quella* chiave presente nello spine.
Per fare il punto, proviamo a schematizzare la situazione da affrontare

[3] Come si evince dalla lettura del DTD di MX.



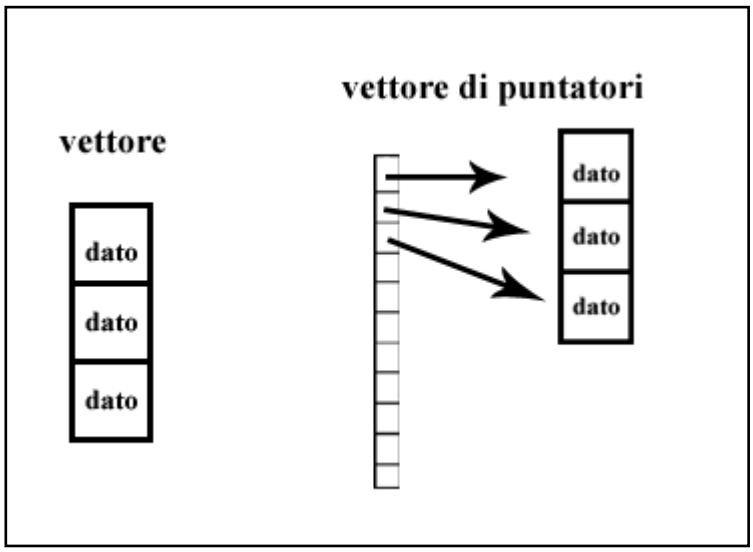

Ogni foglia del nostro albero, dovrà essere necessariamente mappato nello spine, il quale si presente sottoforma di array sequenziale. Si badi che le linee tratteggiate sono puramente indicative, come d'altronde tutto il disegno stesso. Lo spine infatti è anch'esso un nodo del nostro albero. Tuttavia a livello implementativo, esso è strutturato come vettore.

### *5.2.3 – vettori e mappe*

In questa parte analizziamo più approfonditamente le due strutture sopra citate. Esse derivano dallo studio delle *std,* le Standard Libraries di C++. Queste librerie permettono di usare funzioni estremamente utili per la costruzione e manipolazione di dati e contenitori di dati.

Partiamo nuovamente dal presupposto della struttura gerarchica. Abbiamo dato la configurazione di albero alla grammatica MX, ma abbiamo anche detto che la dimensione del numero di figli di un dato nodo può essere estremamente variabile.

Sarà inoltre necessario non solo creare questi nodi-figlio, ma anche accedervi ed eventualmente scrivere al loro interno più volte durante la costruzione del nostro albero.
In definitiva si è scelto di creare vettori o anche vettori di puntatori per tutti quegli oggetti che potevano occorrere in numero strettamente maggiore di 1. Negli altri casi, la classe specifica conteneva o unicamente i richiami per gli elementi figlio o i parametri propri delle sua funzione all'interno di MX.
Nel dettaglio, noi creiamo vettori di oggetti nel caso di **Event, Voice, Part, Measure, Clef, Time Signature, Key Segnature, Staff_id, e Chord.**
Sono tutti elementi che in MX possono occorrere in numero variabile.
Le altro voci, ad esempio le descrizioni date in *Description*, vengono istanziate una volta soltanto.
La possibilità di creare vettori di puntatori ci permette una più rapida gestione dell'acceso ai dati, dovendo memorizzare solo l'indirizzo della nostra variabile e non tutto il suo contenuto.




In effetti useremo vettori di puntatori in tutti casi sopra elencati, con un'unica eccezione, all'interno della classe CMXVoice_list

```
class CMXVoice_list
{
      vector <CString> voice_item;
      VOICE_ITEM_MAP map_voice_item;

public:
      CMXVoice_list(void);
```

```
    ~CMXVoice_list(void);

    void Add_voice_item(CString voice_item);
    CString GetVoiceItem(CString voice_item);
    CString GetVoiceItemByNumber(int n);
    int GetCount();
```

Come si può vedere, viene inizializzato un vettore di stringhe, in quanto voice_item è una variabile di tipo stringa.

In taluni casi, occorrerà scendere in profondità nell'albero per inserire un dato in una determinata posizione del vettore interessato. Questa operazione può avvenire diverse centinaia di volte nell'arco di una singola conversione di un brano. E dovrà ovviamente essere aggiornata la struttura di riferimento, più come al solito, lo spine.
Per rendere più rapido ed efficace questa situazione, la seconda struttura usata delle std è la <map>.
Abbiamo usato questo particolare costrutto 5 volte. Per le Part, i Voice_ref, le Measure, gli Staff_ID e i Voice_item.
Ecco come si presenta la dichiarazione all'interno di MXMeasue.h

```
typedef map <long, class CMXMeasure*> MEASURE_ID_MAP;
```

La particolarità della map, è la possibilità di ricercare l'elemento in una lista attraverso una chiave di ricerca. In effetti, una struttura definita come sopra è composta da due elementi. La variabile che specifica la chiave di ricerca, e il tipo di dato da memorizzare.
Nel caso sopra elencato tutte le battute vengono inserite ed ordinate per numero. Il vantaggio sta nel momento in cui dobbiano inserire una nuova battuta nel nostro vettore.
Attraverso le funzioni *find* e *insert*, cercare il punto di inserimento diventa estremamente semplice, ma soprattutto rapido.

```
CMXMeasure* CMXPart::GetMeasureByNumber(long number)
{
    MEASURE_ID_MAP :: iterator it=this-
>map_measure.find(number);
```

```
     if ((it!=this->map_measure.end()) && ((*it).second-
>GetNumber()==number))
          return (*it).second;
     else
          return NULL;
```

```
void CMXPart::Add_measure(CMXMeasure* new_measure)
{
     this->m_measure.push_back(new_measure);
     this-
>map_measure.insert(MEASURE_ID_MAP::value_type(new_measure-
>GetNumber(),new_measure));
}
```

### *5.2.4 – la funzione GenerateMXText*

Una volta riempita la struttura, dobbiamo preoccuparci di renderla fruibile generando un output. Ciò significa, scrivere un file XML.
Ogni classe che descrive un layer o un elemento di MX , ha al suo interno una funzione, chiamata `GenerateMXText`, che genera la corrispettiva riga in un file testo, al quale assegneremo l'estensione MX. E' una funzione estremamente semplice, da come si evince dalla figura.

```
CString CMXSpine::GenerateMXText()
{
      CString buf;
      CString str;
      char tmpbuf[10];
      buf.Append("\t\t<spine>\r\n");

      for (vector <CMXEvent*>::iterator it=this-
>m_Events.begin();it!=this->m_Events.end();it++)
      {
            buf.Append("\t\t\t<event id=\"");
            str.Format("%s",(**it).GetID());
            str.Replace(" ","_");
```

```
            buf.Append(str);
            buf.Append("\" timing=\"");
            if ((**it).timing==MX_NULL)
                  buf.Append("NULL");
            else
                  buf.Append(_ltoa((**it).timing,tmpbuf,10));
            buf.Append("\" hpos=\"");
            if ((**it).hpos==MX_NULL)
                  buf.Append("NULL");
            else
                  buf.Append(_ltoa((**it).hpos,tmpbuf,10));

            buf.Append("\"/>\r\n");
      }
      buf.Append("\t\t</spine>\r\n");
      return buf;
}
```

Attraverso funzioni SVG possiamo creare elementi grafici partendo da file XML. Tuttavia ciò che ci interessa è che avremmo potuto inserire al posto di GenerateMXText, una funzione chiamata ad esempio GenerateMXGraph, che attraverso l'uso di qualsiasi primitiva grafica creasse la suddetta corrispondenza grafica. Anche questo è un vantaggio della modularità

### *5.2.5 – creazione della struttura*

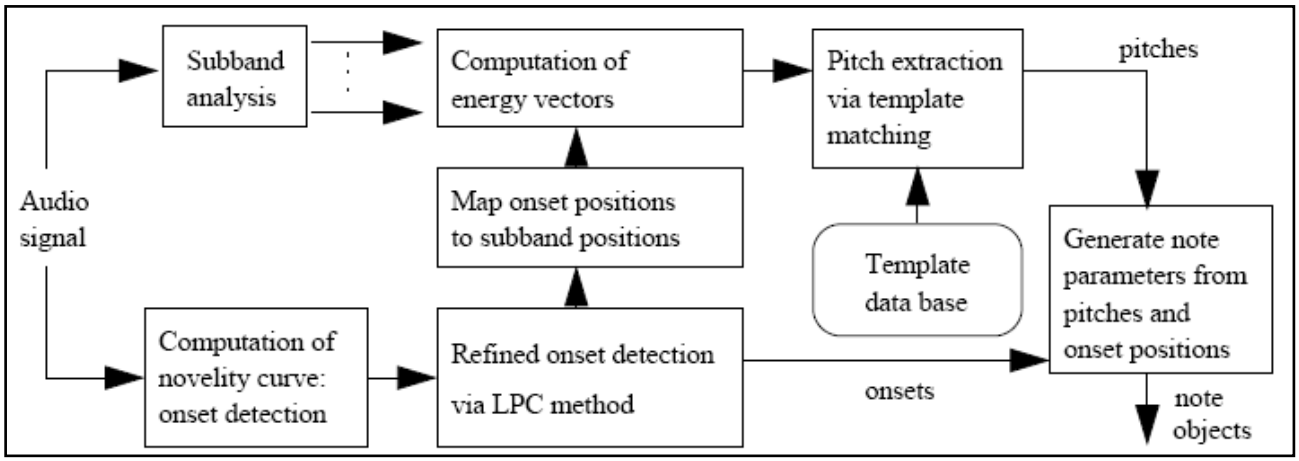
Nel momento in cui importiamo il nostro file MIDI ed avviamo la conversione, la struttura dati viene inizializzata. E' un procedimento che avviene bottom-up. Ciò vuol dire che viene prima creato un elemento foglia, quindi il nodo padre, il padre di quest'ultimo e così via. Per comprendere meglio questo procedimento, immaginiamo di creare un'elemento di tipo "matita". A quel punto, dovremmo verificare se esiste un elemento di tipo "portapenne", in cui inserire "matita". Se non c'è, va creato. Una volta fatto, eseguiamo lo stesso procedimento cercando l'elemento "scrivania" su cui riporre "portapenne". Proseguiamo fino al completamento, in base comunque, ai dati che riceviamo in input.
Costruito lo scheletro, si procederà al riempimento. Un dato in ingresso, verrà inserito nell'albero con strategia top-down. Verifichiamo prima il nodo padre e poi via via tutti i figli fino al raggiungimento dell'esatta collocazione. A quel punto, se ci sarà un vettore come

corrispondenza di quella collezione di elementi, l'inserimento verrà effettuato seguendo le procedure elencate nel paragrafo riguardante i vettori e le mappe.
E' nelle singole classi che vengono fatti tutti gli aggiustamenti derivanti dal modello interpretativo. In questo modo si è relativamente indipendenti dal parser usato per importare l'imput ed è più facile atturare modifiche al codice.

## *5.3 – Il modello interpretativo*

### *5.3.1 – cos'è?*

Immaginiamo di dividere la rappresentazione musicale su tre livelli [24].
Il primo riguarda l'aspetto fisico del suono, quindi musica sottoforma di segnali analogici.
Il secondo, rappresentato dal MIDI, consiste in una serie di eventi musicali i cui i valori numerici sono ben definiti (come le durate e l'altezza delle note).
Il terzo livello, è quello della notazione musicale secondo la teoria occidentale data dalla semiografia. La partitura.
Un modello interpretativo è composto da una serie di algoritmi che riescano ad ottenere una rappresentazione musicale del livello 3, partendo dal livello 1.
Poter ricavare informazioni musicali direttamente dalla forma d'onda è uno dei problemi più spinosi dell'informatica musicale, nonostante i numerosi passi avanti fatti negli ultimi anni.
Il procedimento, generalmente si basa sul riconoscimento del punto di attacco delle note (*onset detection*), e dalla loro altezza in frequenza. Il problema si complica per segnali polifonici, per ovvi motivi. La figura, presa in prestito da [XXX] mostra uno schema abbastanza usato per l'estrazione di note da segnali audio.




Tre sono quindi, i parametri essenziali da considerare:

- l'altezza (pitch)
- l'attacco (onset)
- la durata (duration)

Inoltre, c'è da considerare il timbro dello strumento. A livello notazionale è irrilevante, ma ogni strumento ha una sua forma d'onda precisa e di conseguenza, la ricerca dei tre valori sopra elencati dipende in ogni caso dalla conoscenza di questo dato.
Per il riconoscimento del pitch, ad esempio si può usare la funzione di correlazione

$$r_{xx}(n) = \frac{1}{N} \sum_{K=0}^{N-n-1} x(k) \cdot x(k+n)$$

A quel punto, si può assegnare a tale frequenza, una nota MIDI ed una prima conversione dal livello 1 a livello 2 è attuata. [XXX]
In questi ultimi anni, vari metodi sempre più sofisticati permettono una maggiore accuratezza nel riconoscimento audio. La letteratura è ampia. Sul sito del ISMIR[4] (www.ismir.net) si possono trovare molti degli ultimi studi in materia.

### *5.3.2 – il nostro modello*

Dovrebbe risultare chiaro che in questo lavoro di tesi, ci occupiamo del passaggio dal livello 2 al livello 3. Il passaggio diretto dall'1 al 3 è al momento scarsamente efficace. Questo perché per avere risultati apprezzabili, bisogna partire da segnali audio molto semplici e di conseguenza, si ottengono trascrizioni estremamente semplici. Si veda per l'occasione nuovamente [XXX].
Certamente, la ricerca non si ferma [XXX], tuttavia partendo già da file MIDI, che sono in tutto e per tutto brani musicali, possiamo provare ad ottenere partiture degne di questo nome, sfruttando tra l'altro un libero archivio smisurato ed eterogeneo.
A livello concettuale, le problematiche sono le stesse.
C'è una questione generale, che è quella di ottenere informazioni interpretabili (la notazione) partendo da informazioni già interpretate (l'esecuzione MIDI)
E c'è una questione secondaria, se vogliamo figlia della prima, riscontrabile nel fatto di dover interpretare messaggi MIDI, in eventi musicali tradizionali.

### *5.3.3 – interpretazione degli eventi MIDI*

[4] The International Conferences on Music Information Retrieval and Related Activities

Elenchiamo di seguito i principali eventi MIDI che sono stati interpretati dall'applicazione e la metodologia usata.

- **Interpretazione delle note**

Quello delle altezze è in teoria la parte più immediata. E' vero che nel MIDI, come visto nel capitolo 4, ogni altezza è denominata con un numero univoco, in contrasto con quanto visto nel capitolo 2. Tuttavia una volta selezionata la tonalità, che nella nostra struttura è definita con una doppia `static const` di caratteri con il nome di tutte le note per quella scala, l'operazione consta in una semplice divisione in modulo 12 per le note, alterazioni e ottave.

```
static const char* szNote[] = { "C", "C", "D", "D", "E", "F", "F", "G",
"G", "A", "A", "B" };

static const char* szNoteAlter[] = { "empty", "#", "empty", "#", "empty",
"empty", "#", "empty", "#", "empty", "#", "empty" };
```

```
newNotehead->m_octave=ltoa((byte)(DBCmidi.pEvento->nota/12),buf,10);
                                newNotehead-
>m_pitch=szNote[(byte)((DBCmidi.pEvento->nota)%12)];
                                newNotehead-
>m_alter=szNoteAlter[(byte)((DBCmidi.pEvento->nota)%12)];
```

- **Interpretazione delle durate**

Questa sezione è più spinosa. Considereremo i tempi espressi sempre nell'ordine di VTU. Tralasciando le possibilità parametriche viste in precedenza, ragioniamo sul fatto che il parser ci restituisce due tempi principali:
La durata totale del brano, che parte dallo start=0.
La durata della nota (quantizzata).
Con questi dati dobbiamo non solo assegnare la durate della singola nota che andiamo a descrivere, ma anche la sua posizione temporale rispetto a tutte le altre, ma soprattutto come questa si delinea graficamente rispetto alle battute.

Ad esempio, supponiamo di avere una nota di durata "1024", all'inizio della battuta 1. Sappiamo che la durata totale della singola battuta è "512". Nella nostra struttura, e quindi

nell'MX, dobbiamo inserire una nota di durata 512 nella battuta 1, e un'altra identica nella battuta 2. Con in più l'attributo della legatura di valore.

```
Delta-time      Event
 ------------------------
 00             90 60 20
 1024           80 60 20
```

Fig. 1 - Istruzione MIDI

Fig. 2 - la ricostruzione da effettuare

Nel codice, il primo passo è verificare che la durata della nota non superi quello della battuta. In caso affermativo devo anche preoccuparmi di inserire la legatura di valore (tie)

```
if (end>(m-1)*measureDuration) {
//se la durata supera la durata della battuta, tronco la nota
      unsigned long resto=end-(m-1)*measureDuration;
      char buf[20];
      if (currNote->m_rest.IsEmpty()) {
            currNote->m_duration=ltoa((m-1)*measureDuration-start,buf,10);
            currNote->m_tie=TRUE;
      }
      else
            currNote->m_rest=ltoa((m-1)*measureDuration-start,buf,10);
```

Se è maggiore, il resto di tale durata viene assegnato al nuovo elemento chord, che avrà uguale altezza. Ovviamente dovrà essere creato anche il relativo riferimento allo spine

```
//l'eventuale resto della nota troncata, l'aggiungo nella battuta corrente
CMXEvent* newEvent=new CMXEvent((m-1)*measureDuration);
CString event_id;
CString strumento=prevVoice->GetRef().Mid(5);

      event_id.Format("evento_nota_%s_%.06d",strumento,firstEv++);
            newEvent->SetValues(event_id,MX_UNDEFINED,MX_UNDEFINED);
            this->GetMSpineRef()->AddEvent(newEvent);
            CMXChord *newNote=new CMXChord();
```

```
            newNote->m_duration=ltoa(resto,buf,10);
```

- **Interpretazione delle pause**

Il MIDI non restituisce pause, ma attraverso i NOTE OFF interrompe una data nota. E' un primo inizio, ma non sufficiente. Possiamo benissimo avere in musica, intere battute prive di note, soprattutto in polifonia o in presenza di strumenti con più voci (uno su tutti il pianoforte).
Sono informazioni di cui il MIDI è del tutto sprovvisto, quindi tocca ancora a noi inserire questo dato. Il parser ci restituisce il numero della prima battuta con note. Quindi, basta controllare quali sono quelle mancanti per poterle riempire, creandone di opportune ed inserendole nel posto giusto. Ovviamente, saranno battute che conterranno solo pause di durata stabilita. Quella ricavata dalle battute con note.

```
if (currVoice->GetNote(1)==NULL)
                        {
                              char buf[20];
                              long partenza=(m-1)*measureDuration;
                              CMXEvent* newEvent=new CMXEvent(partenza);
                              CString event_id;
                              CString strumento=currVoice->GetRef().Mid(5);

      event_id.Format("evento_nota_%s_%.06d",strumento,firstEv++);
      newEvent->SetValues(event_id,MX_UNDEFINED,MX_UNDEFINED);
      this->GetMSpineRef()->AddEvent(newEvent);

      CMXChord *newNote=new CMXChord();
      newNote->m_duration=ltoa(measureDuration,buf,10);
      newNote->SetValues(ltoa(measureDuration,buf,10),NULL,NULL,newEvent);
      currVoice->AddNote(newNote);
                        }
```

- **Interpretazione della dinamica**

Il MIDI fornisce la pressione data alla singola nota. In questo modo possiamo ottenere la dinamica. Ciò che si deve interpretare, a parte il range di valori in cui far cadere tale descrizione è quando inserire in partitura questo dato.

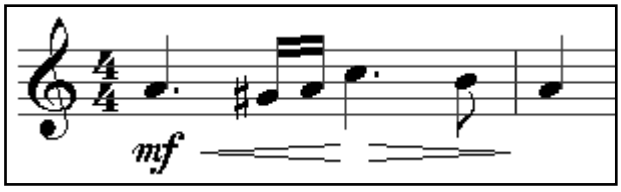
Il valore restituiti dal parser è un valore intero fra 0 e 127. E' il parametro velocità del NOTE ON.

| | |
|---|---|
| Status & Channel Number (0-15)[5]<br>1001 CCCC | I primi 4 bit identificano il messaggio<br>I secondi 4 bit identificano il canale |
| Note Number<br>0NNN NNNN | Il numero della nota (0..127) |
| Velocity<br>0VVV VVVV | Valori da 0 a 127 |

In primis, raggruppiamo le indicazioni di dinamica in questo range.
Ad esempio

| |
|---|
| ***ppp*** = più che pianissimo (da 1 a 10) [6] |
| ***pp*** = pianissimo (da 11 a 20) |
| ***p*** = piano (da 21 a 40) |
| … |
| ***fff*** = più che fortissimo (da 120 a 127) |

Segneremo questo dato sempre sull'attacco della prima nota della prima battuta, e quando questo valore cambia da un range all'altro. Naturalmente, è opportuno che ci siano anche altri criteri. A parte il fatto che la scelta dei valori segnati in figura è indicativa, ma resa parametrica dall'interfaccia dell'aplicazione MIDI2MX, nel caso in cui il cambiamento della dinamica sia graduale, sarebbe opportuno inserire le forcelle di dinamica.
Sono simboli che all'interno di MX vengono codificati nel layer **LOS**, nell'elemento **dynamic**, figlio di **horizontals_symbols.**




- **Interpretazione della tonalità e della segnatura di chiave**

[5] E' la definizione di Status byte e non verrà più segnalata
[6] ricordiamo che un NOTE ON con key velocity=0 viene considerato come un NOTE OFF.

Queste informazioni non sono presenti nel MIDI. Esistono degli algoritmi per ricavare la tonalità, tra cui il *Theme Finder*, sviluppato presso il LIM in cui se ne può vedere un'applicazione in [XXX].
Nel nostro caso, ci limiteremo ad offrire all'utente la possibilità di scegliere fra le 15 scale disponibili. Il pezzo sarà tradotto seguendo quella tonalità per tutta la sua durata.
Questa scelta influenza ovviamente anche l'armatura di chiave che sarà arricchita di conseguenza.
Per la segnatura di tempo, viene impostato il 4/4 canonico.
Per la chiave, viene usata la chiave di violino, mentre in caso di più voci (es. il pianoforte o l'organo) viene usata anche la chiave di basso.
In ogni caso, i tre elementi Clef, Key_Signature e Time Segnature, vengono creati e posti all'inizio del brano per ogni voce presente. Inoltre, grazie ad una funzione della libreria DBClassMIDI, viene presa traccia anche dei cambi di segnatura.

- **Interpretazione delle voci**

Strumenti polifonici come il pianoforte o l'organo vengono descritti in partitura con più voci (pentagrammi), uniti da un'accollatura. Il MIDI dispone di 16 canali, in cui ognuno ha a disposizione una voce. Almeno questo vale per i file di tipo 1. Gli eventi vengono inviati in streaming partendo dal primo canale.
Per disporre correttamente i dati nella struttura, ci avvaliamo prima di tutto del messaggio Program Change e in seconda istanza, della tabella General MIDI. In questo modo riconosciamo il "timbro" dello strumento e possiamo suddividere i part di MX.

```
CMXPart *tmpPart;
      tmpPart=this->Logic.GetMLos()->GetPart(DBCmidi.p_traccia-
>strumento);
      if (tmpPart==NULL)      //Qui aggiungo gli strumenti
                  {
                  CMXStaff_id *newStaff_id=new CMXStaff_id;
                  CString str;
                  str.Format("staff_%s",DBCmidi.p_traccia->strumento);
                  newStaff_id->SetValues(str);
                  this->Logic.Add_staff(newStaff_id);
                  CMXPart* newPart=new CMXPart(DBCmidi.p_traccia-
>strumento);
```

```
                  newPart->SetStaff_ref(newStaff_id);
                  this->Logic.GetMLos()->Add_part(newPart);
                  tmpPart=this->Logic.GetMLos()-
>GetPart(DBCmidi.p_traccia->strumento);
                        str.Format("voce_%s_01",DBCmidi.p_traccia-
>strumento);
                        tmpPart->Add_voice(str);
      }
```

Selezionati gli strumenti, ci occupiamo delle voci. Ci avvaliamo di contare i canali che suonano lo stesso strumento. Ognuno di questi canali sarà visto come una voce appartenente all'accollatura del suddetto strumento.

```
else // se uno strumento è già stato assegnato, ci occupiamo delle voci
      {
      CString str;
      str.Format("voce_%s_%.02d",DBCmidi.p_traccia->strumento,tmpPart-
>GetVoiceCount()+1);
      tmpPart->Add_voice(str);
      }
```

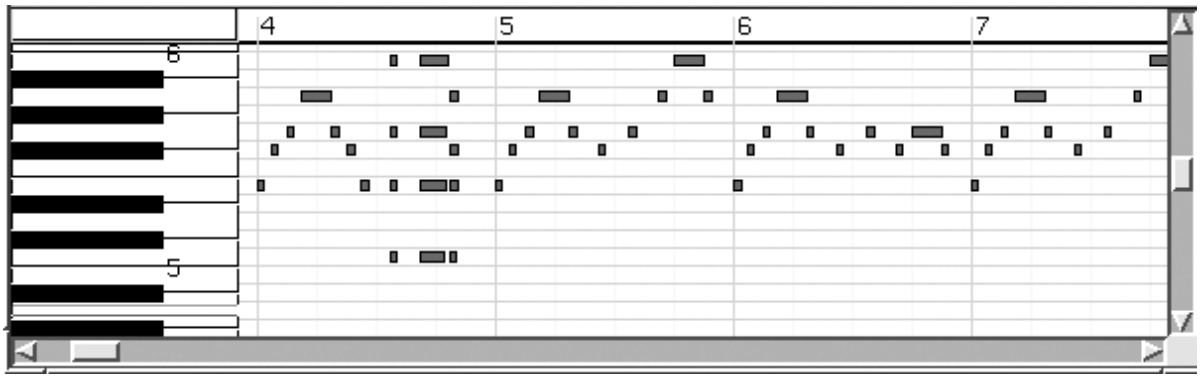
per quanto riguarda i collegamenti nello spine, la porzione di codice è la seguente

```
for (i=0; i<=DBCmidi.arTracceTotali.GetUpperBound(); i++)
      {
      DBCmidi.p_traccia=((DBClasseMidi::traccia*)(DBCmidi.arTracceTotali.G
etAt(i)));
      if (DBCmidi.p_traccia->indiceArTracce!=-1)
            {
            CMXPart *tmpPart;
            tmpPart=this->Logic.GetMLos()->GetPart(DBCmidi.p_traccia-
>strumento);

      DBCmidi.p_arEventi=((CObArray*)(DBCmidi.arTracce.GetAt(DBCmidi.p_tra
ccia->indiceArTracce)));
      vector <VociStrumento*>::iterator it=v_vociStrumento.begin();

                  while (it!=v_vociStrumento.end())
                  {
                        if ((*it)->strumento==DBCmidi.p_traccia->strumento)
```

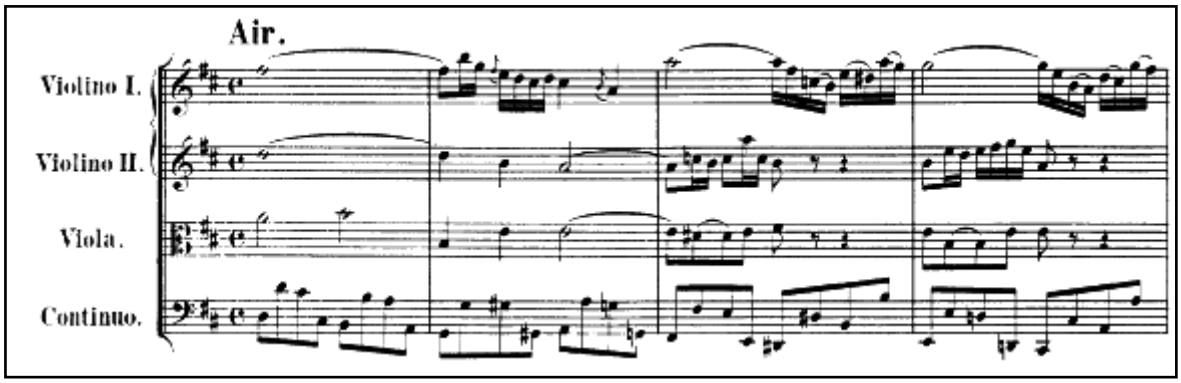
```
                        {
                              (*it)->numero_voci++;
                              break;
                        }
                        else
                              it++;
                  }

                  VociStrumento* currVoceStr=NULL;

                  if (it==v_vociStrumento.end()) //se è presente un'unica
voce monofonica
                  {
                        currVoceStr=new VociStrumento();
                        currVoceStr->numero_voci=1;
                        currVoceStr->strumento=DBCmidi.p_traccia-
>strumento;
                        v_vociStrumento.push_back(currVoceStr);
                  }
                  else //più voci per strumento
                        currVoceStr=(*it);
```

- **Interpretazione del continuum spazio temporale**

Tutti gli eventi trattati finora, devono avere una loro precisa collocazione spazio-temporale regolata dai dettami della teoria musicale. Che è molto diversa da quella data nel MIDI.
In MX, questo aspetto è trattato all'interno dello Spine.
Non solo la concorrenza fra gli eventi, ma anche la loro esatta collocazione nello spartito e le conseguenti indicazioni tipografiche devono essere previste (i valori di timing e hpos di MX).
Funzionalmente, per ogni canale un brano MIDI si presenta sotto questa forma




Vi è una *Timeline*, in è specificata la durate l'altezza di ogni nota e le relative concorrenze.
Il MIDI file ci trasferisce queste informazioni sequenzialmente un canale per volta.
Quindi, dal nostro punto di vista dovremmo tener traccia nel vettore Spine di ogni elemento creato, quindi ordinarlo in base in base alla scansione spazio-temporale (immaginiamo un pentagramma continuo senza accapi), ed infine mettere in concorrenza tutti gli strumenti con le rispettive voci. La figura mostra il "risultato finale" da ottenere.




In MIDI2MX questa operazione viene fatta creando un elemento event nel vettore spine ad ogni istanza di una classe che sia rilevante.
Una volta riempito Spine, bisogna ordinarlo. Prima di tutto teniamo conto di alcuni principi.
Ad esempio la chiave e la segnatura va messa sempre all'inizio di una voce.
Quindi, ordiniamo gli eventi grazie al tempo globale.
La funzione `SortAndCompileEvents` si preoccupa di ordinare lo spine.
Essa richiama al suo interno la funzione booleana `CompareFunction` la quale confronta due eventi a partire dal loro start_time dato dal MIDI.
Nel caso siano uguali, l'ordinamento verrà fatto in base al tipo di evento.

```
BOOL CompareFunction(CMXEvent* ev1,CMXEvent* ev2)
{
      if (ev1->start != ev2->start)
```

```
            return (ev1->start < ev2->start);
      else
      {
            return (ev1->GetID()<ev2->GetID());
      }
}
```

A questo punto, la funzione `SortAndCompileEvents` si preoccupa di fare l'ordinamento e di calcolare il timing, che ricordiamo essere in MX la differenza di tempo tra un evento e quello che lo precede.

```
void CMXSpine::SortAndCompileEvents()
{
      sort(this->m_Events.begin(),this->m_Events.end(),CompareFunction);

      //qui aggiungo i timings
      long curr_timing=0;
      for (vector <CMXEvent*>::iterator it=this-
>m_Events.begin();it!=this->m_Events.end();it++)
      {
            CMXEvent* evt=(*it);
            if (evt->timing==MX_UNDEFINED)
                  evt->SetValues(evt->GetID(),evt->start-curr_timing,evt-
>start-curr_timing);
            curr_timing=evt->start;
      }
}
```

Per quanto riguarda l'hpos, che ricordiamo in MX essere la distanza fisica orizzontale che separa due eventi, la prassi è quella di seguire gli stessi valori del timing. Dobbiamo tuttavia immaginare una diversa distanza per tutti quelli elementi che non hanno timing (vedi le indicazione di dinamica e di chiave, le alterazioni ecc.) ed inoltre, calcolare che la fine della battuta porta con sé l'aggiunta di una stanghetta, la quale influisce sulla distanza del primo evento della nuova battuta rispetto all'ultimo evento della battuta precedente.
Questa operazione viene eseguita dalla funzione `CorrectHpos`

```
void CMXLos::CorrectHPos(long distance)
{
      CMXEvent* minEvt=NULL;
```

```
	CMXEvent* tmpEvt=NULL;

	long max_m=0;
	//cerco il massimo numero di measure in tutti i part
	for (vector <CMXPart*>::iterator it=this->m_part.begin();it!=this->m_part.end();it++)
	{
		CMXPart* prt=(*it);
		max_m=max(max_m,prt->GetMaxMeasureNumber());
	}
//scorro tutti i numeri di measures, e individuo per ognuna il primo evento
	for(long m=1;m<=max_m;m++)
	{
		//devo farlo per tutti i part di questa measure...
		for (vector <CMXPart*>::iterator it=this->m_part.begin();it!=this->m_part.end();it++)
		{
			CMXPart* prt=(*it);
			//seleziono il primo evento in ordine temporale
			CMXMeasure *mx_measure=prt->GetMeasureByNumber(m);
			if (mx_measure==NULL)
				continue;
			tmpEvt=mx_measure->GetFirstEvent();
			if ((minEvt==NULL)||
				((minEvt!=NULL) && (minEvt->start>tmpEvt->start))
||
				((minEvt!=NULL) && (minEvt->start==tmpEvt->start)
&& (minEvt->GetID()>tmpEvt->GetID()))
				)
				minEvt=tmpEvt;
		}
//aggiungo la distanza per la partitura solo al primo evento di ogni measure (per silulare la stanghetta)
minEvt->SetValues(minEvt->GetID(),minEvt->timing,minEvt->hpos+distance);
		minEvt=NULL;
	}
}
```

- **Interpretazione del titolo e altri dati generali**

Il MIDI dispone di meta events, messaggi in cui è possibile inserire dati informativi sul brano in questione quali titolo, autore, data e informazioni sulla segnatura di chiave. Non sempre questi campi sono riempiti.
Per maggior correttezza, questi dati vengono lasciati gestire all'utente attraverso l'interfaccia grafica di MIDI2MX.
In ogni modo, l'applicazione crea un file di tipo MX chiamandolo con il nome originario del file MIDI più le opzioni parametriche scelte. Inoltre, se non specificato, il campo relativo al titolo del pezzo musicale sarà riempito con il nome del file in ingresso.

## *5.4 – Alcuni esempi*

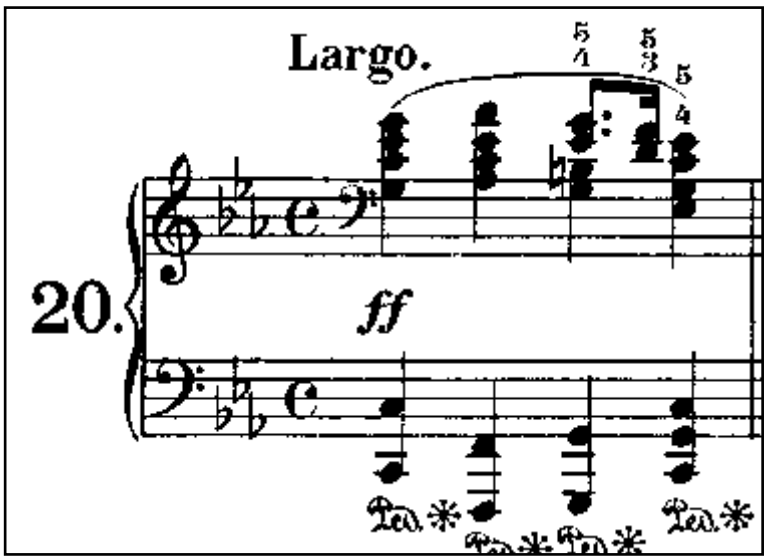
Presentiamo alcuni esempi di conversione per meglio comprendere pregi e difetti del modello interpretativo

### *5.4.1 – Chopin, Preludi N°20 e N° 2, Op. 28*

Sono due brani brevi che presentano alcune delle problematiche da affrontare.
Il preludio 20, una volta convertito in MX presenta questo spine

```
- <spine>
    <event id="evento__clef_0_Acoustic_Grand_Piano_____01_0" timing="NULL" lpos="0"/>
    <event id="evento__clef_1_Acoustic_Grand_Piano_____02_0" timing="NULL" lpos="0"/>
    <event id="evento__keysignature_0_Acoustic_Grand_Piano_____01_0" timing="NULL" lpos="10"/
    <event id="evento__keysignature_1_Acoustic_Grand_Piano_____02_0" timing="NULL" lpos="0"/>
    <event id="evento__timesignature_0_Acoustic_Grand_Piano_____01_0" timing="NULL" lpos="10"
    <event id="evento__timesignature_1_Acoustic_Grand_Piano_____02_0" timing="NULL" lpos="0"/
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000000" timing="0" lpos="10"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000001" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000002" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000003" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000000" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000001" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000004" timing="240" lpos="240"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000005" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000006" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000007" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000002" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000003" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000008" timing="240" lpos="240"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000009" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000010" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000011" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000004" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000005" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000012" timing="180" lpos="180"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000013" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000014" timing="60" lpos="60"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000015" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000016" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000017" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000006" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000007" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____02_000008" timing="0" lpos="0"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000018" timing="240" lpos="250"/>
    <event id="evento_nota_Acoustic_Grand_Piano_____01_000019" timing="0" lpos="0"/>
```

Se lo confrontiamo con la partitura originale

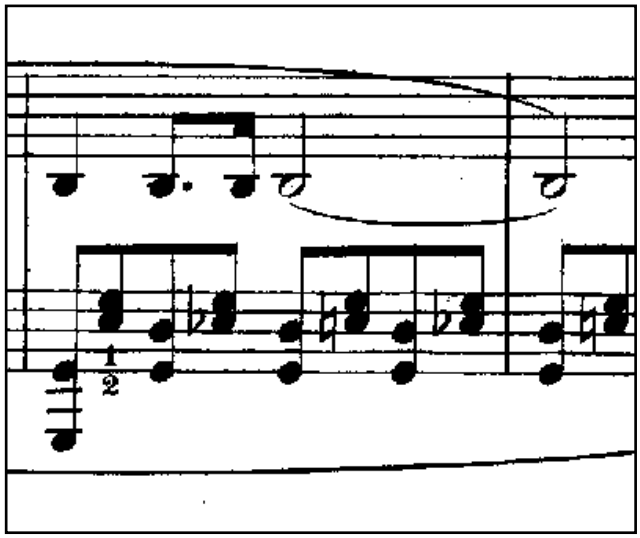



notiamo subito che il posizionamento delle chiavi e le segnature è corretto dal punto di vista del timing e hpos. E stesso dicasi per gli eventi nota. Infatti dato 240 come valore per la semiminima, i valori di 60 e 180 presenti rappresentano le durate esatte degli eventi 01_000012 e 01_000014. Si noti inoltre, la corretta aggiunta all'hpos dell'evento 01_000018.

Per quanto riguarda le altezze, selezionando la chiave di DO i valori sono anche qui corretti

```
- <measure number="1">
  - <voice ref="voce_Acoustic_Grand_Piano_____01">
    - <chord event_ref="evento_nota_Acoustic_Grand_Piano_____01_000000" duration="240">
      - <notehead>
          <pitch step="G" octave="4"/>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic_Grand_Piano_____01_000001" duration="240">
      - <notehead>
          <pitch step="C" octave="5"/>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic_Grand_Piano_____01_000002" duration="240">
      - <notehead>
          <pitch step="D" octave="5" alter="1"/>
        - <accidental>
            <sharp/>
          </accidental>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic_Grand_Piano_____01_000003" duration="240">
      - <notehead>
          <pitch step="G" octave="5"/>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic_Grand_Piano_____01_000004" duration="240">
      - <notehead>
          <pitch step="G" octave="4" alter="1"/>
        - <accidental>
            <sharp/>
          </accidental>
        </notehead>
      </chord>
```

Il preludio 2 ci serve per analizzare un altro aspetto. Le legature di valore.



Nel file MX risultante, nella battuta 6 troviamo la nota SI con la durata spezzata e l'indicazione della legatura. Nella battuta 7, il Si viene ripetuto sempre con durata 960. Il MIDI ci restituiva invece una singola nota di durata 1920.

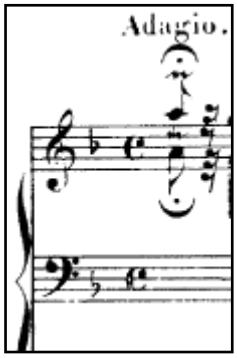
```
+ <measure number="5"></measure>
- <measure number="6">
  - <voice ref="voce_Acoustic Grand Piano _01">
    - <chord event_ref="evento_nota_Acoustic Grand Piano _01_000008" duration="480">
      - <notehead>
          <pitch step="B" octave="4"/>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic Grand Piano _01_000009" duration="360">
      - <notehead>
          <pitch step="B" octave="4"/>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic Grand Piano _01_000010" duration="120">
      - <notehead>
          <pitch step="B" octave="4"/>
        </notehead>
      </chord>
    - <chord event_ref="evento_nota_Acoustic Grand Piano _01_000011" duration="960">
      - <notehead>
          <pitch step="B" octave="4"/>
          <tie/>
```

```
- <measure number="7">
  - <voice ref="voce_Acoustic Grand Piano _01">
    - <chord event_ref="evento_nota_Acoustic Grand Piano _01_000012">
        <rest duration="960"/>
      </chord>
    - <chord event_ref="evento_nota_Acoustic Grand Piano _01_000285" duration="960">
      - <notehead>
          <pitch step="B" octave="4"/>
        </notehead>
      </chord>
    </voice>
```

In ogni caso, entrambi i file, convertono in maniera corretta i due pezzi di Chopin

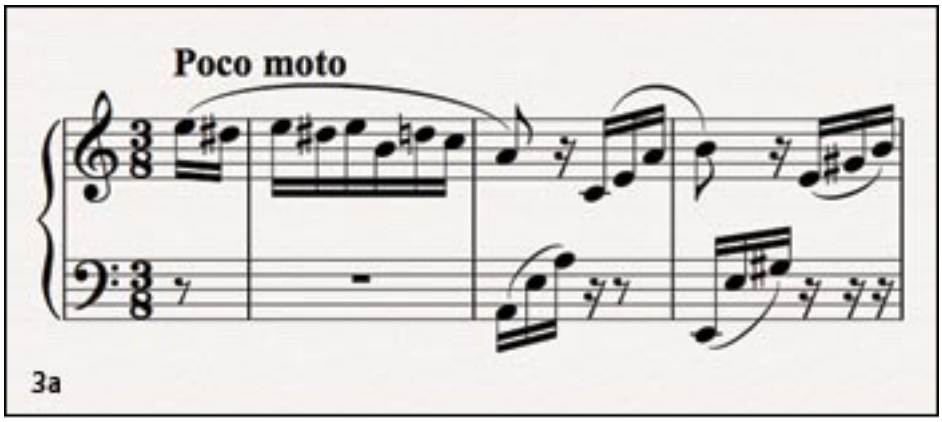
### *5.4.2 – Bach, Toccata e Fuga in Re Minore*
### *Beethoven, Per Elisa*

Esaminiamo brevemente due conversioni non esatte.
Nel pezzo di Bach, vengono correttamente individuate le 4 voci dell'organo, ma a causa dell'eccessiva rapidità dell'attacco delle prime tre note, queste vengono rappresentate come simultanee.

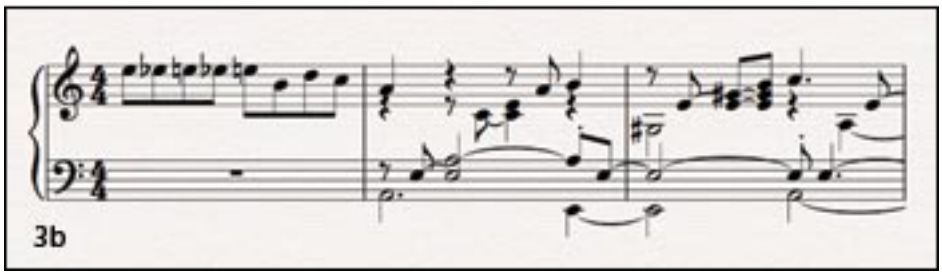



Questo errore non è correggibile neanche aumentando la granularità della quantizzazione dall'interfaccia. E' un esempio del limite del MIDI.

Invece, nell'altro celeberrimo brano di Beethoven, la conversione presenta svariate differenze nella collocazione delle note e dei tempi.
Abbiamo provato a disegnare la partitura dell'MX risultante per meglio chiarire le differenze.




3a



3b

## *5.5 – Conclusioni*

Scopo di questo lavoro era aggiungere un mattoncino al lavoro della ricerca sulla trascrizione automatica delle partiture.
Avere un sistema [27] che in tutto e per tutto si presenti come una black box, con un segnale analogico in ingresso e un file grafico in uscita è un obiettivo lontano ma non impossibile.
Nel nostro piccolo poter ottenere da subito un certo risultato pratico e non solo dimostrarne la fattibilità era uno dei target.
L'applicazione MIDI2MX nasce con questo scopo.
Avere già un'ossatura di un file MX in automatico, è un altro obiettivo raggiunto. Fino ad ora descrivere un brano musicale in XML era un verboso esercizio accademico.
Sviluppi e miglioramenti possibili sono evidenti. Prima di tutto, è possibile aggiungere o modificare il parser. E magari cercare informazioni anche da tipi di file diversi dal MIDI.
Poi, l'aggiunta di modelli per tutti quegli aspetti non trattati in questa sede. Tonalità, cambi di segnatura, abbellimenti ecc.
Ed infine, un risultato grafico fruibile per la massa. E' un punto non trascurabile, ma anche non eccessivamente impegnativo se pensiamo da un lato alle potenzialità dell'XML e dall'altro alla struttura dati creata. Nulla ci vieta infatti di elaborare una funzione di disegno all'interno di ogni classe. Scrivere una riga di testo o disegnare una riga non sono poi operazioni così diverse.

Appendice

# Messaggi MIDI

| STATUS BYTE | | | | DATA BYTES | |
|---|---|---|---|---|---|
| 1st Byte Value | | | Function | 2nd Byte | 3rd Byte |
| Binary | Hex | Dec | | | |
| 10000000 | 80 | 128 | Chan 1 Note off | Note Number | Note Velocity |
| 10000001 | 81 | 129 | Chan 2 " | (0-127) | (0-127) |
| 10000010 | 82 | 130 | Chan 3 " | see | " |
| 10000011 | 83 | 131 | Chan 4 " | Table | " |
| 10000100 | 84 | 132 | Chan 5 " | 2 | " |
| 10000101 | 85 | 133 | Chan 6 " | " | " |
| 10000110 | 86 | 134 | Chan 7 " | " | " |
| 10000111 | 87 | 135 | Chan 8 " | " | " |
| 10001000 | 88 | 136 | Chan 9 " | " | " |
| 10001001 | 89 | 137 | Chan 10 " | " | " |
| 10001010 | 8A | 138 | Chan 11 " | " | " |
| 10001011 | 8B | 139 | Chan 12 " | " | " |
| 10001100 | 8C | 140 | Chan 13 " | " | " |
| 10001101 | 8D | 141 | Chan 14 " | " | " |
| 10001110 | 8E | 142 | Chan 15 " | " | " |
| 10001111 | 8F | 143 | Chan 16 " | " | " |
| 10010000 | 90 | 144 | Chan 1 Note on | " | " |
| 10010001 | 91 | 145 | Chan 2 " | " | " |
| 10010010 | 92 | 146 | Chan 3 " | " | " |
| 10010011 | 93 | 147 | Chan 4 " | " | " |
| 10010100 | 94 | 148 | Chan 5 " | " | " |
| 10010101 | 95 | 149 | Chan 6 " | " | " |
| 10010110 | 96 | 150 | Chan 7 " | " | " |
| 10010111 | 97 | 151 | Chan 8 " | " | " |
| 10011000 | 98 | 152 | Chan 9 " | " | " |
| 10011001 | 99 | 153 | Chan 10 " | " | " |
| 10011010 | 9A | 154 | Chan 11 " | " | " |
| 10011011 | 9B | 155 | Chan 12 " | " | " |
| 10011100 | 9C | 156 | Chan 13 " | " | " |
| 10011101 | 9D | 157 | Chan 14 " | " | " |
| 10011110 | 9E | 158 | Chan 15 " | " | " |
| 10011111 | 9F | 159 | Chan 16 " | " | " |
| 10100000 | A0 | 160 | Chan 1 Polyphonic | " | Aftertouch |
| 10100001 | A1 | 161 | Chan 2 aftertouch | " | amount |
| 10100010 | A2 | 162 | Chan 3 " | " | (0-127) |
| 10100011 | A3 | 163 | Chan 4 " | " | " |
| 10100100 | A4 | 164 | Chan 5 " | " | " |
| 10100101 | A5 | 165 | Chan 6 " | " | " |
| 10100110 | A6 | 166 | Chan 7 " | " | " |
| 10100111 | A7 | 167 | Chan 8 " | " | " |
| 10101000 | A8 | 168 | Chan 9 " | " | " |
| 10101001 | A9 | 169 | Chan 10 " | " | " |
| 10101010 | AA | 170 | Chan 11 " | " | " |
| 10101011 | AB | 171 | Chan 12 " | " | " |
| 10101100 | AC | 172 | Chan 13 " | " | " |
| 10101101 | AD | 173 | Chan 14 " | " | " |
| 10101110 | AE | 174 | Chan 15 " | " | " |
| 10101111 | AF | 175 | Chan 16 " | " | " |

| | | | | | |
|---|---|---|---|---|---|
| 10110000 | B0 | 176 | Chan 1 Control/ | See | See |
| 10110001 | B1 | 177 | Chan 2 Mode change | Table | Table |
| 10110010 | B2 | 178 | Chan 3 " | three | three |
| 10110011 | B3 | 179 | Chan 4 " | " | " |
| 10110100 | B4 | 180 | Chan 5 " | " | " |
| 10110101 | B5 | 181 | Chan 6 " | " | " |
| 10110110 | B6 | 182 | Chan 7 " | " | " |
| 10110111 | B7 | 183 | Chan 8 " | " | " |
| 10111000 | B8 | 184 | Chan 9 " | " | " |
| 10111001 | B9 | 185 | Chan 10 " | " | " |
| 10111010 | BA | 186 | Chan 11 " | " | " |
| 10111011 | BB | 187 | Chan 12 " | " | " |
| 10111100 | BC | 188 | Chan 13 " | " | " |
| 10111101 | BD | 189 | Chan 14 " | " | " |
| 10111110 | BE | 190 | Chan 15 " | " | " |
| 10111111 | BF | 191 | Chan 16 " | " | " |
| 11000000 | C0 | 192 | Chan 1 Program | Program # | NONE |
| 11000001 | C1 | 193 | Chan 2 change | (0-127) | " |
| 11000010 | C2 | 194 | Chan 3 " | " | " |
| 11000011 | C3 | 195 | Chan 4 " | " | " |
| 11000100 | C4 | 196 | Chan 5 " | " | " |
| 11000101 | C5 | 197 | Chan 6 " | " | " |
| 11000110 | C6 | 198 | Chan 7 " | " | " |
| 11000111 | C7 | 199 | Chan 8 " | " | " |
| 11001000 | C8 | 200 | Chan 9 " | " | " |
| 11001001 | C9 | 201 | Chan 10 " | " | " |
| 11001010 | CA | 202 | Chan 11 " | " | " |
| 11001011 | CB | 203 | Chan 12 " | " | " |
| 11001100 | CC | 204 | Chan 13 " | " | " |
| 11001101 | CD | 205 | Chan 14 " | " | " |
| 11001110 | CE | 206 | Chan 15 " | " | " |
| 11001111 | CF | 207 | Chan 16 " | " | " |
| 11010000 | D0 | 208 | Chan 1 Channel | Aftertouch | " |
| 11010001 | D1 | 209 | Chan 2 aftertouch | amount | " |
| 11010010 | D2 | 210 | Chan 3 " | (0-127) | " |
| 11010011 | D3 | 211 | Chan 4 " | " | " |
| 11010100 | D4 | 212 | Chan 5 " | " | " |
| 11010101 | D5 | 213 | Chan 6 " | " | " |
| 11010110 | D6 | 214 | Chan 7 " | " | " |
| 11010111 | D7 | 215 | Chan 8 " | " | " |
| 11011000 | D8 | 216 | Chan 9 " | " | " |
| 11011001 | D9 | 217 | Chan 10 " | " | " |
| 11011010 | DA | 218 | Chan 11 " | " | " |
| 11011011 | DB | 219 | Chan 12 " | " | " |
| 11011100 | DC | 220 | Chan 13 " | " | " |
| 11011101 | DD | 221 | Chan 14 " | " | " |
| 11011110 | DE | 222 | Chan 15 " | " | " |
| 11011111 | DF | 223 | Chan 16 " | " | " |
| 11100000 | E0 | 224 | Chan 1 Pitch | Pitch | Pitch |
| 11100001 | E1 | 225 | Chan 2 wheel | wheel | wheel |
| 11100010 | E2 | 226 | Chan 3 control | LSB | MSB |
| 11100011 | E3 | 227 | Chan 4 " | (0-127) | (0-127) |
| 11100100 | E4 | 228 | Chan 5 " | " | " |
| 11100101 | E5 | 229 | Chan 6 " | " | " |
| 11100110 | E6 | 230 | Chan 7 " | " | " |
| 11100111 | E7 | 231 | Chan 8 " | " | " |
| 11101000 | E8 | 232 | Chan 9 " | " | " |
| 11101001 | E9 | 233 | Chan 10 " | " | " |
| 11101010 | EA | 234 | Chan 11 " | " | " |
| 11101011 | EB | 235 | Chan 12 " | " | " |
| 11101100 | EC | 236 | Chan 13 " | " | " |
| 11101101 | ED | 237 | Chan 14 " | " | " |
| 11101110 | EE | 238 | Chan 15 " | " | " |
| 11101111 | EF | 239 | Chan 16 " | " | " |

| | | | | | |
|---|---|---|---|---|---|
| 11110000 | F0 | 240 | System Exclusive | ** | ** |
| 11110001 | F1 | 241 | MIDI Time Code Qtr. Frame | -see spec- | -see spec- |
| 11110010 | F2 | 242 | Song Position Pointer | LSB | MSB |
| 11110011 | F3 | 243 | Song Select (Song #) | (0-127) | NONE |
| 11110100 | F4 | 244 | Undefined | ? | ? |
| 11110101 | F5 | 245 | Undefined | ? | ? |
| 11110110 | F6 | 246 | Tune request | NONE | NONE |
| 11110111 | F7 | 247 | End of SysEx (EOX) | " | " |
| 11111000 | F8 | 248 | Timing clock | " | " |
| 11111001 | F9 | 249 | Undefined | " | " |
| 11111010 | FA | 250 | Start | " | " |
| 11111011 | FB | 251 | Continue | " | " |
| 11111100 | FC | 252 | Stop | " | " |
| 11111101 | FD | 253 | Undefined | " | " |
| 11111110 | FE | 254 | Active Sensing | " | " |
| 11111111 | FF | 255 | System Reset | " | " |

** **Note**: System Exclusive (data dump) 2nd byte= Vendor ID followed by more data bytes and ending with EOX.

# Bibliografia e riferimenti

## Cap. 1

**[1] Vita di Mozart** – *Stendhal,* Ed. Passigli

**[2] Signal Processing Methods for the Automatic Transcription of Music -** *Anssi Klapuri,* PhD Thesis Tampere University of Technology 2004, Publications 460

## Cap. 2

**[3] L'acustica per il musicista** – *P.Righini*, Ed. Ricordi

**[4] La musica nella storia** - *Piero Mioli*, Bologna 1989

**[5] STORIA DELLA MUSICA MEDIOEVALE E RINASCIMENTALE** – corso del prof. Davide Da Olmi

**[6] Conversione automatica di processi audio analogici in processi MIDI –** *Giuseppe Frazzini*, Tesi di Laurea in Scienze dell'Informazione 1995

**[7] Interpretare ipertesti** – *Daniele Barbieri* in "Il gioco: segni e strategie", a cura di Alessandro Perissinotto, Torino, Scriptorium 1997.

**[8] Conoscere la Musica** – *S. Castelvecchi e E. Stazi*, Editori Riuniti 1988

**[9] Elementi Fondamentali della Teoria della Musica** – *Flora Gallo*, Ed. Simeoli Napoli

## Cap. 3

**[10] Programmare con XML –** *Paolo Pialorsi*, Ed. Mondatori

**[11] Manuale MX** – *Luca Andrea Ludovico*, Università degli Studi di Milano 2005

**[12] Musedata, An Electronic Library of Classical Music Scores** – *AAVV*, Center for Computer Assisted Research in the Humanities (http://www.musedata.org/formats/musedata/)

**[13] MusicXML Definition -** *Michael Good*, from *The Virtual Score: Representation, Retrieval, Restoration*, Walter B. Hewlett and Eleanor Selfridge-Field, eds., MIT Press, Cambridge, MA, 2001, pp. 113-124. (www.musicxml.org/xml.html)

**[14] The Music Encoding Initiative (MEI) -** University of Virginia (http://www.lib.virginia.edu/digital/resndev/mei/)

**[15] MAX 2002, Musical Application using XML** - AAVV, LIM 2002

**[16] XML for Music** – article of *Darin Stewart from "Electronic Musician"*, Oregon Health and Science University 2003

## Cap. 4

**[17] Dispense MIDI** – *Maurizo Longari*, LIM 1999

**[18] MIDI Documentation** - *Hendrik Jan Veenstra*, Omega Art, (http://www.omega-art.com)

**[19] Complete MIDI 1.0 Detailed Specification v96.1** - The International MIDI Association, 2001

**[20] Identificazione automatica di strutture musicali** - *Luca Andea Ludovico*, Tesi di Laurea in Ingegneria, Politecnico di Milano 2003

## Cap. 5

**[21] STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library -** *David R. Musser, Atul Saini*, Addison-Wesley, 1996

**[22] MSDN Library Standard C++ Libraries**, Microsoft (http://msdn2.microsoft.com/en-us/library/ms310245.aspx)

**[23] Signal-to-Score Music Transcription using Graphical Models -** *Emir Kapanci and Avi Pfeffer,* Harvard University Division of Engineering and Applied Sciences 2005

**[24] Automatic Synchronization of Music Data in Score-, MIDI- and PCM-Format -** *Vlora Arifi, Michael Clausen, Frank Kurth, and Meinard Muller,*Universitat Bonn, Institut fur Informatik III, ISMIR 2003 paper.

**[25] Techniques For Automatic Music Transcription -** *Juan Pablo Bello, Giuliano Monti and Mark Sandler* - Department of Electronic Engineering, King's College London, ISMIR 2000 Paper

**[26] Bayesian Analysis of Western Tonal Music** - *Manuel Davy, Simon J. Godsill and Jerome Idier*, Journal of the Acoustical Society of America (JASA), Volume 119, Number 4, pp. 2498-2517, April 2006.

**[27] Elaborazione Numerica dei Segnali** - *Alberto Bertoni, Giuliano Grossi*, DSI 1999

## Ringraziamenti

Ringrazio il professor Goffredo Haus e l'ingegner Luca Andrea Ludovico per la disponibilità concessa e il supporto didattico apportato durante lo svolgimento di questo lavoro di tesi.
Ringrazio altresì Carlo Russo e tutti i colleghi universitari che mi hanno supportato in questo *lungo* percorso accademico.
Il ringraziamento finale va alla mia famiglia, i miei genitori e mia sorella, per avermi concesso questa possibilità e per la loro pazienza.