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

UNIVERSITÀ DEGLI STUDI DI MILANO

UN PROTOTIPO SOFTWARE PER LA VISUALIZZAZIONE DELLE POSIZIONI DI ACCORDI SU CHITARRA

Relatore: Dott. Luca Andrea Ludovico

Correlatore: Dott. Adriano Baratè

Elaborato finale di:

Marco Banzato

Matricola 740975

Anno Accademico 2010/2011

SOMMARIO

Introduzione

La motivazione che mi ha spinto alla realizzazione dell'opera proposta è principalmente l'amore nutrito nei confronti della musica che ha il potere di essere arte ma allo stesso tempo scienza.

Il mio interesse nei confronti di questa disciplina nasce svariati anni fa, ancor prima di intraprendere questo corso di laurea che ha permesso di ampliare notevolmente la sfera delle mie conoscenze, fino ad allora prettamente legate alla parte armonica ed esecutiva della disciplina. Pur avendo studiato molti anni pianoforte, ho recentemente intrapreso lo studio di un altro strumento: La chitarra.
Applicandomi con costanza e dedizione ho scoperto timbro, sfumature e caratteristiche di uno strumento che ha da sempre incuriosito la mia creatività.

Per l'appunto essendo relativamente nuovo allo studio pratico della chitarra, sono sensibile alle prime problematiche che si presentano a chi si accosta a questo strumento. Il voler imparare a suonare le proprie canzoni preferite rappresenta un momento di svago nel corso dello studio vero e proprio di scale, modi ed esercizi tecnici. Tuttavia dietro questa parvenza di svago si cela l'apprendimento e la memorizzazione di accordi e melodie.

Le motivazioni sopra citate hanno portato alla realizzazione di questo elaborato. Esso si pone infatti l'obiettivo di visualizzare in tempo reale gli accordi di una partitura di chitarra, esportata nel formato XML IEEE 1599, su manico di chitarra.

Lo studio si è incentrato prevalentemente sull'approfondimento delle mie conoscenze del linguaggio di programmazione C# e il mondo .NET che gli gravita attorno.

Inoltre è stata dedicata particolare attenzione all'analisi del formato XML musicale dettato dallo standard IEEE 1599 sviluppato dal Laboratorio di Informatica Musicale (LIM) dell'Università degli Studi di Milano.
L'elaborato è introdotto da un capitolo dedicato agli strumenti a corda con particolare attenzione alla chitarra. Si procederà poi alla presentazione di XML, linguaggi di markup e dello standard IEEE 1599 per poi addentrarsi nel vivo di questo elaborato con la presentazione del software che ho realizzato.

Marco Banzato

Gli strumenti a corda

1.1 I Cordofoni

Gli strumenti a corda, comunemente detti cordofoni, sono una famiglia di strumenti musicali che hanno come caratteristica principale il generare suono tramite la vibrazione delle corde, intonate ad altezze determinate, di cui sono dotati.
Sono tra i più antichi tipi di strumenti inventati dall'uomo, alcuni studiosi sostengono che l'origine degli strumenti a corda risieda nell'arco da caccia e secondo questa teoria l'uomo preistorico avrebbe notato il suono prodotto dalla corda dell'arco nel momento in cui veniva scagliata la freccia e avrebbe voluto riprodurre questo suono che gli sembrava gradevole anche in contesti differenti dalla caccia.
Per l'appunto i primissimi esperimenti sulla creazione di strumenti a corda derivano dall'invenzione del cosiddetto "arco terrestre", un banale arco da caccia ricavato da un bastone elastico al quale veniva tesa una corda, dotato in un secondo tempo anche di una cassa armonica, ottenuta da zucche intagliate, noci di cocco e così via. Non c'è da stupirsi riguardo alla diffusione degli strumenti a corda tra i popoli antichi, in quanto strumenti di facile realizzazione, erano molto diffusi sin dall'epoca degli Egizi, dei Greci e dei Romani che erano soliti usare per la maggiore arpe, cetre e lire.
La famiglia dei cordofoni comprende molti strumenti che possono essere catalogati in:

- Strumenti a corde strofinate, meglio noti come strumenti ad arco a partire dal violino fino ad arrivare al contrabbasso.
- Strumenti a corde pizzicate quali banjo, chitarra, lira.
- Strumenti a corde percosse dal cembalo al pianoforte.

Negli strumenti ad arco il suono è generato dallo sfregamento o dalla percussione dell'archetto sulle corde, possono inoltre essere suonati pizzicando le corde proprio come gli strumenti a corde pizzicate; la generazione del suono in questi ultimi avviene pizzicando le corde con un

plettro o con le dita, gli strumenti a corde percosse usufruiscono del principio della vibrazione delle corde effettuata tramite un martelletto generalmente azionato da una tastiera, in alcuni strumenti come ad esempio il pianoforte è presente un dispositivo che solleva il martelletto dopo la percussione per liberare la corda e permettergli così di vibrare liberamente.

1.2 La chitarra

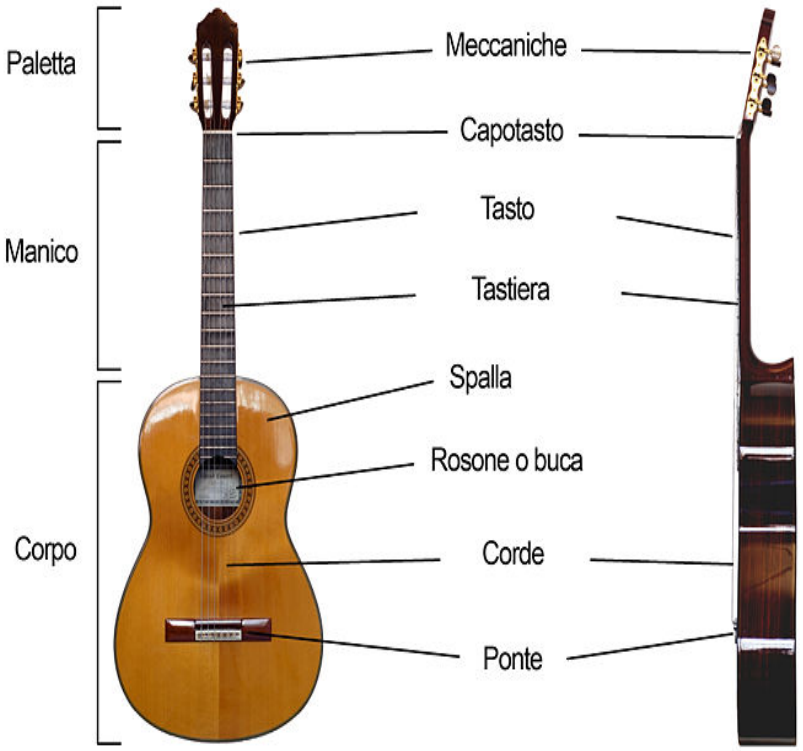

Figura 1.1:Schema tecnico della chitarra

La Chitarra, come anticipato nel capitolo precedente è uno strumento musicale facente parte della famiglia dei cordofoni.

Si può suonare con i polpastrelli, con le unghie o con un plettro.

Una prima divisione tra le chitarre riguarda il diapason ovvero la distanza tra capotasto e ponticello vi sono infatti chitarre dette ¾, ¼ , baritono, tenore etc.

Una seconda divisione può essere effettuata secondo il modo in cui viene amplificato il suono delle corde vibranti ed in particolare siamo in presenza di:

- Chitarre acustiche quando è prevista un'amplificazione che sfrutta le naturali proprietà della fisica del suono, è possibile definire questo tipo di amplificazione meccanico. La sua sonorità dipende principalmente dalla tavola frontale, il fondo da un apporto più ridotto ma produce un grosso contributo nella formazione delle risonanze di bassa frequenza, la meccanica del ponticello e l'urto con la corda contribuisce in maniera sostanziale alla marcatura timbrica dello strumento riconosciuto per avere una voce morbida e capace di diffondersi anche in ambienti di grandi dimensioni.
- Chitarre elettriche hanno un corpo pieno e solido, a differenza delle chitarre acustiche il suono è prodotto per interazione elettromagnetica tra le corde metalliche e piccoli rocchetti di avvolgimento elettrici posti nelle vicinanze del ponticello, questo sistema permette con l'ausilio di un amplificatore o di cassa acustica di rendere il suono chiaro e udibile in ambienti molto affollati.

La chitarra che conosciamo oggi non è molto differente dalla chitarra barocca, suo progenitore del XVI secolo, lo strumento si presenta con una meccanica formata da sei corde tese sopra una cassa armonica dotata di un foro. Come si nota dalla figura 1.1, è possibile dividere la chitarra in tre sezioni principali:

- Paletta contenente le meccaniche, alle corde di essere tese o allentate mantenendo l'accordatura.
- Il manico costituito da una serie di traversine metalliche che delimitano il tasto secondo una spaziatura legata alla distanza del ponticello. In pratica passando da un tasto a quello successivo più vicino al ponticello si ha un aumento di frequenza di un semitono. Questo sistema crea una vera e propria tastiera, affinchè quando viene premuta una corda sul manico con la mano sinistra si accorcia la lunghezza della parte di corda vibrante e ciò permette di suonare la nota desiderata.

- Il corpo costituisce il cuore di questo strumento , la corda di per sé non può produrre sufficiente energia per uso musicale senza il contributo della cassa armonica. Essa è generalmente formata da legno di acero per la tavola superiore e legno duro, mogano, rosewood, acero, ebano, cedro per le altre parti dello strumento.

Il suono è generato dalla vibrazione delle corde tese sopra il piano armonico, il quale poggia sulla cassa armonica che permette di amplificare il suono dello strumento, le corde sono tese tra il tira-corde fissato sul ponticello ed il capotasto.

Linguaggi di mark-up

2.1 Concetto di mark-up language e SGML

La parola mark-up o marcatura è stata storicamente utilizzata in ambiente tipografico dove si usava marcare con annotazioni le parti di testo che andavano evidenziate o corrette, al fine di segnalarle al compositore o al dattilografo. Con l'automatizzazione dei processi di formattazione e stampa di testi, il termine è stato esteso a tutti i tipi di simboli relativi a formattazione, stampa ed elaborazione del testo elettronico. Qualsiasi documento può essere considerato come costituito da contenuto e sistema di contrassegno. Dove per contenuto si intendono i caratteri di un testo, le immagini e quant'altro veicoli il messaggio del documento. Il sistema di contrassegno ha come obiettivo comunicare delle informazioni aggiuntive che vanno ad arricchire il contenuto del documento e si possono dividere nelle categorie di struttura e formazione dove la struttura va a definire una ripartizione logica dei contenuti del documento ad esempio capitoli, paragrafi, sottoparagrafi…, mentre la formattazione si occupa dell'aspetto come ad esempio il tipo di font, il corpo del carattere. Possiamo riassumere schematicamente quanto sopra citato:

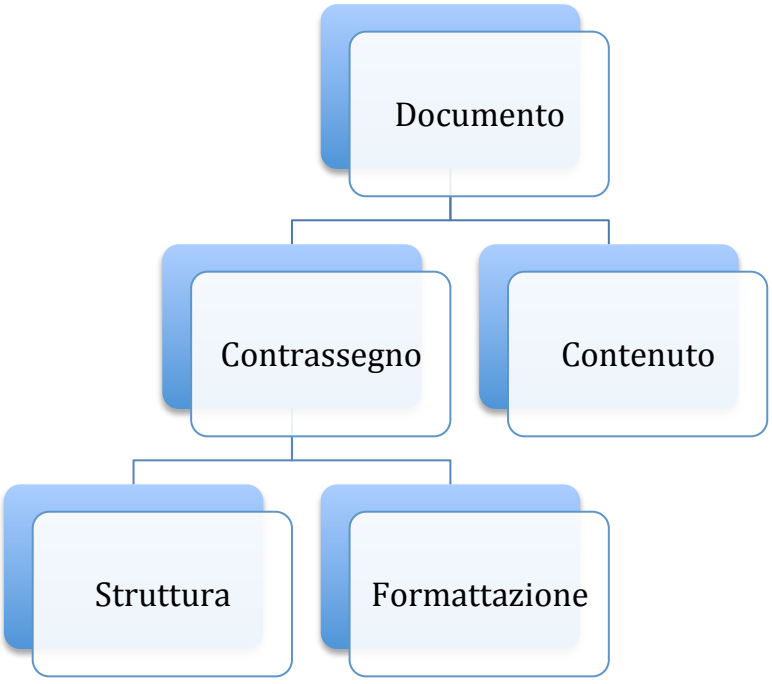

Figura 1.2

Il contrassegno procedurale si occupa degli aspetti di formattazione, il suo vantaggio sta nella separazione della presentazione dal contenuto. L'altra forma di contrassegno è quella descrittiva e si pone l'obiettivo di esplicitare la struttura del documento. Un documento marcato in questo modo risulta fortemente strutturato perché le informazioni sulla struttura sono poste in evidenza.
Per linguaggio di marcatura (mark-up language) si intende una codifica che fa uso di un insieme di marcatori (mark-up) convenzionali, che aderiscono a regole chiare, definite e comunemente accettate.

Un linguaggio di marcatura deve specificare:
- Qual'è il significato di ogni mark-up.
- Come si distinguono i mark-up dal testo.
- Quali mark-up sono consentiti.
- Quali mark-up sono richiesti.

L'SGML (Standard Generalized Markup Language) ad esempio fornisce gli strumenti necessari per rappresentare i primi tre punti. L'SGML è in particolare uno standard internazionale ISO (ISO 8879:1986 SGML) per la descrizione di testi elettronici di tipo mark-up, non è un semplice linguaggio ma ci troviamo davanti a un meta-linguaggio, più precisamente un linguaggio per la descrizione formale di linguaggi. Ha infatti lo scopo di definire linguaggi da utilizzare per la stesura di testi destinati ad essere trasmessi ed archiviati con strumenti informatici, ossia per la stesura di documenti in forma leggibile da computer (machine readable form).
Dal punto di vista di SGML, un documento è rappresentato come una struttura gerarchica di elementi annidati, tale linguaggio non ha i mezzi per specificare aspetti di presentazione di tali elementi, e parallelamente non è in grado di fornire informazioni sul ruolo di alcun elemento. In SGML, l'unica informazione che abbiamo sulla natura di un elemento è relativa a quali contesti e quali livelli di gerarchia esso possa o debba apparire. Tutti i documenti che hanno la stesa gerarchia di elementi sono considerati dello stesso tipo e la struttura di un particolare tipo avviene tramite la definizione del tipo di documento detta anche DTD (Document Type Definition).

2.2 XML

L'XML (eXtensible Markup Language) costituisce uno standard approvato dal World Wide Web Consortium (W3C), un'associazione con lo scopo di migliorare gli esistenti protocolli e linguaggi per il Web e aiutarlo a sviluppare tutte le sue potenzialità.

XML oltre ad essere un sottoinsieme di SGML ne rappresenta anche una semplificazione in quanto:

- Trascura molte opzioni sintattiche e varianti.
- Trascura alcune delle caratteristiche del DTD.
- Trascura alcune caratteristiche problematiche.

Andiamo ad approfondire i suoi vantaggi, l'XML è interscambiabile in rete ovvero permette di condividere e comunicare agevolmente l'informazione codificata in ambienti distribuiti ed in particolare su internet, è indipendente dalla piattaforma, ha struttura gerarchica ovvero una particolare disposizione a rappresentare informazione strutturata secondo schemi gerarchici, ha facilità di lettura dell'informazione codificata questa caratteristica prende il nome di intelliggibilità, inoltre è predisposto ad aggiunte, integrazioni e modifiche, da operare qualora il linguaggio si mostrasse carente o inadeguato agli scopi, ed una grossa disponibilità di tools per l'implementazione del formato e per la validazione dei file prodotti.

Ognuno degli aspetti sopra citati trova la sua applicazione nel campo musicale nel quale l'informazione è fortemente strutturata e gerarchizzata.

2.3 Gli Elementi

Gli elementi sono i mattoni costitutivi dell'XML poiché consentono di attribuire un significato preciso ad una parte di documento.
Ogni elemento è rappresentato da un contrassegno (mark-up) detto anche etichetta (tag). Esistono sostanzialmente due tipi di elementi:

- Gli elementi semplici non possono contenere altri elementi e avere attributi, essi possono contenere solo testo.
- Gli elementi complessi possono contenere testo, altri elementi ed attributi in qualsiasi combinazione.

La sintassi dei tag per la prima casistica è la seguente:

```
<element_name>…</element_name>
```

Esempio 2.1

Dove element_name rappresenta una stringa alfanumerica arbitraria, il singolo elemento viene quindi delimitato da una coppia di etichette, dette rispettivamente etichetta di apertura (start tag) ed etichetta di chiusura (end tag). La sintassi dei tag per la seconda casistica è la seguente:

```
<element0_name>
              <element1_name>…</element1_name>
               <element2_name>…</element2_name>
</element0_name>
```

Esempio 2.2

Come si può notare dall'esempio 2.2 i linguaggi di mark-up consentono l'annidamento di elementi a qualsiasi livello, ovvero ogni elemento può potenzialmente contenerne altri, detti per questo motivo sottoelementi (sub-elements). Andiamo a vedere un esempio.

```
<automobile>
            <marca>Fiat</marca>
            <modello>Punto</modello>
            <colore>Nero</colore>
</automobile>
```

Esempio 2.3

La semplicità di questo linguaggio ha permesso di poter annidare all'interno del tag automobile dei sottoelementi, tra loro differenti come marca, modello e colore.
Tra i diversi elementi, ne troviamo uno che ricopre una particolare importanza, si tratta dell'elemento document, detto anche etichetta radice (root tag). L'etichetta radice è l'elemento più esterno, di conseguenza è quello che racchiude tutti gli altri elementi del documento, pertanto deve esistere sempre.
Meritano tuttavia un'attenzione particolare una serie di elementi detti empty-tag, la loro caratteristica è di non presentare l'end-tag e parallelamente nessun contenuto. Non presentando l'end-tag il loro start-tag avrà una forma particolare:

```
<element_name…/>
```

Esempio 2.4

Dove i puntini di sospensione rappresentano un segnaposto per qualcos'altro, gli attributi, che andremo ad analizzare in seguito, mentre notiamo che il carattere "/" si trova in una posizione anomala. Dal punto di vista sintattico si può immaginare che i tag di inizio e fine collassino in un'unica etichetta.

2.4 Gli Attributi

Gli attributi vengono utilizzati per aggiungere informazione ad un elemento e sono sempre associati allo start-tag:

```
<element_name attribute_1=”value_1”…attribute_N=”value_N”>
…
</element_name>
```

Esempio 2.5

Un elemento può avere un numero qualsiasi di attributi distinti, come si può constatare dall’esempio 2.6.

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

Esempio 2.6

`<ricetta>` rappresenta il tag radice nel quale sono annidati gli altri elementi. Quando segue il nome dei tag all’interno delle parentesi angolari rappresenta un attributo. Dunque quantità 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.

2.5 Confronto fra Elementi e Attributi

Il problema di utilizzare elementi o piuttosto attributi per rappresentare l'informazione è sottile, e comunque non riguarda chi genera un file XML ma piuttosto 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. Nella progettazione del DTD viene usato un elemento quando l'informazione deve essere accessibile in modo veloce, l'informazione deve essere visibile a tutti, l'informazione è importante per il significato del documento, e l'informazione è debolmente tipata. Si predilige un attributo quando esso rappresenta una scelta tra un numero finito di valori, quando è visibile solo per il sistema, quando l'informazione non è di fondamentale importanza per la struttura del documento, e quando l'informazione è fortemente tipata. Tra attributi esiste inoltre un rapporto gerarchico di parità, se quindi ad 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. Notiamo che <cottura> è un elemento vuoto perché non esiste una caratteristica di cottura più importante delle altre, non sarebbe comprensibile scrivere:

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

Esempio 2.7

Accettare in un DTD una scrittura del genere significherebbe 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.

Inoltre in XML esistono altri "oggetti" tra cui:

- Processing instructions, usate per propositi di estensibilità e denotate con `<?target data>`.
- Commenti, racchiusi sintatticamente dai caratteri `<!--` e →.
- Riferimenti a caratteri, quali `&#163`.
- Entità: file esterni o parti del documento cui si può fare riferimento ricorsivamente o da sezioni diverse nel documento stesso.

2.6 DTD, wellformedness & validity

Il DTD fornisce un significato standard per descrivere in modo dichiarativo la struttura di un tipo di documento. In particolare DTD rappresenta un documento che descrive i tag utilizzabili in un documento XML, la loro reciproca relazione nei confronti della struttura del documento e altre informazioni sugli attributi di ciascun tag. Un DTD è logicamente composto da due parti:

- Element Type Definition.
- Attribute List Declaration.

L'element type definition va a specificare la struttura del documento, i contenuti consentiti (content model) e gli attributi consentiti (dal significato delle dichiarazioni delle liste di attributi).

Andiamo ora ad analizzare degli esempi di definizioni nei DTD:

```
<!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*.

Quando la cardinalità non viene specificata, si dice cardinalità di default ed è uguale a uno e solo uno.

Quando la cardinalità è "?" è uguale a zero oppure uno.

Quando la cardinalità è "*" è uguale a zero o più.

Quando la cardinalità è "+" è uguale a uno o più.

```
<!ELEMENT A (B | C+)>
```

Dove l'operatore "|" significa *choice of* ovvero `A` contiene o l'elemento `B` o l'elemento `C`.

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

Si tratta di un elemento di testo.

```
<!ELEMENT A EMPTY>
```

Si tratta di un elemento vuoto.

```
<!ELEMENT A (#PCDATA| B | C)*>
```

Si tratta di un elemento misto.
L'*Attribute List Declaration* è la lista degli attributi permessi per ogni elemento. Ogni attributo è specificato da: nome, tipo e altre informazioni.
I tipi di attributi appartengono a tre gruppi:

- String types (CDATA).
- Tokenized types (ID, IDREF, IDREFS, …).
- Enumerated types (come in Pascal).

Come abbiamo fatto in precedenza per la definizione del DTD andiamo ad analizzare qualche esempio di dichiarazioni nei DTD (attributi dell'elemento ELEM):

```
<ATTLIST ELEM attrib 1 CDATA #IMPLIED>
```

`attrib 1` è di tipo testuale e non è richiesto.

```
<ATTLIST ELEM attrib 1 CDATA #IMPLIED attrib 2 CDATA #REQIRED>
```

`attrib 1` e `attrib 2` sono di tipo testuale, il primo non è richiesto, il secondo si.

```
<ATTLIST ELEM attrib 1 CDATA #IMPLIED "aaa">
```

`attrib 1` è di tipo testuale, non è richiesto e ha valore di default `"aaa"`, se al posto di `#IMPLIED` avessimo trovato `#REQUIRED` l'attributo sarebbe stato richiesto.

```
<!ATTLIST A a CDATA #FIXED "aaa" >
```

attrib 1 è di tipo testuale e ha valore costante `"aaa"`.

```
<!ATTLIST A a (aaa|bbb) #IMPLIED "aaa" >
```

`attrib 1` è di tipo enumerato, con i valori elencati tra parentesi, e ha valore di default `"aaa"`.

```
<!ATTLIST A id ID #REQUIRED >
```

`attrib 1` è di tipo `ID` (assegnazione di valore univoco)

```
<!ATTLIST A ref IDREF #IMPLIED >
```

`attrib 1` deve essere un riferimento a un `ID`.

Secondo le specifiche W3C i parser non tollerano errori, quindi possono fermarsi al primo errore che incontrano all'interno del file XML. Questa scelta è stata fatta per rendere i parser semplici da implementare e per la natura stessa di XML (compatibilità e indipendenza dalla piattaforma e dal parser). Gli errori che i parser possono incontrare durante l'analisi sono:

- Fatal error.
- Error.

Dove una violazione di un vincolo espresso in un DTD viene considerata "ERROR", mentre una violazione di una regola di "buona formattazione" viene considerata un "FATAL ERROR", secondo le specifiche un fatal error è più grave di un semplice error, se si verifica un fatal error, il parser può continuare l'analisi del documento per cercare altri errori ma non deve continuare a fornire il contenuto del documento all'applicazione.

Al presentarsi di un error il parser può continuare l'analisi del documento per cercare altri errori e può continuare a fornire il contenuto del documento all'applicazione, è compito di quest'ultima decidere cosa fare (ignorarlo, segnalarlo, …). Un documento si dice ben formato (well formed) se segue le regole grammaticali indicate dal W3C, aderisce quindi alle regole sintattiche delle specifiche XML 1.0 e ne soddisfa le strutture fisiche e logiche.
Un documento è valido (valid) se è conforme ad un DTD che ne specifica la struttura, solitamente le specifiche di wellformedness & validity vengono controllate da software specifici che si occupano di validazione e che vanno a controllare il rispetto delle regole sintattiche generali e la conformità con il DTD di riferimento e se un file non rispetta tali proprietà genera uno dei due errori precedentemente presentati da parte del motore di parsing XML.

Lo standard IEEE 1599

3.1 Introduzione

Questo capitolo sarà interamente dedicato alla presentazione dello standard IEEE 1599. Partendo da un'attenta analisi dello stesso, che sarà approfondito nel corso del seguente paragrafo, il software presentato si pone l'obiettivo di visualizzare su manico di chitarra una tra le parti a scelta costituenti un brano, che essa sia melodica costituita dal susseguirsi di note singole o che sia armonica, costituita dal susseguirsi di più note suonate in contemporanea. Questa era l'idea di partenza, tuttavia essendo un software pensato per essere utilizzato non solo da utenti esperti ma da qualsiasi persona mossa dalla voglia di imparare a suonare un brano sulla chitarra, sono state integrate diverse caratteristiche che permettono all'applicazione di essere più completa e di facile utilizzo, quali:

- Possibilità di aumentare o diminuire la velocità di rappresentazione delle note o degli accordi sul manico di chitarra.
- Possibilità di settare un valore BPM a piacimento, in alternativa utilizzare il valore indicato come default (120BPM).
- Una list-box atta a visualizzare in tempo reale i nomi delle note rappresentate.
- Possibilità di andare a modificare l'accordatura di ogni singola corda della chitarra.
- Una barra decrescente che determina la durata dell'accordo o della nota affinchè l'utente possa rendersi conto se deve o meno risuonare la stessa nota/accordo.
- Un metronomo atto alla scansione delle battute.
- Possibilità di visualizzare l'accordo o la nota successiva durante la rappresentazione della nota attuale.
- Grazie all'utilizzo di tre controlli grafici (play, pause, stop) di far partire, mettere in pausa o fermare la rappresentazione del brano nel momento scelto dall'utente.

Oggigiorno sono presenti sul mercato innumerevoli software di questo tipo, tuttavia il prototipo che è stato realizzato si propone di essere in primo luogo di facile utilizzo e in secondo luogo molto leggero e quindi richiedere l'impiego limitato di risorse da parte del sistema operativo.

3.2 Lo Standard IEEE1599: Definizione e cenni storici

Il software realizzato è fortemente legato allo standard IEEE1599, in quanto il file che viene dato in pasto al programma è un file di tipo XML IEEE 1599, risulta quindi essenziale presentare lo standard e le sue principali caratteristiche.

IEEE è l'acronimo di Institute of Electrical and Electronic Engineers ed è un'associazione internazionale di scienziati professionisti con l'obiettivo della promozione delle scienze tecnologiche. L'IEEE 1599, nome ufficiale dell'MX (Musical application using XML) è un nuovo formato basato sull'XML il cui scopo è quello di descrivere una serie eterogenea di contenuti musicali, quali notazione musicale, partiture a stampa, tracce audio, metadati di catalogo, testo e contenuti grafici tutti inglobati e relazionati tra loro all'interno di un file unico.

Contenuti eterogenei organizzati in una struttura multi-strato che supporta diversi formati di codifica e un numero di oggetti digitali per ogni strato.

Questo standard è la conseguenza di numerosi anni di studi e di ricerca da parte del LIM (Laboratorio di Informatica Musicale) con base operativa all'Università degli Studi di Milano. Una prima fase del progetto è iniziata nel 2000 con la valutazione delle caratteristiche dei formati esistenti per la descrizione di contenuti musicali e in modo particolare SMDL, NIFF ed XML. Il progetto di standardizzazione IEEE nasce nel 2001, e parallelamente vengono a crearsi una serie di contatti con altri gruppi di ricerca sia in ambito accademico che in ambito commerciale.

Nel 2002 viene rilasciata una versione prototipo del formato, originariamente chiamato "Musical Application using XML" o più semplicemente MAX.

Il formato viene discusso a MAX 2002, la prima conferenza internazionale su applicazioni che usano l'XML, organizzata da IEEE CS TC. Il processo finale di valutazione di IEEE, detto ballottaggio si concluse nel Luglio del 2008 con il risultato di rendere MX/P1599 uno standard internazionale.

3.3 XML e musica

Come anticipato, l'IEEE 1599 è un formato basato su XML, ci sono diversi vantaggi nello scegliere l'XML nella descrizione di informazione musicale.
Prima di tutto l'XML è un metalinguaggio formale adatto a rappresentazioni dichiarative, è in grado di descrivere entità, come ad esempio oggetti musicali come sono, senza sovraccaricarli di informazioni inutili. La rappresentazione risulta modulare e gerarchica, tuttavia consente l'interazione tra i vari moduli in maniera formale ed esplicita, tutti questi aspetti hanno una controparte importante nei linguaggi musicali, i quali sono sia formali che gerarchici.
La modularità dell'XML offre diversi vantaggi per una descrizione comprensibile e ben organizzata. Ogni parte costituisce un'entità separata della descrizione complessiva pur mantenendo la propria identità; l'interdipendenza è consentita e resa esplicita da parte del formato basato sull'XML. I linguaggi derivati dall'XML sono estensibili, in quanto supportano estensioni ed elementi esterni. Questo rappresenta un aspetto fondamentale per i futuri sviluppi del linguaggio musicale ed essendo XML un linguaggio aperto a contributi da parte degli utenti, facile da leggere, decodificare, modificare e capire, chiunque può dare suggerimenti e implementare parti specifiche del formato. Anche se l'XML è molto più facile da usare rispetto a formati binari, non significa che possa essere gestito direttamente, ma può essere scritto e letto con sistemi computer-based.

Per esempio il Music XML non può essere stampato nella sua forma testuale al fine di essere suonato da un esecutore umano, è necessario un software specifico per decodificare il formato e rappresentarlo secondo una sequenza di simboli musicali, interpretabili dall'utente. Fortunatamente le applicazioni per modificare i file XML possono essere facilmente reperibili sul mercato e spesso degli editor di base sono gratuiti.

3.4 Una struttura Multi-Layer

Come detto in precedenza, una descrizione esauriente dell'informazione musicale deve supportare materiali eterogenei. Grazie alla capacità propria dell'XML, le rappresentazioni dell'informazione musicale possono essere organizzate in maniera efficace ed efficiente.
IEEE1599 impiega sei differenti livelli per rappresentare l'informazione musicale, si tratta quindi di una struttura Multi-Layer, composta da sei strati detti (layers):

- General, metadati relativi alla musica, ad esempio il catalogo di informazioni relative al prezzo.
- Logic, contiene la funzione di mappatura spazio temporale è diviso in tre sottolivelli: Spine, Los e Layout.
- Structural, identifica gli oggetti musicali e le loro relazioni reciproche.
- Notational, contiene la rappresentazione grafica dello score.
- Performance, descrizioni computer-based ed esecuzione di musica.
- Audio, rappresenta il livello più basso della struttura IEEE 1599 e si occupa di descrivere le proprietà del materiale musicale audio.

Andiamo a vedere graficamente questa struttura nella figura 3.1.

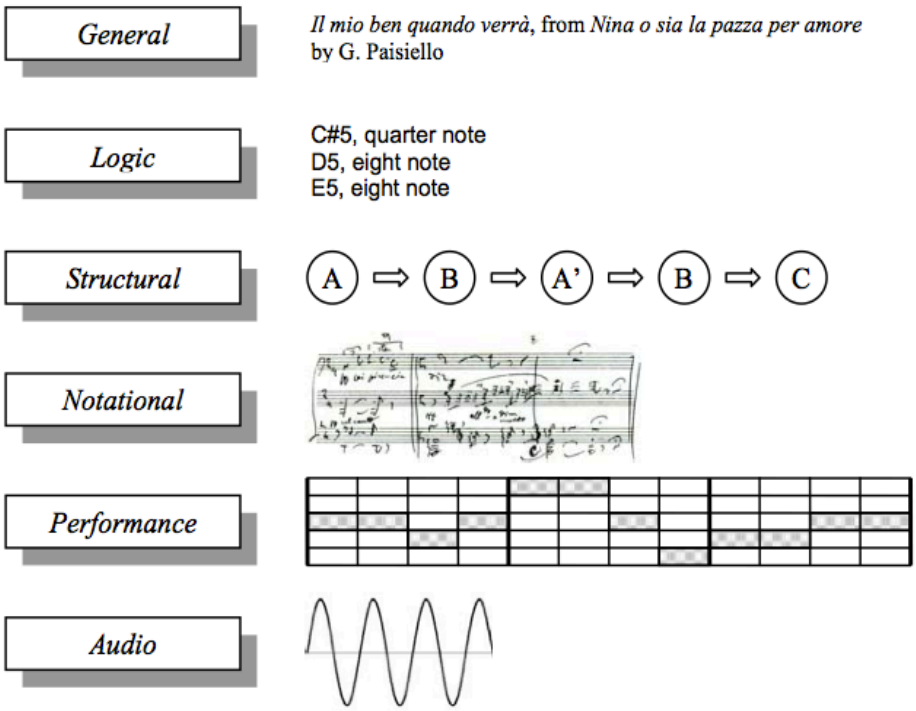

Figura 3.1

L'esempio 3.1 presenta un generico documento XML rappresentante la struttura Multi-Layer di IEEE1599:

```
<?xml version="1.0" encoding="UTF=8"?>
<!DOCTYPE ieee 1599 SYSTEM
      "http://www.mx.dico.unimi.it/ieee1599.dtd">
<ieee1599>
       <general>...</general>
       <logic>...</logic>
       <structural>...</structural>
       <notational>...</notational>
       <performance>...</performance>
       <audio>...</audio>
</ieee1599>
```

Esempio 3.1

Non è necessario che per un brano musicale siano presenti tutti i layer, ovviamente più layer saranno presenti nel documento più ricca sarà l'informazione musicale presentata.
La filosofia dello standard IEEE1599 permette inoltre anche la possibilità di contenere in ogni layer più di un'istanza digitale, per esempio il layer Audio può essere collegato a diverse tracce audio e il layer Structural può provvedere analisi differenti per lo stesso pezzo.

Il prototipo software

4.1 Strumenti utilizzati

Per sviluppare questo progetto è stato scelto C# come linguaggio di programmazione; linguaggio object-oriented sviluppato da Microsoft all'interno dell'iniziativa .NET, è quello che meglio descrive le linee guida sulle quali ogni programma .NET gira, questo linguaggio è infatti stato creato specificatamente per la programmazione nel Framework .NET partendo da concetti derivanti da C, C++ e Java. Un altro software utilizzato nella realizzazione di questo programma è Finale, eccellente editor di partitura che ha permesso di creare le notazioni dei brani e tramite un plug-in di poter convertire ed esportare queste partiture in un file IEEE 1599.

4.2 Interfaccia Grafica

Prima di addentrarsi nel cuore dell'applicazione e trattarne gli algoritmi è opportuno presentare l'interfaccia grafica e le sue potenzialità. L'immagine seguente (Figura 4.1) raffigura la finestra visualizzata al momento dell'apertura dell'applicazione.

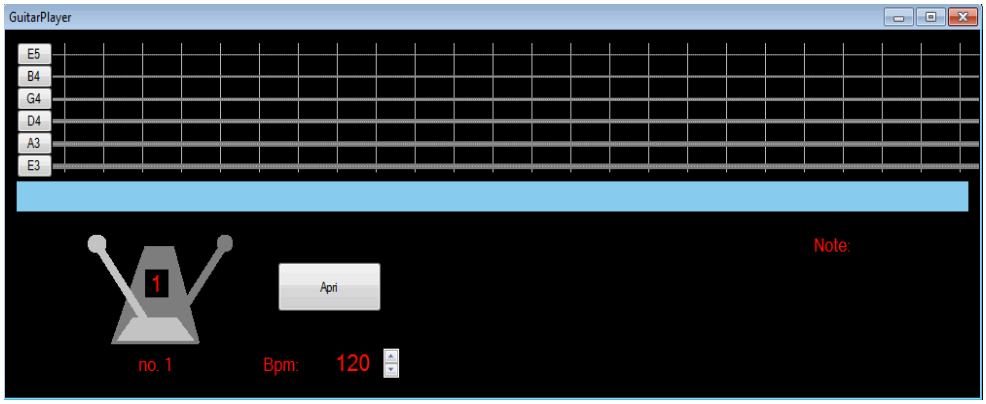

Figura 4.1

L'elemento principale di questo programma è il manico della chitarra sul quale verranno posizionati dall'algoritmo gli accordi o le note costituenti un brano musicale con l'ausilio di pallini rossi.

Si è voluto dare all'utente la possibilità di poter cambiare l'accordatura di ogni singola corda, rendendo così il software versatile e dinamico.
Come si può osservare dalla figura 4.2, andando a cliccare su uno qualsiasi dei sei bottoni contenenti la nota di accordatura sarà visualizzato il Form `notadialog` che tramite l'impiego di tre combobox, la prima dedicata alla nota, la seconda dedicata all'alterazione e la terza all'ottava di riferimento, permetterà di andare a modificare i valori corrispondenti, cambiando così l'accordatura dello strumento virtuale. In questo modo, le note saranno posizionate sul manico in tasti differenti, tenendo conto dell'accordatura scelta dall'utente.

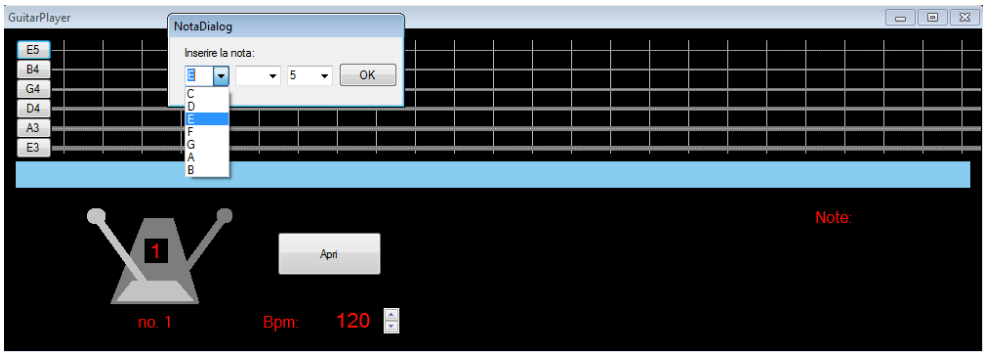

Figura 4.2

È stato implementato un metronomo che va a scandire gli accenti e a dare un riferimento di metro durante la riproduzione del brano. Non tutti i brani hanno lo stesso metro, si è quindi trovata una soluzione per far rappresentare dal metronomo virtuale questa informazione ritmica in modo corretto. In particolare all'interno del file IEEE 1599 nel layer LOS, l'informazione ritmica si trova racchiusa all'interno del tag `time_indication`, il software andrà a memorizzarsi questo dato e parallelamente adatterà l'informazione del metro di conseguenza. Un altro elemento grafico è il pulsante Apri, una volta cliccato ci permette di selezionare un file IEEE 1599 a piacimento, dopo aver selezionato il file si aprirà un menù denominato `strumentsdialog` sul quale saranno visualizzate le parti costituenti il brano estrapolate dal file XML (Figura 4.3).

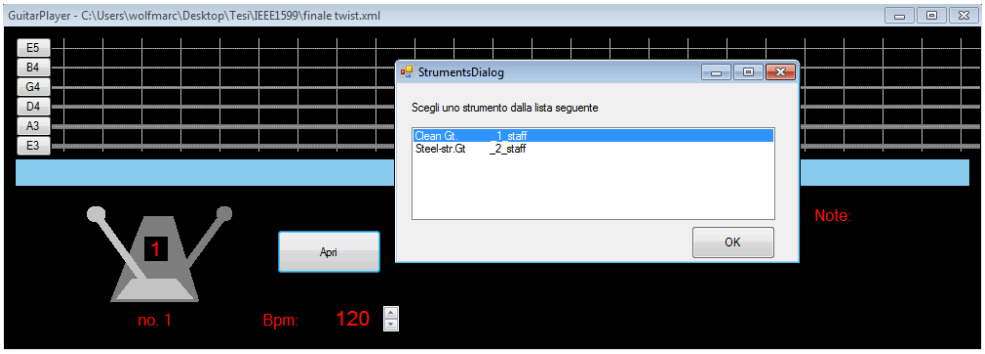

Figura 4.3

Si è deciso di implementare anche un campo relativo alle BPM del brano, dando così la possibilità all'utente di accelerare o diminuire a piacimento la velocità di rappresentazione delle note sul manico dello strumento virtuale.

L'indicazione relativa alle Battute Per Minuto del brano non è presente all'interno del file XML si è quindi deciso di far apparire un menù chiamato `Timedialog` che da la possibilità all'utente di settarsi le BPM a suo piacimento o in alternativa di utilizzare un valore di Default che è stato scelto di porre a 120 (Figura 4.4).
Segue l'immagine relativa all'impostazione delle BPM:

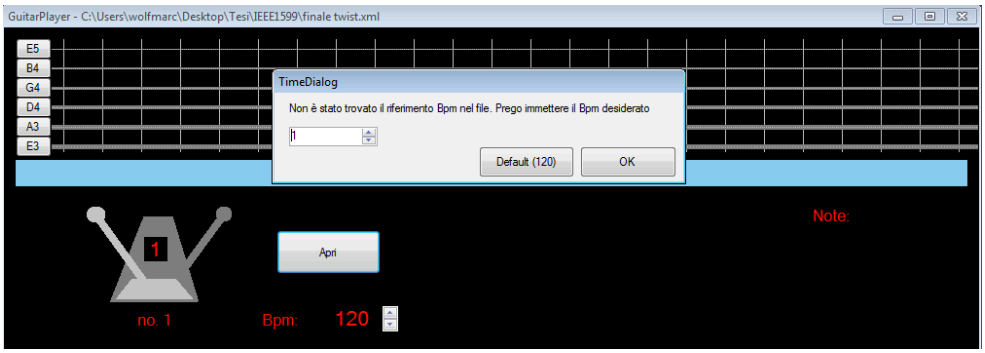

Figura 4.4

Grazie all'uso di una List-Box `(Note)` il programma visualizzerà il nome delle note corrispondenti all'accordo rappresentato. Uno dei problemi riscontrati è stato: come permettere all'utente di rendersi conto quando deve risuonare lo stesso accordo più volte di seguito. In alcuni brani il medesimo accordo può essere suonato più volte di seguito, per ovviare a questo problema si è deciso di implementare una progress-bar decrescente.

Si vada a spiegarne il suo funzionamento, si ha accordo_1 e accordo_2, dove accordo_1 è uguale ad accordo_2, non appena viene visualizzato accordo_1 sul manico della chitarra, la progress-bar inizierà a decrescere per tutta la sua durata temporale, si riempirà di nuovo con l'apparire di accordo_2 e decrescerà di nuovo rispettando il valore di durata di accordo_2. In questo modo l'utente si renderà conto che deve risuonare l'accordo.

Le note, come detto in precedenza sono visualizzate sulla chitarra virtuale tramite dei pallini rossi e in particolare:

- Se il pallino si trova sul capotasto indica che la corda va suonata a vuoto.
- Se il pallino si trova in un determinato tasto va suonata la nota corrispondente a quel tasto.

Si è deciso di implementare un algoritmo che permetta di visualizzare durante la riproduzione di un accordo, tramite l'ausilio di pallini grigi, l'accordo successivo, affinchè l'utente conoscendo l'accordo da suonare in seguito abbia il tempo necessario per cambiare posizione.

4.3 Test dell'applicazione

In questo sotto-capitolo si effettuerà un test dell'applicazione, si selezionerà un file IEEE 1599 che verrà dato in pasto all'applicazione, si andrà a seguire passo per passo il suo funzionamento con l'ausilio di alcune immagini.

Prima di tutto si vada a cliccare sul pulsante Apri, si aprirà una finestra di navigazione che permetterà di selezionare un file XML, per questo test si è scelto di utilizzare il file XML del brano Twist and Shout dei Beatles (Figura 4.5).

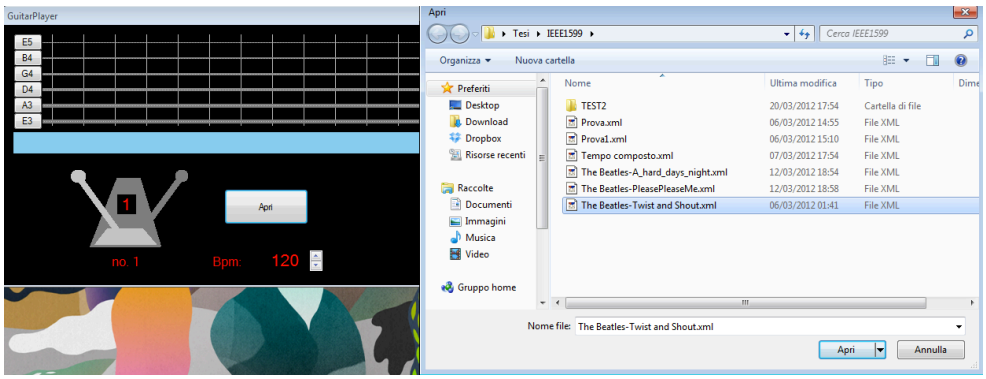

Figura 4.5

Dopo aver selezionato il file XML, il programma andrà ad estrapolare le parti contenute all'interno dello stesso e le visualizzerà all'utente tramite un menù, si potrà così selezionarne una. Si vada a selezionare la prima delle due parti "Clean Gt." (Figura 4.6).

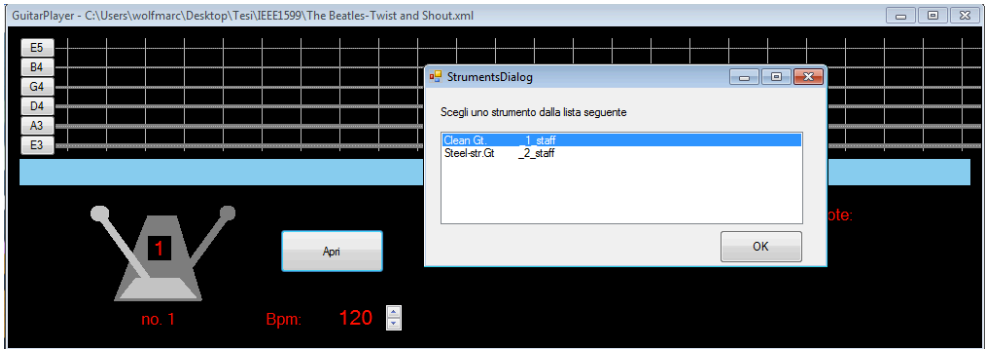

Figura 4.6

Il programma chiederà ora all'utente di inserire un valore BPM, in alternativa, di usare il valore impostato di default, durante questo test si è deciso di settare il valore originale del brano, 128 (Figura 4.7).

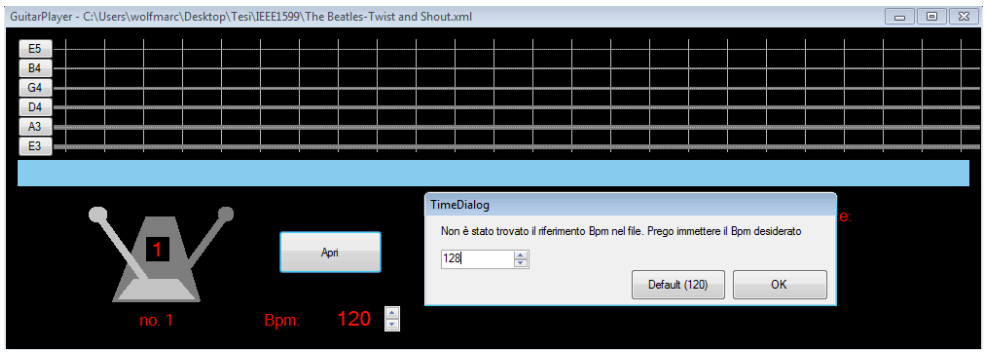

Figura 4.7

A questo punto il software sarà pronto per rappresentare gli accordi del brano sul manico della chitarra virtuale, si vada ora a cliccare sul pulsante Play (Figura 4.8).

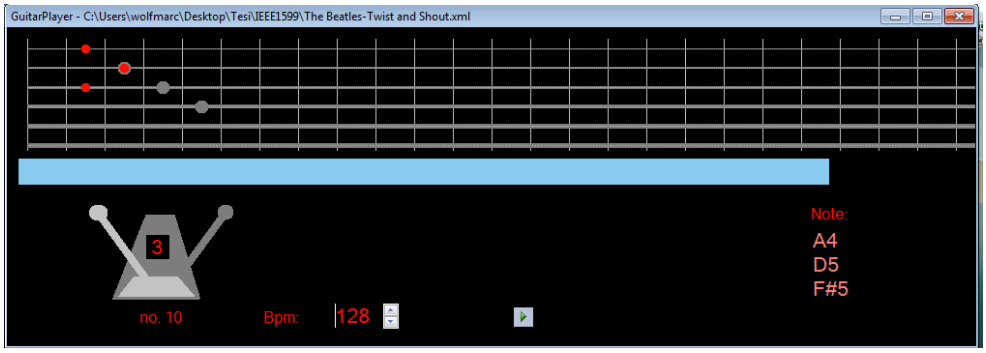

Figura 4.8

La figura 4.8 ritrae il programma in esecuzione, si notino tutte le caratteristiche sopra citate:
la progress-bar decrescente che sta indicando il valore temporale dell'accordo, nel manico la presenza dei pallini rossi che in questo caso indicano di suonare l'accordo di Re maggiore e i due pallini grigi con l'indicazione dell'accordo da suonare successivamente, la list-box (Note) indicante il nome di ciascuna nota dell'accordo, il metronomo che si trova al terzo accento della battuta numero 10, il valore delle BPM che può essere aumentato o diminuito durante l'esecuzione ed il tastino play per far riprendere l'esecuzione del brano.

Algoritmi utilizzati

5.1 Introduzione

Le difficoltà maggiori riscontrate nella realizzazione del progetto sono quelle relative al modo in cui far posizionare le note sul manico dall'algoritmo, una nota può essere rappresentata in sei modi differenti dipendentemente dall'accordatura della chitarra e dalla relativa posizione della stessa sui tasti. Di fondamentale importanza che un accordo sia fisicamente suonabile e che le note si trovino in posizioni raggiungibili dalla mano sinistra, si è creato quindi un algoritmo che permetta di rappresentare sulla chitarra un accordo nel modo più semplice possibile, sfruttando l'ausilio delle corde vuote quando possibile, in modo da generare accordi facilmente suonabili anche da un principiante. Una seconda difficoltà riscontrata è trovare un modo per far vedere all'utente una scansione temporale ovvero come mostrare all'utente che un accordo deve essere ripetuto ritmicamente, da qui nasce l'idea di aggiungere un metronomo e una progress-bar. Tuttavia il primissimo passo da compiere è stato lo studio del formato XML IEEE 1599, degli elementi che lo costituiscono e di quelli che in particolare racchiudono le informazioni di interesse per il programma.

L'analisi è ricaduta all'interno del layer Logic e del suo sottoelemento LOS.

Il Los è composto dai seguenti sotto-elementi:

- `staff_list`
- `part`

Lo `staff_list` contiene informazioni sull'accollatura, ovvero l'elenco degli staff contenute nel suo sottoelemento staff. All'interno dello staff sono presenti due ulteriori sotto-elementi: il `time_signature`, contenente informazioni temporali; al suo interno troviamo la `time_indication` che indica il valore dell'accento e del metro, che sarà utilizzato dal metronomo, ed il `key_signature` rappresentante informazioni sulla chiave della Parte a cui lo `staff_list` si riferisce.

L’esempio 5.1 presenta il contenuto di `staff_list`:

```
<staff_list>
  <staff id=”Clean_Gt._1_staff” line_number=”5”>
    <time_signature event_ref=”TimeSignature_Clean_Gt_1_1>
      <time_indication num=”4” den=”4”/>
    </time_signature>
    <key_signature event_ref=”KeySignature_Clean_Gt_1_1>
      <sharp_num number=”0”/>
    </key_signature>
    <clef shape=”F” event_ref=”Clef_Clean_Gt_1_1”
staff_step=”6” octave_num=”0”/>
  </staff>
</staff_list>
```

Esempio 5.1

L’elemento `part` contiene tutte le informazioni riguardanti le voci della partitura, al suo interno si trova il sottoelemento `measure`, avente attributo number, che presenta tutte le informazioni riguardanti le note musicali esistenti in quella determinata misura. Focalizziamo l’attenzione ora in due elementi fondamentali per il funzionamento del software, si tratta di `chord`, contenente il tag `duration` che va ad indicare la durata della nota, e l’elemento `notehead` contenente il tag `pitch` che possiede tre attributi:

- octave individua l’ottava della nota.
- step viene indicata la nota.
- actual_accidental definisce la possibile alterazione della nota, naturale, diesis o bemolle.

L’esempio 5.2 raffigura il contenuto dell’elemento `part`:

```
<part id=”Clean_Gt._1”>
  <voice_list>
    <voice_item id=”Clean_Gt._1_0_voice”
staff_ref=”Clean_Gt._1_staff/>
  </voice_list>
  <measure number=”1”>
    <voice voice_item_ref=”Clean_Gt._1_0_voice”>
      <chord event_ref=”Clean_Gt._1_voice0_measure1_ev0”>
        <duration num=”1” den=”4”/>
        <notehead>
          <pitch octave=”4” step=”A”
actual_accidental=”natural”/>
        </notehead>
      </chord>
    </voice>
  </measure>
</part>
```

Esempio 5.2

Il nostro scopo è rappresentare su manico di chitarra gli accordi o le note di un brano partendo da un file IEEE 1599, tutti gli altri elementi che costituiscono il nostro file verranno scartati.

5.2 Panoramica sul progetto

Per comprendere a pieno il funzionamento dell’applicazione e per darne una visione generale prima di andare ad affrontare le parti di codice, si è deciso di presentare tutti i livelli mediante un grafico “a livelli” e tutte le classi mediante un diagramma UML.
Nel seguito di questo elaborato verranno citati tre concetti principali che è bene andare a chiarire fin da ora.

Sarà utilizzato il termine *NotaLogica* per indicare una nota musicale ad esempio il MI della terza ottava, il termine *NotaFisica* per indicare le coordinate che rappresentano una nota sul manico della chitarra virtuale, infine *ChitarraLogica* per rappresentare una chitarra con la sua accordatura dove saranno visualizzati fisicamente gli accordi o le note. È importante tenere presente che un insieme di note logiche suonate su diverse chitarre logiche possono dar luogo a diversi insiemi di note fisiche in relazione all'accordatura scelta per la *ChitarraLogica*.

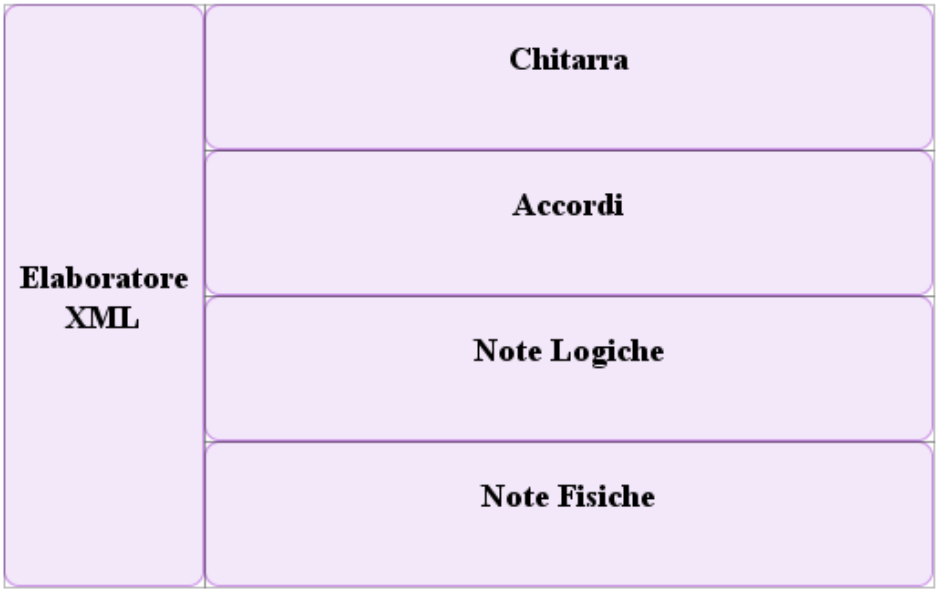

Figura 5.1

Il grafico in figura 5.1 presenta l'ottica in cui è stato concepito il software, si presenterà di seguito una prima idea "grezza" del progetto. La prima operazione da compiere a livello logico è segmentare il problema principale in piccoli sotto-problemi che possano essere gestiti individualmente, il grafico in figura 5.1 presenta proprio questa prima divisione. La primissima cosa da fare è implementare un processo atto a elaborare il file XML dato in ingresso per estrapolarne delle informazioni. Questo compito è affidato all'Elaboratore XML, che lavorando ortogonalmente agli altri livelli, va a destinare ad ognuno di essi le informazioni necessarie.

Il primo livello in cui ci si imbatte si chiama *Chitarra*, esso ha il compito di gestire una lista di Accordi. Il secondo livello è occupato da *Accordi,* che va a gestire le *NoteLogiche*, il terzo livello *NoteLogiche* si occuperà di generare le *NoteFisiche* ovvero le coordinate corrispondenti ai tasti della chitarra virtuale. Questo schema a livelli rappresenta un'idea di partenza, in fase di implementazione non è tuttavia stato seguito in modo ferreo poiché risulta impossibile attenersi ad uno schema così rigido, ogni livello svolgerà quindi il suo compito principale e allo stesso tempo dei sotto-compiti più piccoli.
Il seguente diagramma UML fornisce invece una reale visione della suddivisione in classi del codice.

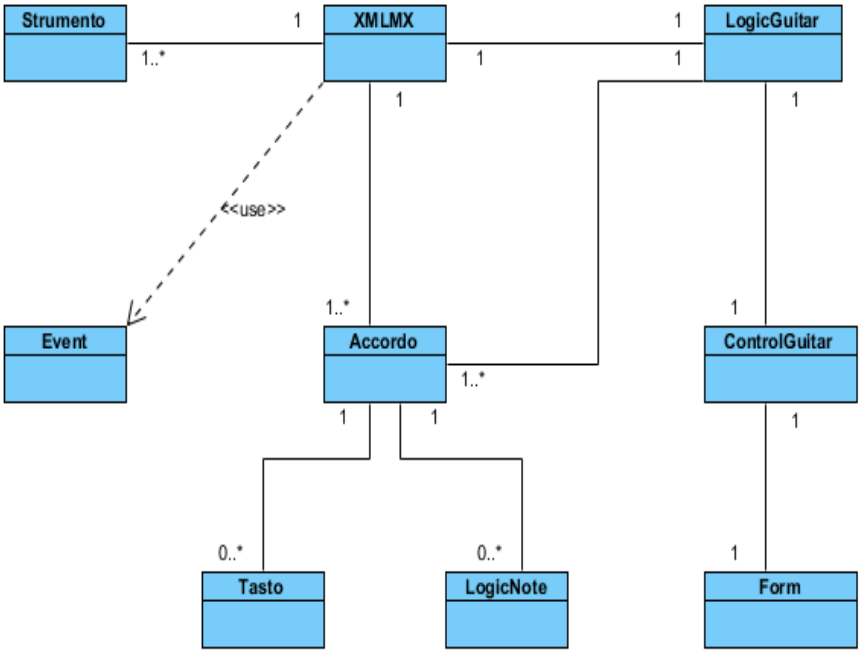

Figura 5.2

In particolare troviamo:

- La classe *XMLMX* che ha il compito di elaborare un file XML, ed estrapolare dallo stesso strumenti, eventi e accordi al fine di generare una lista di accordi utili alla classe *LogicGuitar*.
- La classe *Accordo* che dato un insieme di *LogicNote* ed una *LogicGuitar* permette di estrapolare l'insieme di note fisiche.
- La classe *Tasto* che rappresenta semplicemente la nota fisica.
- La classe *LogicNote* che rappresenta la nota logica ovvero la nota musicale (es: il Mi della terza ottava).
- La classe *Strumento* che contiene il nome della parte, indicata all'interno del file XML e il relativo `time_indication`.
- La classe *Event* che rappresenta l'informazione riguardante un evento definito nel layer Spine
- La classe *ControlGuitar* che si occupa di rappresentare graficamente una chitarra logica istante per istante
- La classe *Form* che contiene un *ControlGuitar* e tramite un timer scandisce il tempo in modo che gli accordi siano suonati in una sequenza corretta da quest'ultimo.

5.3 La classe XMLMX

Dopo aver analizzato attentamente nel sotto-capitolo 5.1 gli elementi di nostro interesse all'interno del file XML IEEE 1599 è necessario utilizzare una tecnologia che ci permetta di estrapolarli per poi farli elaborare dall'applicazione. La soluzione scelta è LINQ (Language-Integrated Query), un set di funzionalità introdotte da Visual Studio 2008 che migliora la gestione delle query nella sintassi dei linguaggi C# e Visual Basic. LINQ introduce modelli standard di facile apprendimento per l'esecuzione di query e l'aggiornamento dei dati e la tecnologia può essere estesa per supportare qualsiasi tipo di archivio dati.

Visual Studio include degli assembly del provider LINQ che permettono di utilizzare LINQ con insiemi di .NET Framework, database SQL Server, DataSet ADO.NET e in particolare documenti XML. Per poter usufruire delle potenzialità di LINQ è necessario includere la direttiva assembly using System.Linq e using System.Xml.Linq.

La classe XMLMX rappresenta una delle parti di codice più corpose , si occupa di ricavare dal file XML la lista degli accordi. Al suo interno troviamo un metodo costruttore a cui passiamo "path", ovvero il percorso del file XML, "path" andrà così a caricare il file selezionato su un XDocument, classe di LINQ atta a rappresentare un documento XML.

All'interno di questa classe sono presenti quattro metodi:

- `getStrumenti`
- `crea_lista_eventi`
- `elabora`
- `norm`

Il metodo *`getStrumenti`* ha un duplice scopo, restituire un array di stringhe dove ogni stringa rappresenta il valore *id* di un elemento *staff* , riferendoci alla figura n. il valore *`id`* è *Clean Gt._1_staff* e memorizza all'interno della classe `XMLMX` in opportuni dizionari le informazioni relative al *`time_signature`,* che racchiude il *`time_indication`* contenente l'informazione del metro della parte presa in considerazione, e gli elementi *`measure`* .

Il secondo metodo che incontriamo all'interno della classe è *`crea_lista_eventi`* che ha il compito di memorizzare in una lista tutti gli eventi contenuti nel layer spine. Si è deciso di utilizzare una lista per permettere che gli eventi siano memorizzati mantenendo il loro ordine naturale, come sono stati trovati all'interno del file XML IEEE 1599.

Il terzo metodo di questa classe, quello più corposo, e quello che svolge il compito più significativo è `elabora` (Esempio 5.3).

Questo metodo riceve in ingresso un oggetto *LogicGuitar* e una stringa contenente l'*id* dello strumento, a questo punto richiama il metodo *crea_lista_eventi*, prende le informazioni contenute all'interno del dizionario *Strumenti*, le elabora e ne ricava gli accordi. Gli accordi vengono successivamente memorizzati all'interno di un dizionario chiamato `eve_chord` che ha come valore l'accordo stesso e come chiave il suo `event_ref` di riferimento.

L'ultimo metodo di questa classe di cui andiamo a parlare è il metodo *norm*, che si occupa semplicemente di normalizzare una stringa andando ad eliminare tutti i caratteri che non sono una lettera o un numero ottenendo così il valore pulito che potrà essere elaborato dal programma con maggiore semplicità.

```
public void elabora(string strumento,LogicGuitar lg)
{

  try
  {
    //ricava le sequenze di accordi
    crea_lista_eventi();
    XElement partid =  strumenti [norm(strumento)];
    IEnumerable<XElement> measure = partid.Elements(XName.Get("measure"));
    Dictionary<string, XElement> chord = new Dictionary<string, XElement>();
    foreach (XElement x in measure)
    {
       XElement voice = x.Element(XName.Get("voice"));
       IEnumerable<XElement> chords = voice.Elements(XName.Get("chord"));
       foreach (XElement ch in chords)
       {
            chord.Add(norm(ch.Attribute(XName.Get("event_ref")).Value), ch);
       }
    }
    //Estrapola le singole note
    foreach (KeyValuePair<string, XElement> ch in chord)
    {
        IEnumerable<XElement> notes = ch.Value.Elements("notehead");
        List<Nota> lista_note = new List<Nota>();
        Accordo acc;
        foreach (XElement note in notes)
        {
            XElement no = note.Element(XName.Get("pitch"));
            //Nota n = new Nota();
            int accidental = 0;
            char step = no.Attribute(XName.Get("step")).Value[0],
            int octave =
            Convert.ToInt16(no.Attribute(XName.Get("octave")).Value);
            string acci =
            no.Attribute(XName.Get("actual_accidental")).Value;
            if (acci == "natural") accidental = 0;
            if (acci == "flat") accidental = -1;
            if (acci == "sharp") accidental =1;
            lista_note.Add(new Nota(step, accidental, octave));
        }


        //lista_tasto = getTasti(numeric_note);
        acc = new Accordo(lista_note, 0, lg);

        eve_chord.Add(ch.Key, acc);
    }
  }
  catch
  {
    strumentoexist = false;
  }
```

Esempio 5.3: Il metodo Elabora

5.4 La classe Accordo

La classe *Accordo* rappresenta assieme alla classe *XMLMX* e alla classe *LogicGuiar* il cuore di questa applicazione. Il suo compito principale, tramite l'impiego di informazioni riguardanti le note logiche, le note fisiche e la durata, è di rappresentare un accordo. Il metodo costruttore della classe prende come parametri una lista di note, una durata, una chitarra logica, e ne ricava le note fisiche intese come coordinate grafiche corrispondenti alla nota sul manico della chitarra virtuale. Per compiere questa operazione, il costruttore richiama il metodo privato `getTasti` il quale, dato un insieme di note logiche e una chitarra logica, si occupa di restituire le coordinate. Il metodo `getTasti` inoltre estrapola dalla chitarra logica la sua accordatura e la memorizza in un array di interi. Ogni intero corrisponde alla distanza in semitoni del DO dell'ottava zero alla nota con cui è accordata l'n-esima corda. La corda assegnata al numero zero risulterà essere sempre quella accordata sulla nota più acuta e per lo stesso principio la quinta corda risulterà sempre quella accordata sulla nota più grave, per esempio se si decide di usare l'accordatura standard di una chitarra (MI, LA, RE, SOL, SI, MI) la corda zero sarà memorizzata all'interno dell'array come Mi cantino, la corda uno come Si cantino, la corda due come Sol cantino e così via.
Due note non possono essere suonate sulla stessa corda nello stesso istante di tempo, per gestire questo caso il metodo `getTasti` inizializza un array booleano, chiamato `corda_occupata`. Questo array booleano ha valore true all'i-esima posizione se l'i-esima corda è occupata e valore false se all'i-esima posizione l'i-esima corda è libera, inizialmente l'array avrà tutti valori false per permettere all'algoritmo di tener traccia delle corde in cui è già stata posizionata una nota. Il metodo `getTasti` ha al suo interno un terzo ed ultimo array di interi in cui è memorizzato i corretti tasti da premere estrapolati dall'algoritmo per ogni corda (Figura 5.4) , si noti che il valore zero indica una corda suonata a vuoto, il valore uno il primo tasto e così via.

5.5 Algoritmo e la classe Logic Guitar

Il primo compito svolto dall'algoritmo è ordinare in modo crescente le note che gli sono state passate come parametri grazie all'ausilio di una primitiva Array.Sort. Successivamente partendo dalla nota più acuta tenta di posizionarla sul manico della chitarra virtuale usufruendo della corda più acuta ancora disponibile. L'algoritmo ripete lo stesso procedimento per tutte le altre note con valore frequenziale decrescente fino a quando avranno trovato la loro occupazione. Di fondamentale importanza è l'ordine con cui questa operazione viene eseguita dall'algoritmo, esso si pone infatti l'obiettivo di rappresentare le note in posizioni facilmente raggiungibili dalla mano sinistra, e che si trovino il più possibile vicino al capotasto.

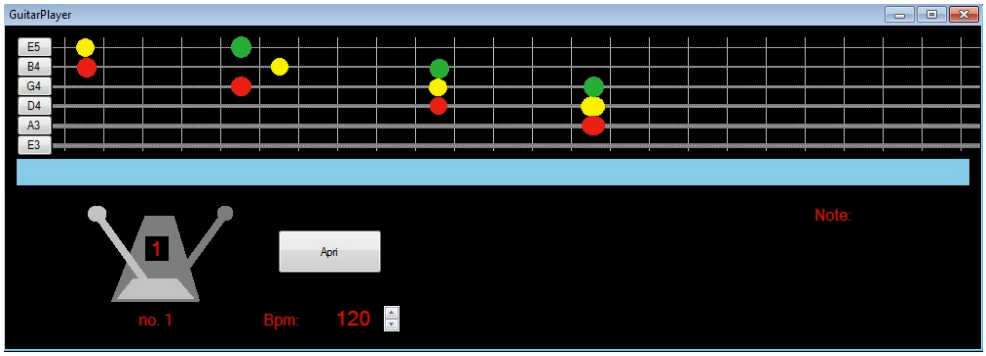

Figura 5.4

Si prenda come esempio il primo rivolto di Fa maggiore, composto da: C4, F5, A5. Ciascuna nota può essere rappresentata in tasti differenti sul manico della chitarra come è ravvisabile dalla figura 5.4.

C4 = pallino rosso

F5 = pallino giallo

A5 = pallino verde

Si vada ad analizzare il caso in cui l'algoritmo sia applicato alla sequenza di note C4, F5, A5 partendo quindi a posizionare le note sul manico da quella più grave a quella più acuta.

Si inizi con C4, la prima corda libera sulla quale può essere posizionata è la seconda, posizioniamola. Si Prosegua ora con F5, che verrà posizionata dall'algoritmo ancora sulla prima corda non occupata, quella del Mi cantino.

Infine si vada a posizionare la terza nota sul manico, l'algoritmo trovando prima e seconda corda occupata andrà a posizionare A5 nella prima corda ancora disponibile, si tratta della terza, sul quattordicesimo tasto. L'accordo risultante come si nota nella figura 5.5 ha senso logico ma è fisicamente insuonabile dalla mano sinistra.

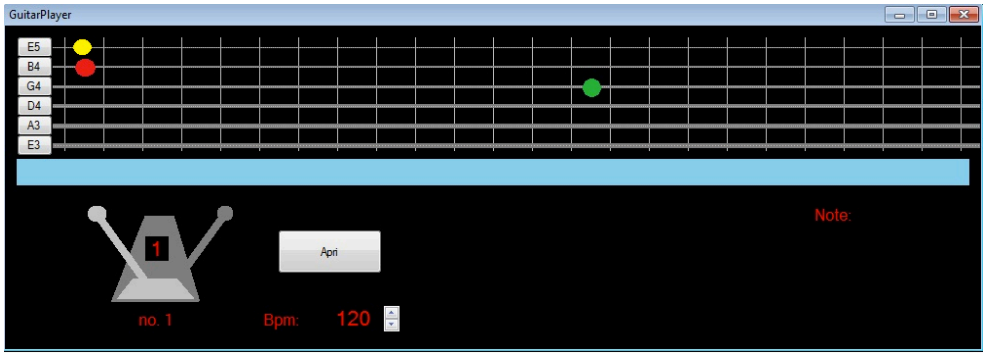

Figura 5.5

Si analizzi ora il vero funzionamento dell'algoritmo. Si parta a posizionare le note nella sequenza corretta, partendo dalla nota più acuta fino ad arrivare a quella più grave (A5, F5, C4).

L'algoritmo inizia a posizionare A5 sulla corda del Mi cantino nel quinto tasto, si passi ora ad F5 che trovando la prima corda occupata sarà posizionata sulla seconda nel sesto tasto, infine si posizioni C4 che trovando la prima e la seconda corda occupate sarà visualizzata sulla terza corda nel quinto tasto. L'accordo risultante avrà senso logico e sarà allo stesso tempo facilmente suonabile (Figura 5.6).

Figura 5.6

Una delle caratteristiche che sono state aggiunte al software in fase di creazione è la possibilità di visualizzare tramite dei pallini grigi posizionati sul manico della chitarra l'accordo da suonare successivamente.
La classe a cui è stato assegnato questo compito si chiama `LogicGuitar`, essa tiene traccia, istante per istante, dell'accordo da suonare al momento n e dell'accordo al momento immediatamente successivo (n+1).
`LogicGuitar` presenta al suo interno una serie di metodi, oltre al metodo costruttore, che setta l'accordatura della corda, sono presenti il metodo `getListaAccordi`, che una volta memorizzata la lista degli accordi passata come parametro va a settare come `Now_chord` il primo accordo di durata non nulla della lista e come `After_chord` l'accordo successivo, si trova il metodo `Next`. Questo metodo svolge la funzione di scorrere gli accordi e passare all'accordo successivo come rappresentato dall'Esempio 5.4.

```
private void Next()
{
  Now_Chord = After_Chord;
  After_Chord = After_Chord.Next;
  if (After_Chord == null && !preflag)
  {
    After_Chord = new LinkedListNode<Accordo>(new Accordo());
    preflag = true;
  }
  else
  {
    if (After_Chord == null && preflag)
    {
      After_Chord = new LinkedListNode<Accordo>(new
      Accordo());
      flag = true;
    }
  }
 }
```

Esempio 5.4: Il metodo privato Next

Si consideri il seguente esempio: si ha un accordo nella posizione $\texttt{Now\_Chord}_0$, accordo visualizzato sulla chitarra con dei pallini rossi al momento n. Si consideri ora `After_Chord`, l'accordo seguente della lista, e si vada ad analizzare il comportamento del metodo `Next` di conseguenza. Il metodo scarta $\texttt{Now\_Chord}_0$ in quanto è scaduto il suo valore temporale e sposta `After_Chord` nella posizione precedentemente occupata da $\texttt{Now\_Chord}_0$, risulta così che quello che prima era `After_Chord` è ora $\texttt{Now\_Chord}_0$ e l'accordo seguente della lista è il nuovo `After_Chord`. L'ultimo metodo presente in questa classe è `Play` che quando viene chiamato controlla se `After_Chord` ha durata pari a zero, in tal caso richiama il metodo `Next` e in parallelo decrementa la durata di `After_Chord`. Per fare in modo che `Now_Chord` e `After_Chord` siano visualizzati sulla chitarra virtuale nello stesso istante, la classe `guitarControl` avvalendosi delle funzionalità del metodo `Play` disegna i due accordi sul manico della chitarra con l'ausilio delle direttive grafiche.

Conclusioni

Il presente lavoro, svolto come elaborato finale del corso in Scienze e Tecnologie della Comunicazione Musicale si pone l'obiettivo di dimostrare le potenzialità offerte dallo standard IEEE 1599. Nella realizzazione di questo software è stata presa in considerazione una piccolissima parte delle informazioni contenute in un file XML IEEE 1599, tuttavia le potenzialità di questo standard permettono di inglobare una moltitudine di contenuti differenti: partitura del brano, video del brano, audio, immagini, il testo ecc.. tutte all'interno di un unico file. L'informazione finale è quindi completa in ogni sua parte. Il software realizzato vuole essere un esempio di applicazione creata partendo da questo standard, è stata la prima volta che mi sono cimentato nella realizzazione di un applicazione informatica che coinvolgesse diverse conoscenze, è stato un lavoro costruttivo che ha permesso di migliorare le mie capacità in ambito programmativo e di approfondire tutti gli argomenti trattati in questo elaborato.

Ringraziamenti

Quest'oggi si conclude una parte del mio percorso universitario, in un misto di allegria e malinconia. Un percorso che mi ha permesso di imparare molte cose e allo stesso tempo di crescere.

Le persone a cui rivolgo il più grosso ringraziamento sono mio padre Giuseppe e mia madre Sandra, persone splendide che sono sempre al mio fianco e hanno fatto in modo che potessi studiare questi tre anni a Milano, nonostante l'impegno economico fosse considerevole.

Voglio ringraziare anche mia zia Oriana e mia nonna Cesira, pronte ad accogliermi sempre con il sorriso e un buon piatto di minestra calda. Ringrazio inoltre tutti i docenti, in particolar modo il mio relatore, Dottor Luca Andrea Ludovico ed il Correlatore, Dottor Adriano Baratè per l'aiuto e le dritte fornitemi durante questi mesi di tirocinio e preparazione dell'elaborato finale.

Ho vissuto ogni corso con entusiasmo e voglia di apprendere, è stata un'esperienza bellissima.

Voglio per l'appunto ringraziare i miei compagni di corso, che hanno rappresentato per me una vera e propria famiglia in questi anni: Kekko, Limo, Ale, Jack, Spritz, Fausto, Sam, Fabio, Simo, Ste e Diego.

Ringrazio Fausto Dasè per avermi dato l'opportunità di intraprendere la prima esperienza seria all'interno del mondo lavorativo e per essere un caro amico allo stesso tempo; lavorare in un vero studio di registrazione e confrontarmi ogni giorno con professionisti nel campo della musica, del cinema e della fotografia rappresenta uno spunto importante per mettermi in gioco e esporre le mie idee.

Ringrazio in modo particolare i SEESIDE: Cipo, Fili, Art, Giovà e tutti i collaboratori dell'Accademia del suono di Milano fra cui: Vito, Sciur Simoni, Fabio e Ettore per farmi passare momenti di gioia e di serenità. Un ultimo ringraziamento è rivolto ai miei coinquilini, Kekko e Laura, per l'affetto dimostratomi in questi tre anni, per essere sempre presenti chi con una partita alla Wii e un tè caldo, chi con lunghi e noiosissimi discorsi di medicina davanti ad un piatto di pasta.

Grazie a tutti.

Bibliografia

[1] Sergio Cingolani e Renato Spagnolo (2008), *Acustica musicale e architettonica*, Città Studi Edizioni.

[2] Devan Shepherd (2001), *XML Guida Completa*, Apogeo.

[3] Luca Andrea Ludovico (2005), *Manuale di MX, LIM* – Laboratorio di Informatica Musicale.

[4] Luca Andrea Ludovico (2009), IEEE 1599: *a Multi-layer Approach to Music Description*, Journal of Multimedia, Vol.4, Num.1.

[5] G.Haus and M.Longari (2002), *Proceedings of the First International IEEE Conference on Musical Application using XML (MAX2002)*, IEEE Computer Society.

[6] John Sharp (2010), *Microsoft Visual C#*, Mondadori Informatica.

Sitografia

http://www.ludovico.net

http://msdn.microsoft.com

http://www.lim.dico.unimi.it

http://www.mx.dico.unimi.it

http://www.comunicati-stampa.ws