UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Dipartimento di Informatica e Comunicazione
Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale
Cl.26

PROTOTIPO PER LA RAPPRESENTAZIONE GEOMETRICA DI ACCORDI IN DOCUMENTI IEEE 1599

Relatore: Prof. Luca Andrea Ludovico

Correlatore: Dott. Adriano Baratè

Tesi di:
Lorenzo Caccia
Matricola: 738776

Anno Accademico 2011/2012

Indice

Introduzione

Questo progetto ha lo scopo di sviluppare un prototipo che fornisca una rappresentazione grafica bidimensionale degli accordi musicali.
L'input dell'algoritmo è rappresentato da un brano codificato in IEEE 1599, il formato standard di codifica della partitura.
Il programma elaborato, in fase di lettura, analizza e preleva i dati che rappresentano ogni singola nota o pausa, ovvero: durata, alterazione e strumento che la sta eseguendo. Inoltre, fornisce la possibilità, attraverso l'interfaccia grafica, di selezionare un singolo strumento.
Dopo aver effettuato il parsing del file originale e dopo aver eseguito un algoritmo di verticalizzazione, viene prodotto un output grafico che fornisce la rappresentazione grafica bidimensionale prescelta.
Il prototipo è stato sviluppato in C# per Mono, in modo da poter funzionare in ambiente multipiattaforma, sia in ambiente Windows o Mac.

Capitolo 1

Di seguito vengono introdotti i fondamenti di teoria musica e armonia. [4]

1.1 FONDAMENTI DI TEORIA MUSICALE

La musica può essere definita la giustapposizione di due elementi fondamentali: altezza e durata, ovvero melodia e ritmo. L'unità minima di organizzazione musicale è la nota: un suono che possiede una certa altezza e una specifica durata .
L'altezza è la frequenza fondamentale di una nota musicale o suono che viene percepita, ed è una delle caratteristiche principali di un suono. L'altezza indica se un suono è acuto piuttosto che grave e dipende dalla frequenza dell'onda sonora che lo genera. In particolare: più la frequenza di un'onda sonora è elevata e più il suono è acuto, mentre più è bassa la frequenza e più il suono è grave.
La durata è determinata dal periodo di tempo in cui l'oggetto sonoro emette vibrazioni.
Nel linguaggio musicale la durata e l'altezza delle note viene rappresentata attraverso la partitura e le informazioni contenuta in essa.

L'intervallo è la distanza fra due suoni o note, intesa come rapporto. Un intervallo può essere melodico se le note sono suonate in successione, armonico se suonate simultaneamente
Consideriamo la nota *Do* e supponiamo che abbia una frequenza *f*. Se raddoppiamo la frequenza a 2*f* otteniamo una nota che ha la massima consonanza con la prima. Queste due note vengono chiamate con lo stesso nome e l'intervallo che fra esse intercorre è denominato ottava.
Per ottenere le suddivisioni dell'ottava, nella musica occidentale, viene utilizzato il cosiddetto sistema temperato. Questo sistema consiste nel suddividere l'ottava in 12 intervalli
uguali. Ognuno di questi intervalli si chiama semitono ed è anche l'intervallo più piccolo.
Il numero totale delle note che abbiamo a disposizione è perciò 12. Anche se è possibile ricavarne altre 12 per ogni ottava successiva o precedente, sono sempre da considerasi le stesse 12 note trasportate su ottave diverse.

Do	Do# Re*b*	Re	Re# Mi*b*	Mi	Fa	Ra# Sol*b*	Sol	Sol# La*b*	La	La# Si*b*	Si

Questa è quella che conosciamo anche con il nome di scala cromatica.

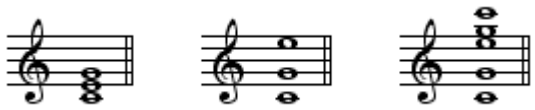
Attualmente la normale dicitura che determina il nome delle note viene vista su doppia terminologia. Da un lato la notazione tradizionale, quella che solitamente viene definita come europea (o latina), dall'altro la notazione cosiddetta anglosassone nel quale le note vengono tradotte utilizzando delle singole lettere in maiuscolo, secondo lo schema che potete vedere qui sotto.

Not. Tradizionale	Do	Re	Mi	Fa	Sol	La	Si
Not. Anglosassone	C	D	E	F	G	A	B

Nel presente progetto verrà utilizzata per l'ottava musicale la notazione anglosassone proprio perché quest'ultima viene sostanzialmente usata in quasi tutti i paesi, diventando presto uno standard.

1.2 FONDAMENTI DI ARMONIA

L'armonia è il ramo della teoria musicale che studia gli accordi. Nel lessico proprio della teoria musicale occidentale, si definisce accordo la combinazione di due o più intervalli armonici ordinati per intervalli di terza, e non solo, ove per combinazione (detta anche sovrapposizione) di due intervalli armonici si intende la loro simultaneità ed il fatto di avere in comune una nota fondamentale. In questo senso si può dire che un accordo è formato da una serie di intervalli a partire dalla fondamentale. Scelta la nota fondamentale, ad essa si aggiungono le note che si trovano spostandosi da essa di un certo intervallo.
I suoni che costituiscono un accordo vengono contati una sola volta a prescindere dall'ottava a cui appartengono; ad esempio, i tre accordi mostrati qui di seguito sono accordi di tre suoni essendo composti, a varie altezze, dalle note *do*, *mi* e *sol*:



Nel primo e nel secondo caso l'unica differenza consiste infatti nell'ottava di appartenenza del suono *mi*. Si dice che i due accordi in questione hanno la stessa composizione, ma differente disposizione. Nel terzo caso invece i suoni *sol* e *do* sono raddoppiati, sono cioè presenti simultaneamente in due differenti ottave.
I tre accordi si presentano inoltre in tre differenti posizioni, in quanto il suono più acuto è differente nei tre casi. Avendo al contrario tutti e tre la stessa nota come suono più grave, essi si presentano nel medesimo stato (in questo caso, lo stato fondamentale).

Gli accordi basilari dell'armonia tonale si ottengono mediante la combinazione di due o più intervalli armonici di 3ª (maggiore, cioè costituito da due toni o minore, costituito da un tono e un semitono), a partire da uno dei gradi di una tonalità prefissata che viene detto “suono fondamentale” o semplicemente “fondamentale” dell'accordo.
A seconda del numero di note dei quali sono formati, nell'armonia tonale si individuano i seguenti gruppi di accordi basilari:

- Triadi (3 voci) ovvero la composizione di due intervalli armonici di terza.
- Accordi di settima (4 voci) costituiti dalla sovrapposizione di quattro suoni, ciascuno distante una terza dal precedente.
- Accordi di nona (5 voci) costituiti aggiungendo un intervallo di nona ad un accordo di settima.
- Accordi di undicesima (6 voci) costituiti aggiungendo un intervanno di undicesima ad un accordo di nona.
- Accordi di tredicesima (7 voci) sono accordi dissonanti che si generano dalla sovrapposizione alla fondamentale di altre sei note, ciascuna ad un intervallo di terza dalla precedente.

Capitolo 2

Dopo aver introdotto i fondamenti di teoria musicale, verranno presentate le caratteristiche del file XML e del linguaggio che ne sta alla base. Esso, entrando in relazione con il software progettato, permetterà la visualizzazione grafica della partitura che rappresenta. [1]

2.1 XML (*eXtensible Markup Language*)

Per linguaggio di marcatura (*mark-up language*) si intende una codifica che fa uso di un insieme di marcatori (*mark-up*) convenzionali, ossia aderenti a regole chiare, definite e comunemente accettate.
La struttura di un particolare tipo di file XML avviene tramite la cosiddetta definizione del tipo di documento (DTD, *document type definition*).
Il DTD fornisce un significato standard per descrivere in modo dichiarativo la struttura di un tipo di documento. Ciò significa descrivere:

- quali (sub-)elementi costituiscono un elemento
- se quest'ultimo possa contenere del testo
- quali attributi appartengano all'elemento
- tipizzazione e defaultizzazione degli attributi.

Un DTD é composto da 2 parti:

1. *Element Type Definition* che specifica la struttura del documento, i contenuti consentiti (content model) e gli attributi consentiti (dal significato delle dichiarazioni delle liste di attributi).
2. *Attribute List Declaration* che è la lista degli attributi permessi per ogni elemento.Ogni attributo è specificato da: name, type, e altre informazioni.

2.1.1 *Elementi*

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.

2.1.2 *Attributi*

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.
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*.

2.2 IL FORMATO IEEE 1599

Si analizza, ora, il formato IEEE 1599, DTD specifico dei file XML usati quali input del software progettato.

Un brano musicale può contenere elementi comunicativi che possono risultare assai più complessi di quanto emerga da un'analisi superficiale.

I principali aspetti su cui si sofferma un musicista nel descrivere una partitura, sono tutti i simboli del pentagramma, aventi significati diversi, e tutti indispensabili alla perfetta esecuzione del brano. Il fruitore, invece è concentrato sul segnale del suono audio e quindi sull'ascolto.

Viceversa, la descrizione di maggior interesse per un fruitore è legata al segnale audio e di conseguenza all'ascolto.

Ulteriore elemento fondamentale, è la necessità di conservazione dell'esecuzione musicale, degli aspetti iconografici ad essi legati e alla loro possibilità di catalogazione e preservazione digitale.

Ciascuno di questi aspetti ha una controparte in ambito informatico in uno o più formati. Il risultato, dal punto di vista della rappresentazione musicale, è la disponibilità di un vasto insieme di informazioni che spesso non sono correlate tra loro, codificate in formati eterogenei e dunque difficilmente fruibili.

Un'efficace risposta alla problematica di descrivere la musica nel suo complesso, all'interno di un unico documento, è data dal formato IEEE 1599-2008. Si tratta di uno standard internazionale recentemente approvato dall'Institute of Electrical and Electronics Engineers (IEEE) il cui obiettivo è fornire una descrizione in XML di singoli brani musicali, prendendo in considerazione tutte le forme di descrizione collegabili al brano stesso. Tale risultato viene conseguito organizzando i dati e i metadati collegati in sei livelli (layers).

Ogni livello rappresenta un grado di astrazione diverso per l'informazione musicale:

- General, che contiene i metadati relativi al brano in oggetto, tra cui le informazioni catalografiche su titolo dell'opera, autori e genere;
- Logic, vero nucleo del formato, destinato alla descrizione simbolica dei contenuti musicali;
    - Spine
    - Los (logically organized symbols)
    - Layout
- Structural, che identifica gli oggetti musicali su cui il brano è costruito e permette di evidenziarne i mutui rapporti;
- Notational, che contiene le differenti rappresentazioni grafiche della partitura, ad esempio riferibili a diverse edizioni o trascrizioni;

- Performance, che è dedicato ai formati per la generazione di esecuzioni sintetiche da parte dell'elaboratore;
- Audio, che consente di legare al brano in oggetto le esecuzioni audio/video della partitura.

Nel formato IEEE 1599-2008, tali livelli corrispondono tecnicamente a sei sotto-elementi dell'elemento radice, ciascuno avente stesso importanza, come mostrato in Fig. 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "http://www.mx.dico.
unimi.it/ieee1599.dtd">
<ieee1599>
   <general>...</general>
   <logic>...</logic>
   <structural>...</structural>
   <notational>...</notational>
   <performance>...</performance>
   <audio>...</audio>
</ieee1599>
```

Nel creare il formato, si è tenuto conto della possibilità di descrivere lo stesso brano musicale con molteplicità di materiali di stessa natura. Ad esempio, all'interno del livello Notational è possibile codificare l'informazione relativa a partiture di differenti versioni, manoscritte e a stampa, dello stesso brano; analogamente, nel livello Audio è possibile inserire più esecuzioni del brano in oggetto avvenute in momenti e luoghi diversi. Un documento IEEE 1599-2008 acquisterà più valore quanto più esso risulterà ricco di contenuti. E tale ricchezza può esprimersi tanto nella compilazione di una molteplicità di livelli, quanto nell'inserimento di una molteplicità di oggetti digitali in ciascun livello.
Un altro aspetto di fondamentale importanza nell'approccio descritto è il pieno supporto ai formati già comunemente in uso per i contenuti multimediali. Infatti in XML vengono esplicitamente descritti solo i metadati e le informazioni simboliche sulla partitura, mentre per l'audio, il video e la grafica si codificano gli opportuni riferimenti a file esterni. In tal modo si beneficia del corpus di oggetti digitali già disponibile (ad esempio, dei molti file MP3 relativi a una data canzone) e delle peculiarità di ciascun formato nel fornire una descrizione opportuna per i contenuti multimediali (ad esempio, delle caratteristiche del formato JPEG o TIFF per la grafica).

2.3 IL LAYER LOGIC

La parte fondamentale del lavoro si è concentrata sul Layer Logic che ora viene analizzato.

Il livello Logic rappresenta il nucleo fondamentale del formato ed è costituito da tre sottolivelli (elementi figli):

1. Spine che contiene la funzione di mappatura spazio-temporale
2. Los che descrive gli oggetti in partitura
3. Layout che si occupa di rappresentare il layout del brano

Spine e Los sono fondamentali per un file XML: 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 sub-layer **Spine** è fondamentale data la natura multi-livello dell'IEEE 1599: esso è il collante tra i vari livelli di rappresentazione e la codifica stessa non avrebbe significato.
Il suo obiettivo è elencare tutti gli eventi musicali contenuti all'interno del brano, cioè gli eventi rilevanti per chi effettua la codifica. Lo Spine, costituito da un elenco di eventi, permette di ordinare gli eventi stessi e di etichettarli in modo univoco. Ogni altro livello conterrà descrizioni o riferimenti agli eventi basandosi sull'identificativo univoco dello Spine. Per quanto concerne i dati multimediali contenuti nei file esterni, il formato IEEE 1599 attua una specifica mappatura degli eventi dello Spine (attraverso l'etichettatura univoca) su coordinate spaziali (esempio partiture e manoscritti), temporali (esempio audio, video) riferite al formato adottato.
Ad esempio, si consideri una nota identificata nella struttura dati dall'etichetta eventoN; per collocarla nelle diverse tracce presenti nel livello Audio sarà sufficiente esprimere in XML la relazione tra eventoN e il millisecondo o il frame relativo all'audio. Allo stesso modo, per indicare la nota suonata in un certo istante, si potrà legare nel sottolivello Notational l'etichetta eventoN alle coordinate dei punti che ne delimitano l'area occupata dei file grafici relativi.
Il meccanismo che demanda a file esterni la descrizione multimediale degli eventi e che si basa su riferimenti allo Spine è molto efficace ed efficiente, in quanto consente di aggiungere un certo numero di nuovi documenti, mappando di nuovo gli eventi senza re-identificarli. Cioè, se si aggiunge un'altra traccia audio o le scansioni di un'ulteriore partitura, ciò non comporta

la creazione e l'identificazione di ulteriori eventi, ma solo la determinazione dei punti o degli istanti in cui occorrono gli eventi musicali già listati nello Spine all'interno del nuovo file.

Nel sub-layer **Los** invece 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 Los, che sta appunto per Logical Organized Symbols che sarà analizzato nel prossimo paragrafo.

2.4 IL SUBLAYER LOS

L'informazione musicale simbolica, la partitura, trova posto nell'XML all'interno dell'elemento Los (logical organized symbols). Si riporta di seguito, l'intera gerarchia del formato IEEE 1599, mettendo in evidenza il punto di inserimento delle informazioni di partitura.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "">
<ieee1599>
        <general>
        …
        </general>
        <logic>
                <spine>
                …
                </spine>
                <los>
                        PUNTO DI INSERIMENTO DELLA PARTITURA
                </los>
        </logic>
        …
</ieee1599>
```

L'XML è capace di rappresentare dati strutturati, anzi la sua natura richiede la strutturazione dell'informazione. Quindi è necessario stabilire una gerarchia delle informazioni nella partitura. Il livello più alto è costituito dall'intera partitura, mentre quello più basso, il più elementare, è rappresentato da simboli quali note e pause. Come si strutturano, quindi, le informazioni?

Il formato IEEE 1599 risponde nel seguente modo:

1. una partitura è costituita da più accollature
2. un'accollatura è formata da più pentagrammi
3. un pentagramma contiene una o 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.

Ecco alcune osservazioni su quanto detto.

- Per quanto riguarda la gerarchia pentagramma > parte > voce possiamo fare esempi chiarificatori: Una semplice situazione è quella in cui ciascun pentagramma contiene esattamente una parte e la parte si compone di un'unica voce. E' il caso in cui la linea vocale del solista è associata a un ben preciso pentagramma ed è composta solo da esso. In casi più complessi, nel pentagramma possono esserci più voci, tutte riconducibili ad un'unica parte: la parte del pianoforte può contemplare tre o più voci.
- 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. Non sono quindi visibili ad occhio nudo le figure musicali (note e pause) suonate da tutti le parti della partitura, al contrario di quanto avviene in una partitura stampata o manoscritta. Ma ciò non è rilevante poiché le finalità dell'IEEE 1599 sono di codifica dell'informazione e non di presentazione. Sarà poi un eventuale operatore che attraverso uno specifico software potrà a ri-trasformare l'informazione in modo da renderla fruibile al musicista.
- Per accordo si intende la sovrapposizione di note all'interno di una parte e di una voce specifica. Quindi in IEEE 1599, è un accordo un insieme di note suonate contemporaneamente dalla stessa voce, e non quello che scaturisce dall'esecuzione contemporanea di linee melodiche differenti di parti diverse.
- La singola nota, cioè quella che non si sovrappone verticalmente ad alcun altro suono appartenente alla voce stessa, viene vista e codificata dal IEEE 1599 come accordo (degenere).

Quanto detto qui sopra origina nel IEEE 1599 il seguente scheletro:

```
<los>
        <!-- descrizione dell'accollatura -->
        <staff_list>
                <!-- descrizione delle proprietà dei pentagrammi -->
                <staff id="staff_1"> … </staff>
                <staff id="staff_2"> … </staff>
        </staff_list>
        <!-- descrizione delle singole parti -->
        <part id="flauto">
                <!--elenco delle voci della parte -->
                <voice_list>
                        <voice_item id="voce_flauto" staff_ref="staff_1"/>
                </voice_list>
                <measure number="1"> … </measure>
                <measure number="2">    … </measure>
                …
        </part>
        <part id="pianoforte">
                <voice_list>
                        <voice_item id="voce1" staff_ref="staff_2"/>
                        <voice_item id="voce2" staff_ref="staff_2"/>
                        …
                </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.

Prendendo in considerazione la gerarchia descritta nella figura, notiamo che le parti (part) sono di parti posizione gerarchica rispetto all'accollatura (staff_list) nonostante in teoria le prime siano associate ai pentagrammi.

Altra cosa degna di nota è proprio la costante introduzione di identificatori (attributi id) per ciascun nuovo elemento gerarchico. L'attributo id assegna un identificativo univoco all'evento: id duplicati costituiscono errore e il file XML 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. Ad esempio, avendo a che fare con una partitura complessa con molti strumenti, si potrebbe decidere di etichettare gli eventi con nomi che li distinguano temporalmente e li denotino voce per

voce: ad esempio “tromba_ev0”,…, “tromba_ev100”, “flauto_ev0”,…,”flauto_ev38”, e via dicendo. Ciò consente di poter fare precisi riferimenti nei livelli inferiori.

2.4.1 *Le proprietà dei pentagrammi (Staff)*

Il DTD evidenzia che uno staff_list (accollatura) deve contenere almeno un pentagramma.
Ogni pentagramma può contenere informazioni ripetute più volte, in quanto è possibile trovare nella stessa partitura: cambi di chiave, cambi di armatura, e cambio di tempo ecc. ecc. Saranno poi i riferimenti allo Spine da parte di tali elementi a individuare la loro corretta posizione spazio-temporale nella partitura.
Ciascun pentagramma contiene le informazioni su:

- Clef ovvero la chiave usata sul pentagramma

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

  - “type” rappresenta il tipo di chiave (sono supportate le chiavi di Sol, di Fa, di Do, nonché la chiave per le percussioni, quella per le intavolature e la doppia chiave di Sol);
  - “event_ref/staff_step” definiscono la posizione della chiave o più chiavi sul pentagramma;

- time_signature e il suo elemento figlio time_indication, rappresentano la divisione metrica della partitura

```
<time_signature event_ref="timesig_1">
    <time_indication num="4" den="4" vtu_amount="4000"/>
</time signature>
```

  - “event_ref” definisce l’evento dal quale avviene il cambio di metrica sul pentagramma;
  - “num/den” si tratta di numeratore e denominatore della frazione che viene utilizzata per indicare la segnatura di tempo;
  - “vtu_amount” è un attributo strettamente collegato allo Spine. Si tratta della somma dei vtu che compongono una battuta, così come li ha scelti l’utente nello

Spine. Il calcolo di tale valore è assai semplice: basta mettere in relazione il tempo con la scelta dei vtu a livello di Spine. Ad esempio, se il tempo è ¾ e nello Spine abbiamo assegnato vtu = 1 agli ottavi, è chiaro che il vtu_amount complessivo di una battuta da ¾ sia quello di 6 ottavi, ossia 6. Un esempio analogamente semplice ma con valori di vtu più grandi potrebbe essere: vtu = 1024 per il quarto, tempo di 2/2 e quindi vtu_amount = 1024 x 4 = 4096;

- key_signature che prevede uno o più elementi figli per spiegare come sia costruita l'armatura di chiave.

```
<key_signature event_ref="keysig_1">
    <flat_num number="2"/>
</key_signature>
```

  - "event_ref" definiscono la posizione dell'etichettatura corrispondente nello Spine;
  - possibili elementi figli di key_signature: "natural_num" (numero di bequadri in armatura), "sharp_num" (numero di diesis in armatura) e "flat_num" (numero di bemolli in armatura)

2.4.2 *Le battute (Measure)*

Measure rappresenta un elemento figlio di part, per cui la descrizione della partitura viene effettuata parte per parte: prima tutte le battute del flauto, poi tutte le battute della mano destra del pianoforte, poi tutta la mano sinistra del pianoforte. Invece, all'interno della misura vengono descritte tutte le voci che compongono la parte. E dunque, sempre nell'esempio citato, avremmo la descrizione della mano destra di battuta 1, immediatamente seguita dalla mano sinistra di battuta 1 e il pedale di battuta 1, per poi procedere alla mano destra di battuta 2 e via dicendo…

Schematicamente:

```
<los>
        <staff_list> … </staff_list>
        <part id="part_organo">
                <voice_list>
                        <voice_item id="mano_dx" staff_ref=”staff_1”/>
                        <voice_item id="mano_sx" staff_ref=”staff_1”/>
                        <voice_item id="pedale" staff_ref=”staff_1”/>
                </voice_list>
                <measure number="1">
                        <voice ref=”mano_dx”> … </voice>
                        <voice ref=”mano_sx”> … </voice>
                        <voice ref=”pedale”> … </voice>
                </measure>
                <measure number="2">
                        <voice ref=”mano_dx”> … </voice>
                        <voice ref=”mano_sx”> … </voice>
                        <voice ref=”pedale”> … </voice>
                </measure>
                …
        </part>
</los>
```

Gli attributi di measure sono

- “number”, ossia il numero di battuta.
- “id”, ossia un eventuale identificatore che potrebbe tornare utile per riferimenti alla particolare battuta. Si tratta di un attributo facoltativo.

Quali sono la segnatura di tempo, di chiave e l’armatura per la battuta corrente?
La risposta a questa domanda risiede non nell’elemento measure stesso bensì negli elementi figli di staff trattati nelle pagine precedenti. Quindi la chiave della battuta corrente si ricava scendendo in una voce (voice), considerandone gli eventi (rest o chord > notehead), risalendo dall’attributo staff_ref allo staff corrispondente e leggendo infine l’attributo, o gli attributi, key_signature figli di staff.

Capitolo 3

In questo capitolo vengono introdotti i linguaggi di programmazione utilizzati per la realizzazione del software. [2] [3]

3.1 C#

C# è un linguaggio di programmazione orientato agli oggetti, ovvero uno stile fondamentale di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi. Esso supporta i concetti d'incapsulamento, ereditarietà e polimorfismo.
L'*incapsulamento* è la proprietà per cui i dati che definiscono lo stato interno di un oggetto sono accessibili ai metodi dell'oggetto stesso, mentre non sono visibili ai *clients*. Per alterare lo stato interno dell'oggetto, è necessario quindi invocarne i metodi, ed è questo lo scopo principale dell'incapsulamento.
Il meccanismo dell'*ereditarietà* permette di derivare nuove classi a partire da quelle già definite. Una classe derivata attraverso l'ereditarietà, o *sottoclasse*, mantiene i metodi e gli attributi delle classi da cui deriva (*classi base*, o *superclassi*). Mediante l'ereditarietà, una classe può essere utilizzata come diversi tipi, ossia il tipo specifico della classe, qualsiasi tipo base o qualsiasi tipo interfaccia, se vengono implementate interfacce. Questa caratteristica è nota come *polimorfismo*.
In C# tutti i tipi sono polimorfici. Possono infatti essere utilizzati come tipo specifico o come istanza di Object, poiché qualsiasi tipo considera automaticamente Object come tipo base.
Il concetto di classe è alla base di ogni linguaggio di programmazione orientato agli oggetti come C#, ed ha la capacità di definire le caratteristiche di un insieme di oggetti che hanno proprietà e compiono azioni uguali. Una classe è identificata anche come "tipo di dato", infatti, porta con sé sia la rappresentazione dei dati, sia le operazioni sugli stessi.

3.2 LINQ

LINQ è l'acronimo di *Language INtegrated Query*, ed è una delle novità del .NET Framework 3.5. Rappresenta il primo framework Microsoft per l'accesso ai dati, indipendente dall'architettura e dalle strutture cui si tenta di accedere.

Con LINQ è possibile eseguire query e manipolare dati sfruttando un modello indipendente dalle varie tipologie di fonti; possiamo infatti accedere a database, file di testo, file XML, array, file Excel e qualsiasi altro oggetto riconducibile ad una collezione di oggetti enumerabile; il tutto utilizzando un unico modello di programmazione che riunisce molteplici tecniche differenti di accesso ai dati.

L'implementazione di LINQ per la gestione di dati in formato XML consente di compiere le normali operazioni di selezione, creazione e cancellazione di nodi, ricerca di valori, creazione di nuovi documenti e modifiche a documenti già esistenti (chiaramente in formato XML) utilizzando una sintassi molto chiara e che diminuisce notevolmente il numero di righe di codice da scrivere.

Il namespace di riferimento per queste nuove funzionalità è System.Xml.Linq.

Le operazioni che siamo in grado di compiere attraverso questa libreria sono più o meno le stesse messe a disposizione dal namespace System.Xml, con la differenza però che queste operazioni vengono esplicitate attraverso una nuova sintassi, più semplice ed intuitiva.

Con LINQ, è possibile:

- caricare informazioni in formato XML in memoria.
- Creare nuovi alberi XML.
- Inserire, modificare o cancellare nodi da alberi XML in memoria.
- Salvare le informazioni create o gestite in memoria su file.

Inoltre contiene nuove classi per rappresentare documenti, elementi, attributi, nomi e in generale informazioni presenti nei documenti XML e per mantenere tali dati in memoria.

La novità che salta subito all'occhio utilizzando queste nuove classi è che c'è la possibilità di rappresentare ogni singolo oggetto di un documento XML, senza dover creare il contesto che rappresenta l'intero documento XML. Per manipolare un singolo elemento, ci basta classe XElement, senza dover per forza definire il documento XML cui questo elemento appartiene.

3.3 MONO DEVELOP

Mono Develop è un ambiente di sviluppo open source realizzato principalmente per C# e altri linguaggi .NET.

Tra le varie caratteristiche di MonoDevelop, il supporto allo sviluppo multipiattaforma (Linux, Windows e Mac OSX), l'ambiente di lavoro configurabile, il supporto a vari linguaggi (C#, Visual Basic, C/C++ e altri).

3.4 VISUAL STUDIO

Visual Studio è un ambiente di sviluppo integrato sviluppato da Microsoft, che supporta attualmente diversi tipi di linguaggio, quali C, C++, C#, F#, Visual Basic .Net e ASP .Net, e che permette la realizzazione di applicazioni, siti web, applicazioni web e servizi web.

In particolare, per questo progetto è stata sfruttata la libreria di Windows Form, che permette di progettare la finestra del prototipo.

Capitolo 4

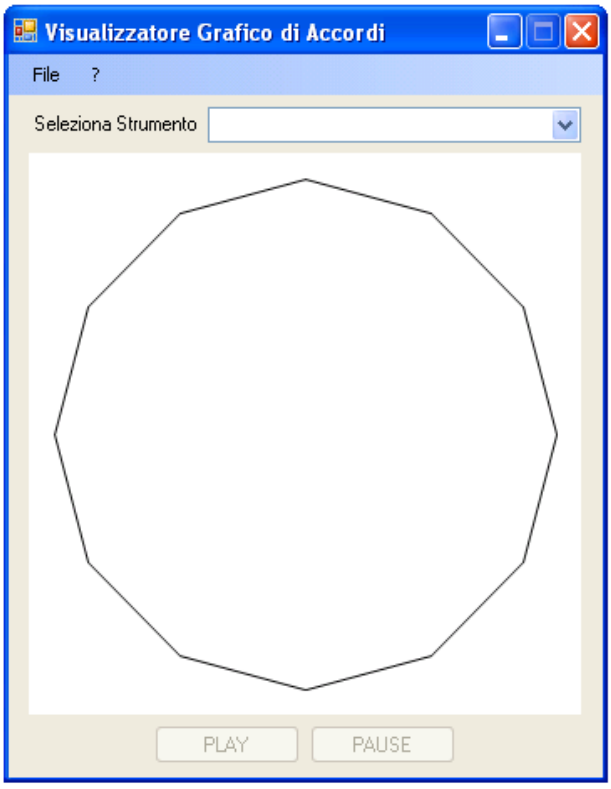
Dopo aver introdotto i linguaggi utilizzati, si procede ad analizzare il prototipo stesso nelle sue parti fondamentali, facendo esempi riferiti al codice. Esso è stato sviluppato tramite MonoDevelop in C# in modo da poter funzionare in ambiente multipiattaforma, sia in ambiente Windows o Mac.

4.1 LIBRERIE E GRAFICA

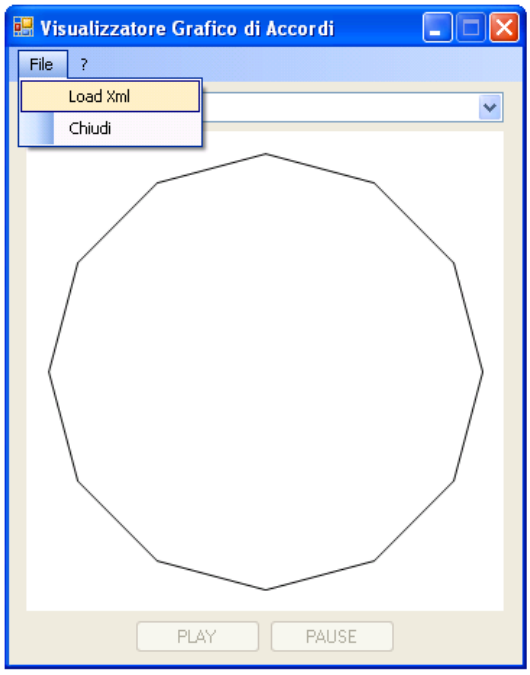
Le librerie utilizzate sono le seguenti:

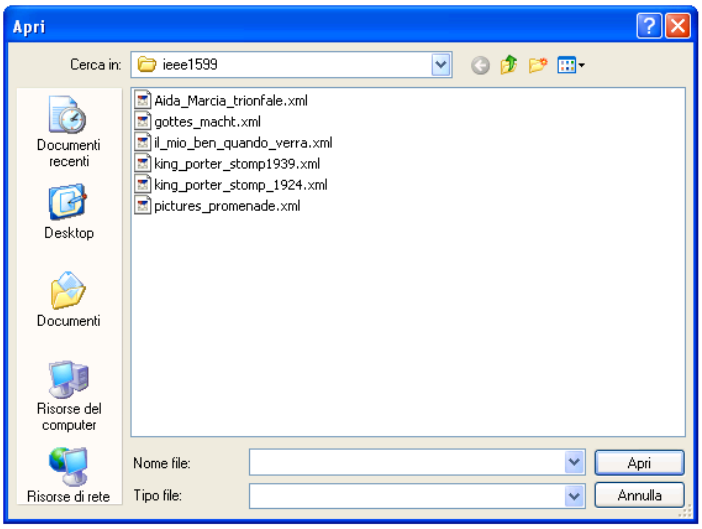
```
using System;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Linq;
```

In particolare, bisogna importare la libreria *System.Xml* e *System.Xml.Linq* utili a sfruttare le potenzialità, già descritte, di Linq, allo scopo di manipolare il file XML che verrà caricato dall'utente.
Il prototipo, al momento dell'avvio, presenterà la seguente visualizzazione:
Il Software, ricevendo in input un file XML IEEE1599, avvierà le sue funzionalità.



4.2 CARICAMENTO FILE

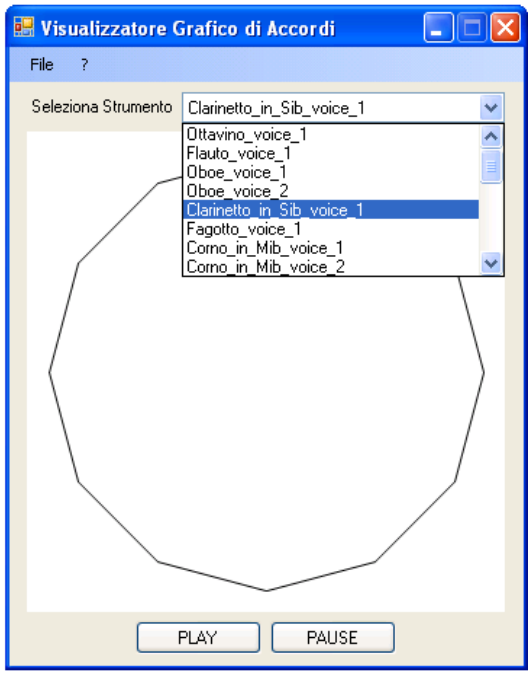




L'apertura della finestra, che dà la possibilità, all'utente di caricare i file XML è generata dal seguente codice:

```
OpenFileDialog openFile = new OpenFileDialog();
openFile.ShowDialog();
nomefile = openFile.FileName;

XDocument doc = XDocument.Load(nomefile);

var voice_item = from v in doc.Descendants("logic")...
                 select (string)v.Attribute("id");

foreach (var vi in voice_item)
{
    comboBox1.Items.Add(vi);
    VOICE.Add(vi);
}

comboBox1.SelectedIndex = 0;
```

Inoltre, nel momento stesso in cui viene selezionato e aperto un file XML, il software interroga lo stesso file, tramite una query "var voice_item", sopra indicata, che permette di estrapolare tutte le voci (voice_item) contenute in partitura. Una volta estrapolate le voci, le inserisce nella ComboBox, controllo grafico che permette all'utente di selezionare una di esse tramite un elenco.



Contemporaneamente all'apertura del file, il software compie una serie di operazioni fondamentali per i successivi passaggi, che permettono la memorizzazione di dati riguardanti tutte le "voci".

```
Dictionary<string, List<info>> STRUCT_INFOVOICE = new Dictionary<string, List<info>>();

 for (int contvoice = 0; contvoice < VOICE.Count; contvoice += 1)
 {
     List<info> LISTASTRUCT = new List<info>();

     //...........(codice)

     var measure = from d in doc.Descendants("logic").Descendants("los").....................
                   where d.Attribute("voice_item_ref").Value == VOICE[contvoice]
                   select d;

     int contevent = 0;
     foreach (XElement c in measure)
     {
         var chordrest = from b in c.Elements()
                         select b;
         foreach (XElement b in chordrest)
         {
             List<string> LISTA_CHORDREST = new List<string>();

             //.............(codice)

             info tabellastruct;
             tabellastruct.NumUnità = NUM_UNITA[contevent];
             tabellastruct.RestChord = LISTA_CHORDREST;
             LISTASTRUCT.Add(tabellastruct);
             contevent += 1;
         }
     }
         STRUCT_INFOVOICE[VOICE[contvoice]] = LISTASTRUCT;
 }
```

In particolare, per ogni "voce" viene creata una lista LISTASTRUCT di tipo <info>.

Il tipo <info> è stato precedentemente elaborato dal software attraverso la creazione di una struct con le seguenti estensioni:

```
public struct info
{
    public List<string> RestChord;
    public int NumUnità;
}
```

Le **LISTASTRUCT** create prendono il nome di ogni singola "voce". Ogni lista si estende per tutti gli event_ref (nota o pausa) di ogni voce. Vengono inseriti nella LISTASTRUCT le coppie di valori **RestChord** e **NumUnità.** Ogni evento, quindi, viene descritto dal numero di volte che l'evento stesso si ripete (NumUnità) e da una relativa lista di note o pause che esso rappresenta (RestChord).
Le informazioni relative allo struct, sono state precedentemente calcolate dal software:

- NumUnità, è stata calcolata tenendo conto del *vtu_amount*, contenuto nel layer los del XML. Sapendo che il *vtu_amount* rappresenta l'estensione in vtu della battuta, è possibile calcolare la durata in unità di ogni singolo evento. Ciò avviene estrapolando il *num* (numeratore) e il *den* (denominatore) che caratterizzano la durata di ogni singola nota o pausa in partitura [*duration(D)*] e il relativo *num* e *den* che rappresentano la segnatura di tempo della partitura [*time_indication(T)*]. La relazione tra duration e time_indication è:

  DT = (DenD * NumT / (NumD * DenT)

  Nel caso in cui il *vtu_amount* fosse un valore alto, è stato implementato un processo che divide lo stesso valore del vtu per il massimo comune divisore (MCD) calcolato tra tutti i timing contenuti nello Spine dell'XML. Così facendo, il valore del *vtu_amount* potrebbe essere sostituito con valore molto più basso senza intaccare in alcun modo il funzionamento del software, ma anzi velocizzandolo. Ne deriva che:

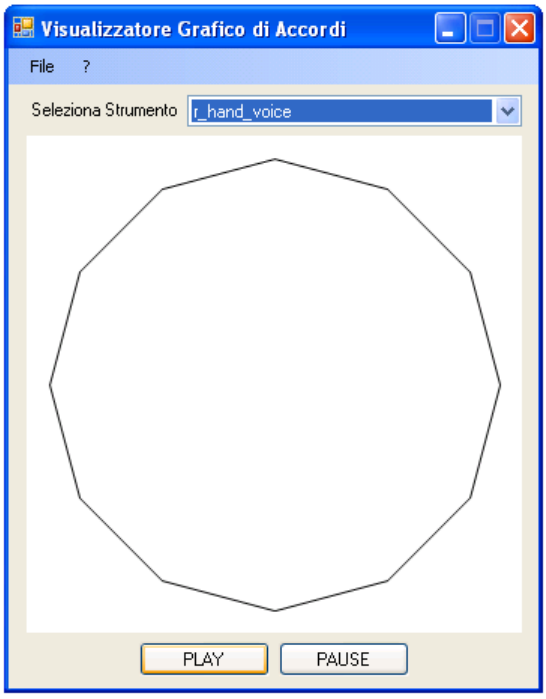
  NumUnità = (vtu_amount / MCDtiming) / DT

  Successivamente è stato aggiornato questo valore tenendo conto della possibile presenza del punto (`augmentation_dots`) che ha lo scopo di aumentare di metà il valore della figura musicale.

- RestChord, ovvero la lista contenente note o pause dell'evento relativo, è stata estrapolata tramite delle query che vanno a interrogare il file XML nel sublayer los. In particolare, vengono distinte note (*chord*) e pause (*rest*), tenendo conto della possibile presenza di alterazioni (*actual_accidental*). Una volta rilevate le pause e le note con la loro relativa alterazione, sono state memorizzate tramite numeri (da 01 a 12 come tutti i semitoni), per facilitare l'implementazione grafica che verrà successivamente sviluppata.

4.3 PLAY & PAUSE

Dopo aver caricato il file XML e dopo aver selezionato la voce, vengono abilitati i button Play e Pause.



Essi hanno il compito di avviare o mettere in pausa la visualizzazione grafica del relativo strumento scelto, dunque sono legati ad un timer che è impostato con un certo intervallo.

```
private void buttonPlay_Click(object sender, EventArgs e)
{

    timer1.Tick += new EventHandler(timer1_Tick);
    timer1.Interval = 200;
    timer1.Start();
}

private void buttonPause_Click(object sender, EventArgs e)
{
    timer1.Stop();
}
```

Per ogni singolo intervallo di tempo, entra in esecuzione un ciclo for lungo quanto tutti gli event_ref della voce che è stata selezionata. Attingendo alla struct riempita in precedenza, il software collega, all'evento in esecuzione, l'informazione che permette di riprodurre un certo numero volte, lo stesso (NumUnità) e la lista relativa di pause o note (RestChord). Si crea una lista contenente le note/pause di quel singolo istante RESTCHORD_SingolaUnità (che poi il software pulirà ogni volta che rientra nel ciclo).

```
int voce_selezionata = comboBox1.SelectedIndex;

var event_ref = from er in doc.Descendants("logic").Descendants("los").....................
                where er.Attribute("voice_item_ref").Value == VOICE[voce_selezionata]
                from t in er.Elements()
                select (string)t.Attribute("event_ref").Value;

foreach (var eventr in event_ref)
{
    EVENT_VoceSelezionata.Add(eventr);
}

RESTCHORD_SingolaUnità.Clear();

for (int b = 0; b < EVENT_VoceSelezionata.Count(); b++)
{
    if (click < STRUCT_INFOVOICE[VOICE[voce_selezionata]][b].NumUnità)
    {
        foreach (var rc in STRUCT_INFOVOICE[VOICE[voce_selezionata]][b].RestChord)
        {
            RESTCHORD_SingolaUnità.Add(rc);
        }

        break;
    }

}

pictureBox1.Invalidate();
click++;
```

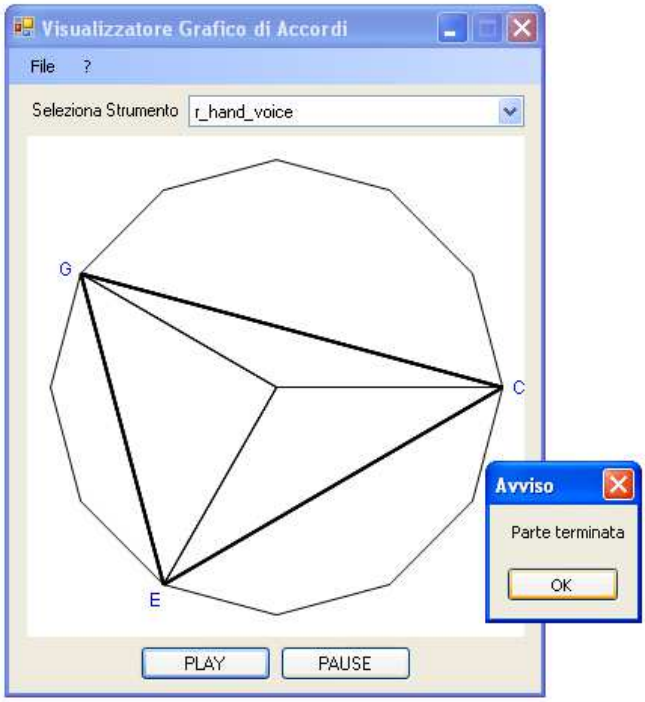
Precedentemente è stato inizializzato un contatore click che viene incrementato ogni qual volta il programma esegue le operazioni del ciclo for (timer). Durante il ciclo for, il contatore, viene relazionato al NumUnità del singolo evento. Se esso risulta minore del secondo, verrà creata una lista RESTCHORD_SingolaUnità contenente le stesse note/pause dell'istante precedente. Facendo così, verranno ripetute le note/pause un certo numero di volte in relazione alla durata del singolo evento. Una volta che il contatore click ha raggiunto lo stesso numero contenuto in NumUnità, passa all'evento successivo.

Inoltre picturebox.invalidate consente, ad ogni click, di invalidare l'intera superficie della picturbox e determinare una nuova grafica, descritta dal evento picturbox_Paint che verrà spiegata in seguito.

Ripete tutto questo procedimento per tutte le unità totali che compongono la partitura e fino alla sua conclusione. Questa procedura viene attivata attraverso questo controllo:

```
if(contatoreclick<NUM_UNITATOTALI.Last())
{contatoreclick+=1;}
else
{
timer1.Stop();
MessageBox.Show("Parte terminata","Avviso");
buttonPlay.Enabled = false;
buttonPause.Enabled = false;
contatoreclick = 0;
click = 0;
}
```

Il Software, in questo caso, genera un altro contatore (contatoreclick) che viene messo in relazione alla lista NUM_UNITATOTALI (creata precedentemente). Nello stesso istante in cui il contatore-click raggiunge lo stesso valore dell'ultimo numero contenuto in NUM_UNITATOTALI, azzera tutti i contatori utilizzati fino a quel momento e stampa a video un avviso di avvenuta conclusione della parte.

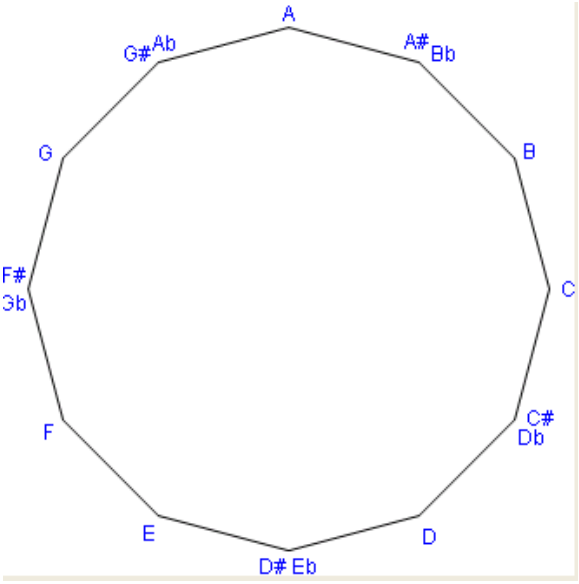


4.4 RAPPRESENTAZIONE GRAFICA DEGLI ACCORDI

Nel evento picturbox_Paint, il software ha il compito di passare in rassegna il contenuto di ogni lista RESTCHORD_SingolaUnità, creata in precedenza ad ogni click, e disegnare nella picturbox la relativa rappresentazione grafica.

Ad ogni singola nota estrapolata dalla lista, corrispondono delle precise coordinate (x,y), rappresentanti un punto specifico della picturebox. Al contrario, alle pause non corrisponde alcuna coordinata (x,y).

Le coordinate sono state scelte in modo tale da formare un dodecagono, ove i 12 punti risultanti, rappresentano i 12 semitoni possibili. Inoltre ad ogni singola nota, corrisponde la relativa notazione anglosassone come rappresentato di seguito.

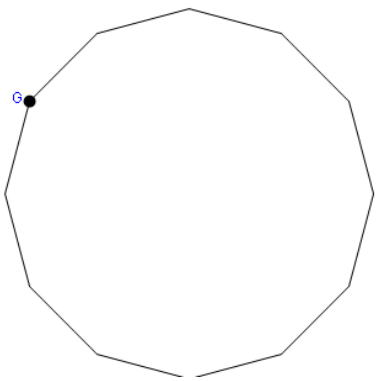


Il prototipo, a questo punto, memorizza le coordinate delle note suonate in quel particolare istante e disegna nella picturebox la relativa rappresentazione grafica sfruttando proprio le coordinate dei punti per inserire puntatori a forma ellittica (FillEllipse), per tracciare linee (Drawline), ecc..

Ora si procede analizzando, come il software gestisce i vari casi:

1. Nel caso in cui il numero di note presenti nella lista RESTCHORD_SingolaUnità sia 1, viene semplicemente disegnato un puntatore a forma ellittica (FillEllipse).
   Inoltre, nel momento in cui l'elemento presente nella lista fosse una pausa, il software non esegue alcuna operazione e la picturebox rimane vuota.

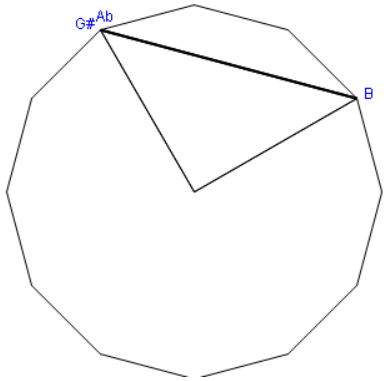
```
if(RESTCHORD_SingolaUnità.Count==1)
{
    if (notepause == "01")
    {
        e.Graphics.FillEllipse(Brushes.Black, 160, 10, 10, 10);
    }
    else if (notepause == "02")
    {
        e.Graphics.FillEllipse(Brushes.Black, 235, 30, 10, 10);
    }

    //.......(codice)

    else if (notepause == "12")
    {
        e.Graphics.FillEllipse(Brushes.Black, 85, 30, 10, 10);
    }
}
```



2. Nel caso in cui il numero di note presenti nella lista RESTCHORD_SingolaUnità sia 2, viene tracciata una linea (Drawline) di spessore 2 tra le due coordinate che rappresentano le note.

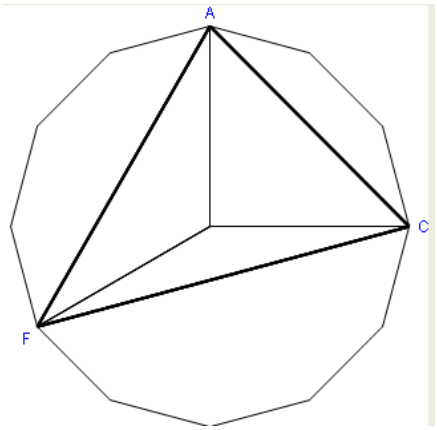
   Inoltre, è possibile distinguere il caso in cui sono suonate due note uguali (ottave diverse), dal caso precedente in cui veniva rappresentata una singola nota: dal centro del dodecagono si attiva una linea che si collega alle coordinate della nota suonata.

```
if (RESTCHORD_SingolaUnità.Count == 2)
{
    e.Graphics.DrawLine(
       new Pen(Color.Black, 1f),
       PuntiXY[0],
       new Point(165, 165));
    e.Graphics.DrawLine(
       new Pen(Color.Black, 1f),
       PuntiXY[1],
       new Point(165, 165));

    e.Graphics.DrawLine(
       new Pen(Color.Black, 2f),
       PuntiXY[0],
       PuntiXY[1]);
}
```



3. Nel caso in cui il numero di note presenti nella lista RESTCHORD_SingolaUnità sia 3, vengono eseguite le stesse operazioni del secondo caso aggiungendo la coordinata della terza nota. E così anche nel caso in cui nella lista il numero di elementi sia 4, 5, 6, ecc..

```
if (RESTCHORD_SingolaUnità.Count == 3)
{
    e.Graphics.DrawLine(
       new Pen(Color.Black, 1f),
       PuntiXY[0],
       new Point(165, 165));
    e.Graphics.DrawLine(
       new Pen(Color.Black, 1f),
       PuntiXY[1],
       new Point(165, 165));
    e.Graphics.DrawLine(
       new Pen(Color.Black, 1f),
       PuntiXY[2],
       new Point(165, 165));

    e.Graphics.DrawLine(
       new Pen(Color.Black, 2f),
       PuntiXY[0],
       PuntiXY[1]);
    e.Graphics.DrawLine(
       new Pen(Color.Black, 2f),
       PuntiXY[1],
       PuntiXY[2]);
    e.Graphics.DrawLine(
       new Pen(Color.Black, 2f),
       PuntiXY[2],
       PuntiXY[0]);
}
```



Capitolo 5

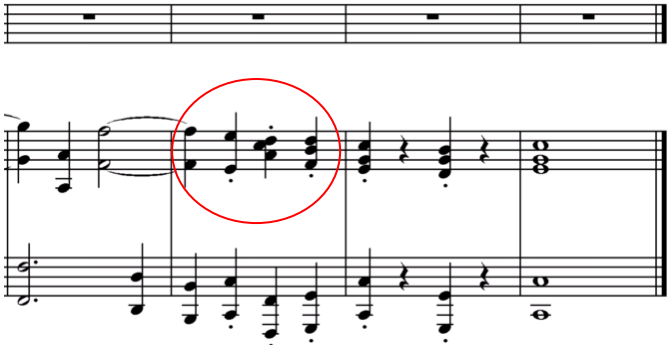
Dopo aver analizzato il prototipo nelle sue parti fondamentali, viene svolto un test per poter verificare l'effettivo funzionamento del software progettato e come esso si comporta a seconda del tipo di dati che preleva dal file XML.

5.1 TEST GOTTES MACHT

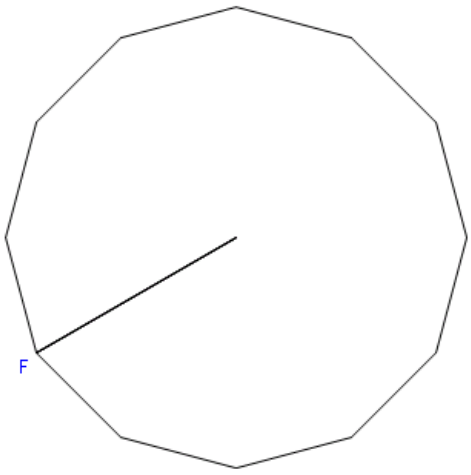
Per questo test, si è utilizzato il file XML "gottes_macht.xml" (Gottes Macht - Beethoven), analizzando la seconda voce "r_hand_voice", in particolare le ultime tre battute (16,17,18). Di ogni battuta verranno analizzati gli eventi contenuti in essa (pause o note), calcolando la relativa NumUnità e la loro rispettiva durata in millisecondi. Questo brano possiede le seguenti caratteristiche:

- vtu_amount/MCDtiming = 8 (vtu_amount=8 MCDtiming=1)
- time_indication= 1/1 (NumT=1 DenT=1)
- intervallo timer = 200 ms

5.2.1 *Misura n°16*



```
<measure number="16">
  <voice voice_item_ref="r_hand_voice">
    <chord event_ref="v2_e39">
      <duration num="1" den="4" />
      <notehead staff_ref="staff_2">
        <pitch step="F" octave="6" />
      </notehead>
      <notehead staff_ref="staff_2">
        <pitch step="F" octave="5" />
      </notehead>
    </chord>
```

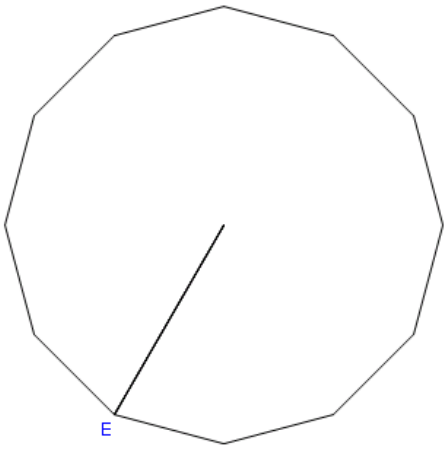


Evento: v2_e39 (F,F)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="v2_e40">
  <duration num="1" den="4" />
  <notehead staff_ref="staff_2">
    <pitch step="E" octave="6" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="E" octave="5" />
  </notehead>
  <articulation>
    <staccato />
  </articulation>
</chord>
```

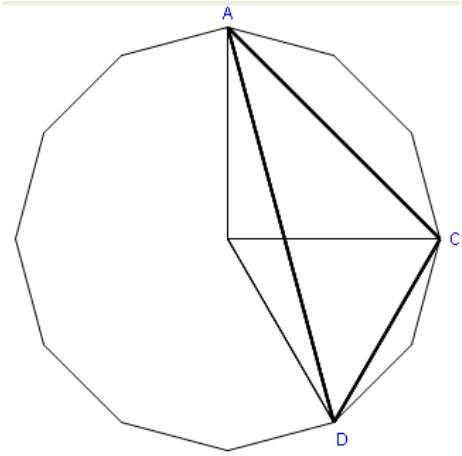


Evento: v2_e40 (E,E)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="v2_e41">
  <duration num="1" den="4" />
  <notehead staff_ref="staff_2">
    <pitch step="D" octave="6" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="C" octave="6" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="A" octave="5" />
  </notehead>
  <articulation>
    <staccato />
  </articulation>
</chord>
```

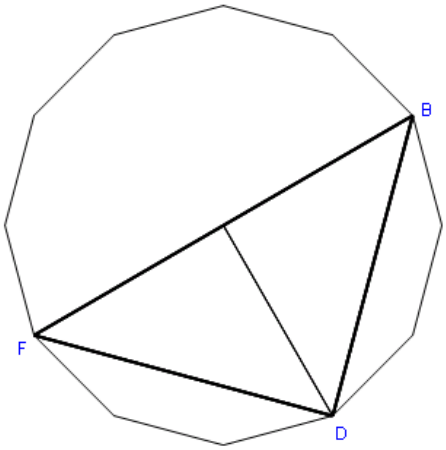


Evento: v2_e41 (D,C,A)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="v2_e42">
  <duration num="1" den="4" />
  <notehead staff_ref="staff_2">
    <pitch step="D" octave="6" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="B" octave="5" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="F" octave="5" />
  </notehead>
  <articulation>
    <staccato />
  </articulation>
</chord>
</voice>
```

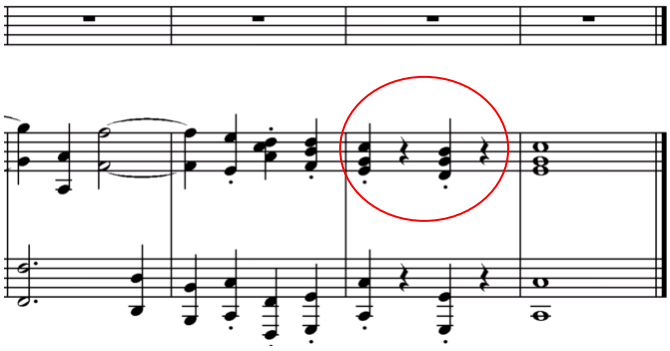


Evento: v2_e42 (D,B,F)

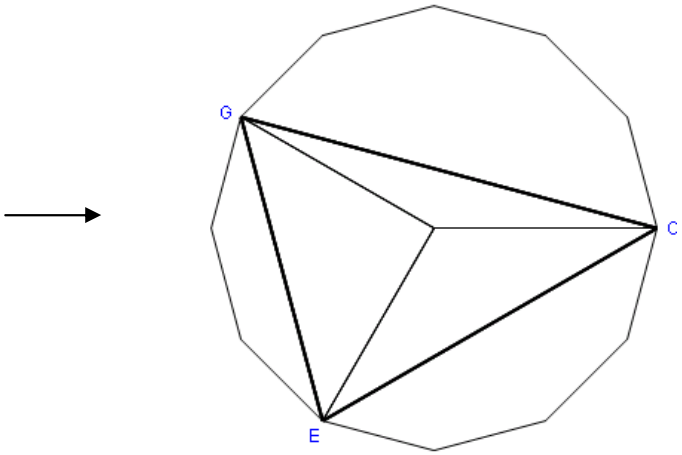
NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

5.2.2 *Misura n°17*



```
        <voice voice_item_ref="l_hand_voice">
        </measure>
        <measure number="17">
          <voice voice_item_ref="r_hand_voice">
            <chord event_ref="v2_e43">
              <duration num="1" den="4" />
              <notehead staff_ref="staff_2">
                <pitch step="C" octave="6" />
              </notehead>
              <notehead staff_ref="staff_2">
                <pitch step="G" octave="5" />
              </notehead>
              <notehead staff_ref="staff_2">
                <pitch step="E" octave="5" />
              </notehead>
              <articulation>
                <staccato />
              </articulation>
            </chord>
```

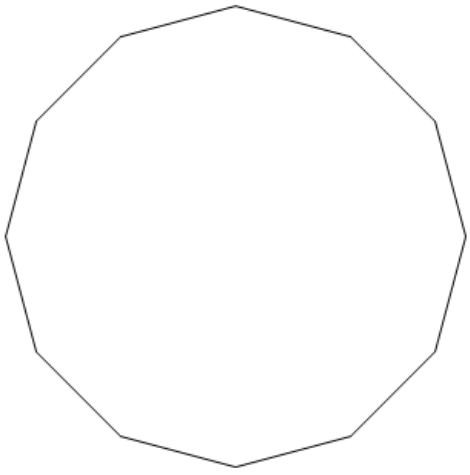


Evento: v2_e43 (C,G,E)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

```
            <rest event_ref="v2_e44" staff_ref="staff_2">
              <duration num="1" den="4" />
            </rest>
```

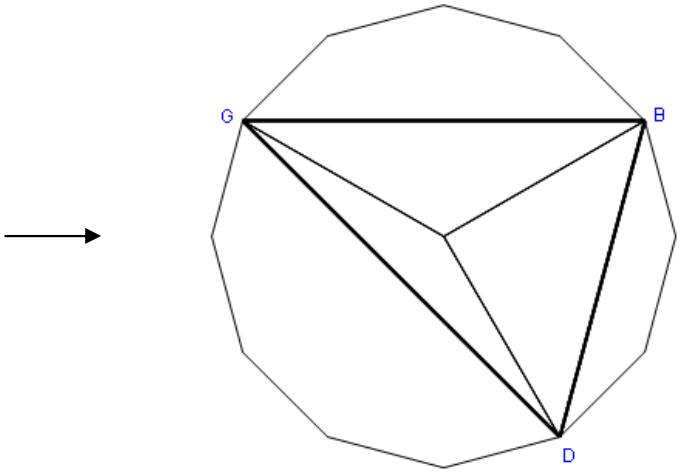


Evento: v2_e44 (PAUSA)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="v2_e45">
  <duration num="1" den="4" />
  <notehead staff_ref="staff_2">
    <pitch step="B" octave="5" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="G" octave="5" />
  </notehead>
  <notehead staff_ref="staff_2">
    <pitch step="D" octave="5" />
  </notehead>
  <articulation>
    <staccato />
  </articulation>
</chord>
```

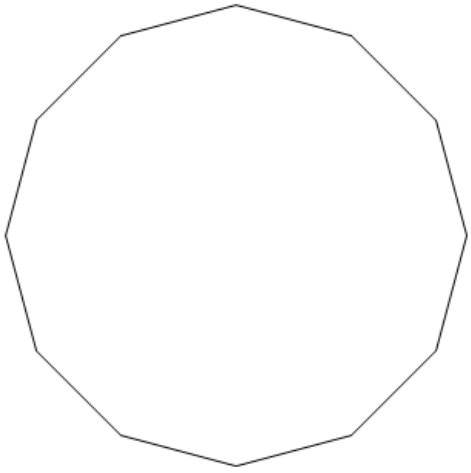


Evento: v2_e45 (B,G,D)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

```
<rest event_ref="v2_e46" staff_ref="staff_2">
  <duration num="1" den="4" />
</rest>
</voice>
```

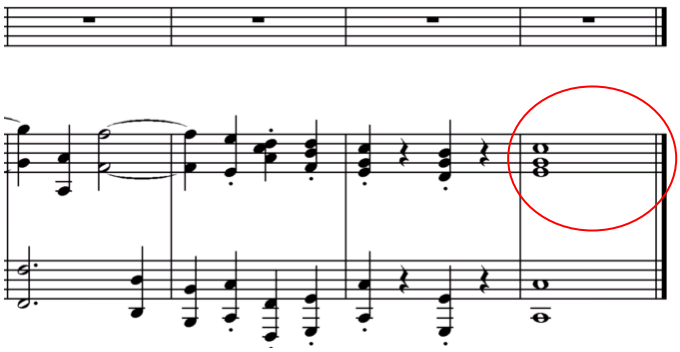


Evento: v2_e46 (PAUSA)

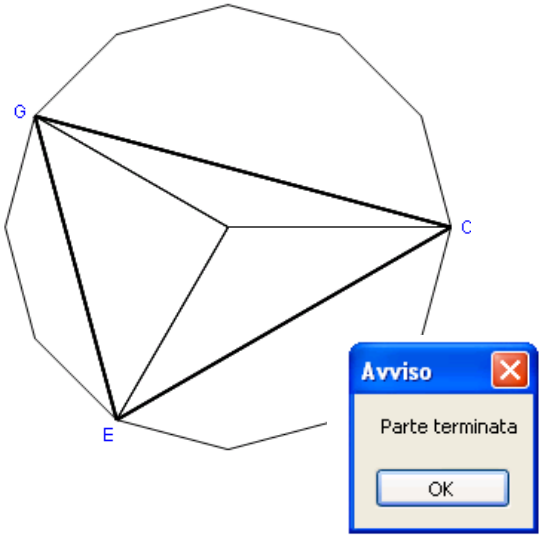
NumUnità: 8 / [(DenD*1) / (NumD*1)] = 2

Durata: 200ms * 2 = 400ms

5.2.3 *Misura n°18*



```
<measure number="18">
  <voice voice_item_ref="r_hand_voice">
    <chord event_ref="v2_e47">
      <duration num="1" den="1" />
      <notehead staff_ref="staff_2">
        <pitch step="C" octave="6" />
      </notehead>
      <notehead staff_ref="staff_2">
        <pitch step="G" octave="5" />
      </notehead>
      <notehead staff_ref="staff_2">
        <pitch step="E" octave="5" />
      </notehead>
    </chord>
  </voice>
```



Evento: v2_e47 (C,G,E)

NumUnità: 8 / [(DenD*1) / (NumD*1)] = 8

Durata: 200ms * 8 = 1600ms

Inoltre essendo v2_e47 l'ultimo evento della voce "r_hand_voice", verrà visualizzato l'avviso di avvenuta conclusione della parte.

5.3 TEST PICTURE PROMENADE

Per questo ulteriore test, si è utilizzato il file XML "pictures promenade.xml" (Pictures Promenade - Mussorgskij), analizzando la seconda voce "piano_dx1", in particolare le ultime due battute (23,24). Di ogni battuta verranno analizzati gli eventi contenuti in essa (pause o note), calcolando la relativa NumUnità e la loro rispettiva durata in millisecondi.

In particolare è stato svolto questo test poiché nel file in questione, sono presenti due diversi vtu_amount. Infatti, bisogna precisare che il prototipo non è stato creato per gestire l'eventualità in cui ci siano due o più valori di vtu_amount. Perciò, per effettuare il test sul presente brano, è stato impostato manualmente il vtu_amount relativo alle battute in analisi (6144). Con questo sistema, è stato possibile avviare il prototipo senza mandarlo in errore.

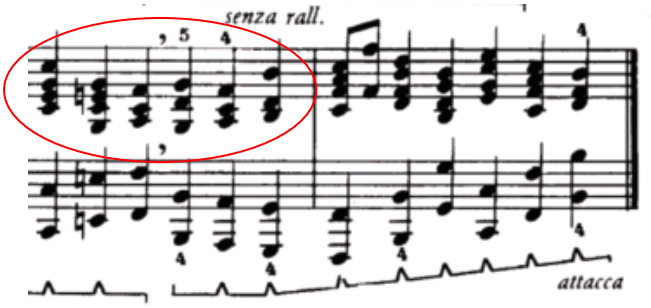
Questo brano ha le seguenti caratteristiche:

- vtu_amount/MCDtiming= 12 (vtu_amount=6144 MCDtiming=514)
- time_indication= 6/4 (NumT=6 DenT=4)
- intervallo timer = 200 ms

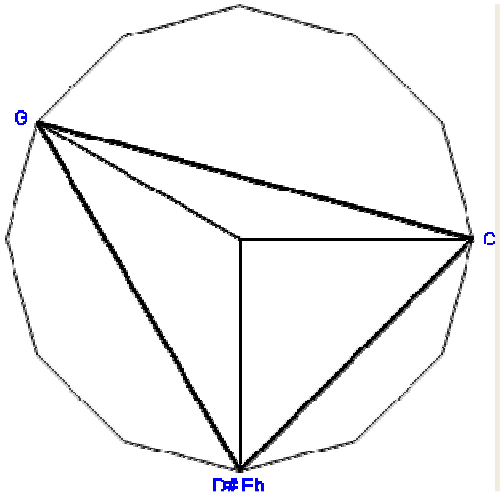
I calcoli del prototipo, come si può notare dal test, risultano coerenti.

Questo brano, contenente due vtu_amount, è stato eseguito correttamente dal software solo grazie alla gestione manuale di uno dei due vtu. La possibilità di gestire più vtu in modo automatico, potrà essere uno degli sviluppi futuri, realizzabili attraverso l'implementazione del prototipo.

5.3.1 *Misura n°23*




```
<measure number="23">
  <voice voice_item_ref="piano_dx1">
    <chord event_ref="dx1_359">
      <duration num="1" den="4"/>
      <notehead>
        <pitch step="C" octave="6"/>
      </notehead>
      <notehead>
        <pitch step="G" octave="5"/>
      </notehead>
      <notehead>
        <pitch step="E" actual_accidental="flat" octave="5"/>
      </notehead>
      <notehead>
        <pitch step="C" octave="5"/>
      </notehead>
    </chord>
```

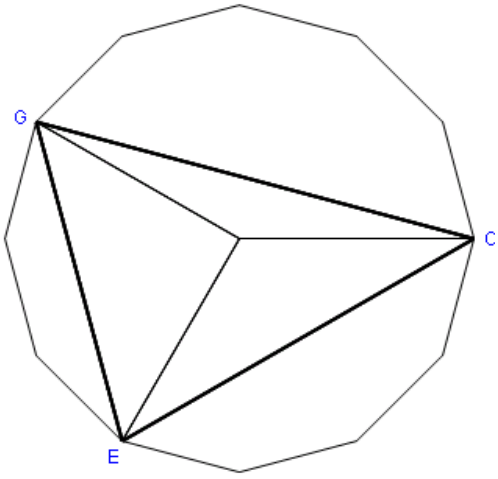


Evento: dx1_359 (C,G,Eb,C)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
    <chord event_ref="dx1_363">
      <duration num="1" den="4"/>
      <notehead>
        <pitch step="G" octave="5"/>
      </notehead>
      <notehead>
        <pitch step="E" octave="5" actual_accidental="natural"/>
        <printed_accidentals>
          <natural/>
        </printed_accidentals>
      </notehead>
      <notehead>
        <pitch step="C" octave="5"/>
      </notehead>
      <notehead>
        <pitch step="G" octave="4"/>
      </notehead>
    </chord>
```

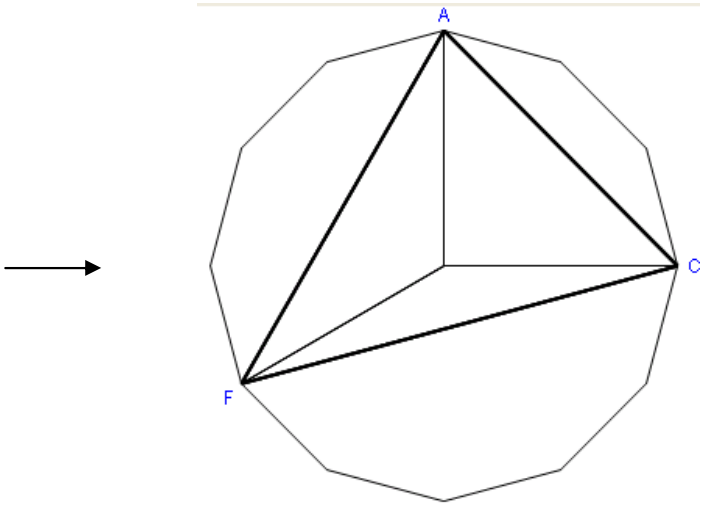


Evento: dx1_363 (G,E,C,G)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_367">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="F" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="C" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="A" octave="4"/>
  </notehead>
</chord>
```

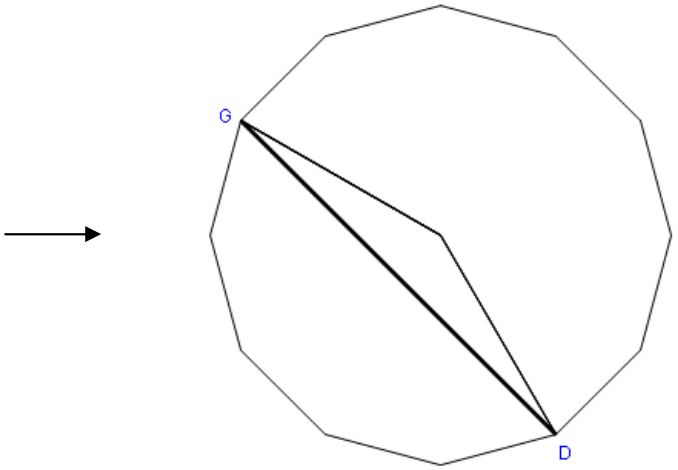


Evento: dx1_363 (F,C,A)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_370">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="G" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="D" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="G" octave="4"/>
  </notehead>
</chord>
```

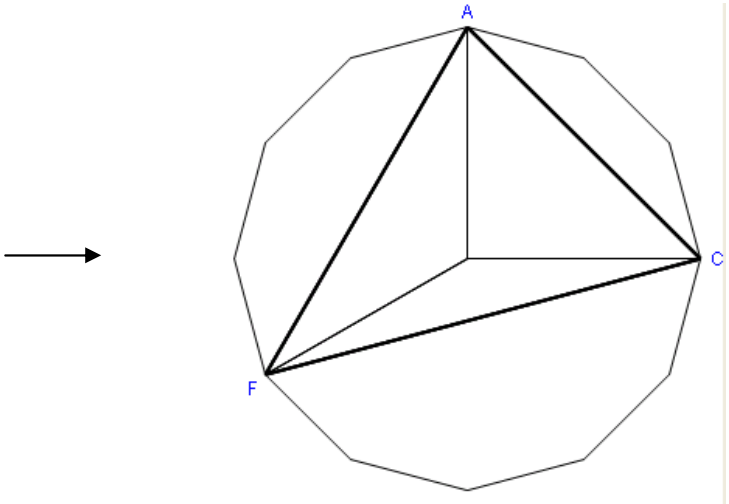


Evento: dx1_370 (G,D,G)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_373">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="F" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="C" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="A" octave="4"/>
  </notehead>
</chord>
```

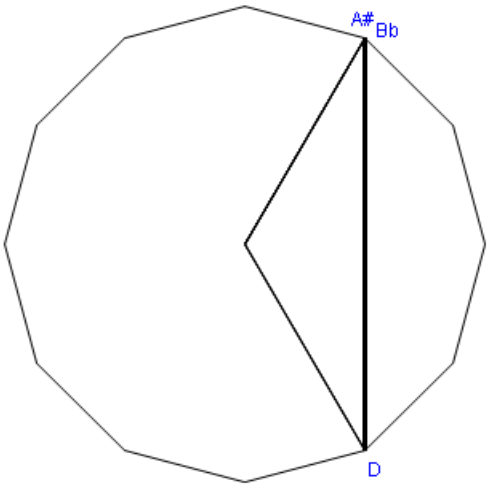


Evento: dx1_370 (F,C,A)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_376">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="B" actual_accidental="flat" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="D" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="B" actual_accidental="flat" octave="4"/>
  </notehead>
</chord>
</voice>
```

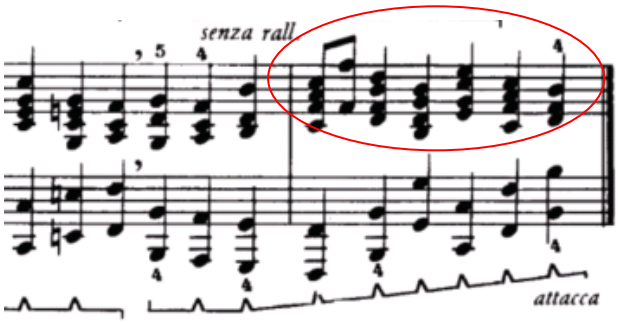


Evento: dx1_376 (B,D,B)

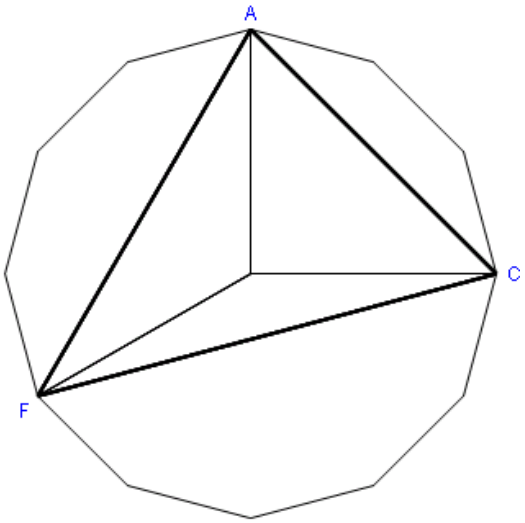
NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

5.3.2 *Misura n°24*




```
<measure number="24">
  <voice voice_item_ref="piano_dx1">
    <chord event_ref="dx1_379">
      <duration num="1" den="8"/>
      <notehead>
        <pitch step="C" octave="6"/>
      </notehead>
      <notehead>
        <pitch step="A" octave="5"/>
      </notehead>
      <notehead>
        <pitch step="F" octave="5"/>
      </notehead>
      <notehead>
        <pitch step="C" octave="5"/>
      </notehead>
    </chord>
```

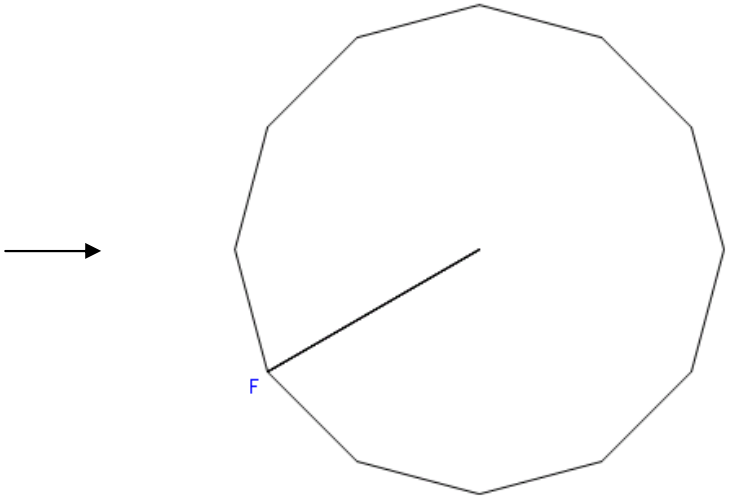


Evento: dx1_379 (C,A,F,C)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 1

Durata: 200ms * 1 = 200ms

```
<chord event_ref="dx1_383">
  <duration num="1" den="8"/>
  <notehead>
    <pitch step="F" octave="6"/>
  </notehead>
  <notehead>
    <pitch step="F" octave="5"/>
  </notehead>
</chord>
```

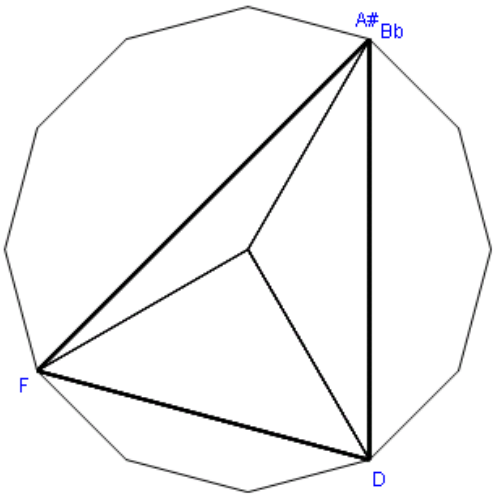


Evento: dx1_383 (F,F)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 1

Durata: 200ms * 1 = 200ms

```
<chord event_ref="dx1_385">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="D" octave="6"/>
  </notehead>
  <notehead>
    <pitch step="B" actual_accidental="flat" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="F" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="D" octave="5"/>
  </notehead>
</chord>
```

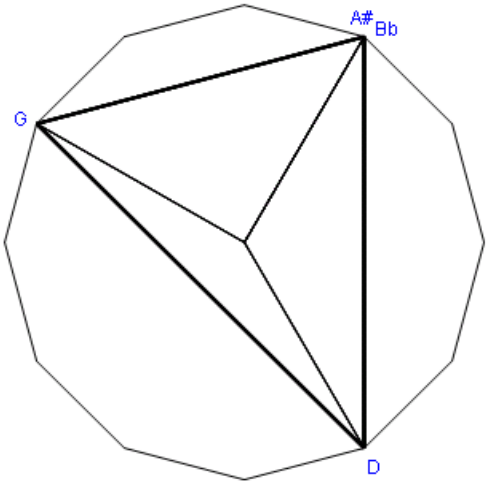


Evento: dx1_385 (D,Bb,F,D)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_389">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="B" actual_accidental="flat" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="G" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="D" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="B" actual_accidental="flat" octave="4"/>
  </notehead>
</chord>
```

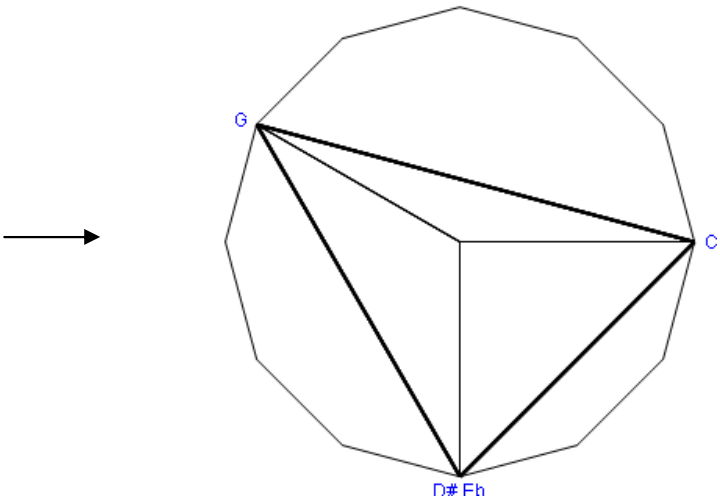


Evento: dx1_389 (Bb,G,D,Bb)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_393">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="E" actual_accidental="flat" octave="6"/>
  </notehead>
  <notehead>
    <pitch step="C" octave="6"/>
  </notehead>
  <notehead>
    <pitch step="G" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="E" actual_accidental="flat" octave="5"/>
  </notehead>
</chord>
```

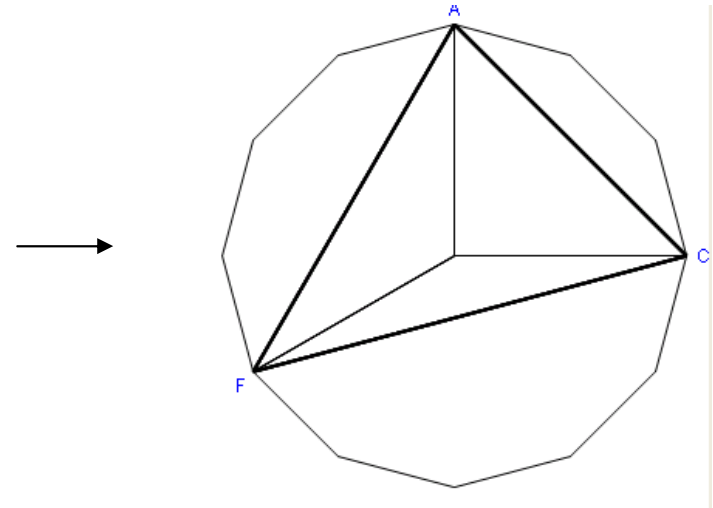


Evento: dx1_389 (Eb,C,G,Eb)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_397">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="C" octave="6"/>
  </notehead>
  <notehead>
    <pitch step="A" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="F" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="C" octave="5"/>
  </notehead>
</chord>
```

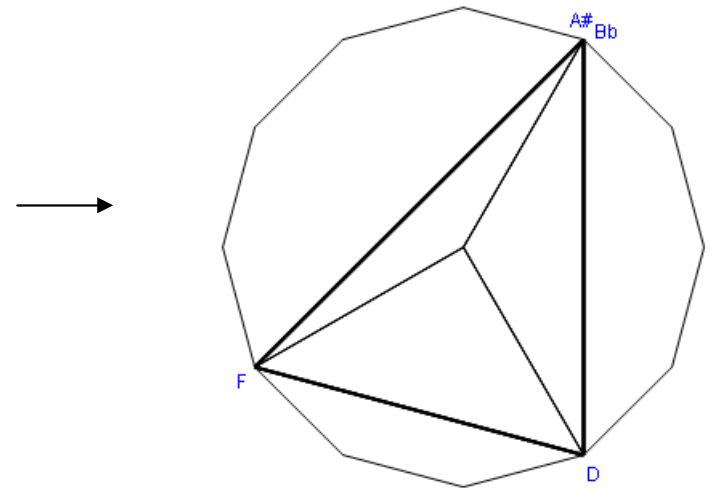


Evento: dx1_389 (C,A,F,C)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

```
<chord event_ref="dx1_401">
  <duration num="1" den="4"/>
  <notehead>
    <pitch step="B" actual_accidental="flat" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="F" octave="5"/>
  </notehead>
  <notehead>
    <pitch step="D" octave="5"/>
  </notehead>
</chord>
</voice>
<voice voice_item_ref="piano_sx">
</measure>
```



Evento: dx1_389 (Bb,F,D)

NumUnità: 12 / [(DenD*6) / (NumD*4)] = 2

Durata: 200ms * 2 = 400ms

Conclusioni

6.1 PROBLEMI APERTI

Si è visto come il prototipo, analizzando un file XML contenente un solo vtu_amount, possa essere eseguito correttamente. La presenza, invece, di due o più vtu è gestibile solo manualmente attraverso l'inserimento del vtu relativo alle battute analizzate. La stessa problematica si presenta anche nel momento in cui , nella partituta le indicazioni di tempo subiscono delle variazione. Per esempio da 4/4 a 3/4. Ciò sta a significare che il prototipo realizzato svolge le sue piene funzioni solo in presenza dei seguenti requisiti:

1. XML (formato IEEE 1599) contenente l'attributo vtu_amount
2. Un solo vtu_amount per l'intera partitura
3. Stessa divisione metrica per tutta la durata della partitura

E' lasciata aperta la possibilità di studiare soluzioni che permettano di gestire le situazioni dei punti 2 e 3 sopra indicate.

6.2 CONSIDERAZIONI FINALI E SVILUPPI FUTURI

E' stato piuttosto semplice trattare ed estrapolare i dati del file XML, mentre è stato più complesso elaborare un algoritmo capace di calcolare il NumUnità di ogni singolo evento facente parte di ogni strumento, tenendo conto dei fattori che lo influenzano.

Il lavoro risente delle limitate capacità grafiche di Windows Form (Visual Studio) e in particolare all'utilizzo della Picture Box. La limitazione è dovuta al fatto che essa non è nato come piattaforma specifica per la grafica. Implementare il visualizzatore grafico di accordi con aspetti grafici più articolati, renderebbe il risultato del prototipo esteticamente più interessante.

Il prototipo elaborato, necessiterebbe, oltre alle implementazione sopra dette, della riproduzione simultanea audio-grafica del pezzo musicale.

La sincronizzazione audio-grafica è di possibile realizzazione perché il formato IEEE 1599 presenta un layer specifico, il layer Audio, il cui compito fondamentale è la localizzazione temporale in un file audio, di ogni evento musicale.

Fonti Bibliografiche

- Manuale MX, Ludovico, 2005 [1]
- Key concepts of the IEEE 1599 Standard, Ludovico L.A., IEEE CS, 2008
- Visual C# 2010, Sharp, Microsft Press, 2010 [2]
- C# 4, Bocchino, Hoepli Editore, 2012
- Programming Microsoft® LINQ, Paolo Pialorsi, Marco Russo, Microsoft Press, 2008 [3]
- Armonia, Piston Walter, EDT, 1989 [4]
- Tecnologie basate su XML per la fruizione avanzata dei contenuti musicali, Baratè, A., Ludovico, L.A., Mauro, D.A., Associazione Consortium GARR, Neaples (2010)
- An IEEE 1599-Based Interface for Score Analysis, Baratè, A., Ludovico, L.A., Pinto, A., Springer-Verlag, 2009