UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica Musicale

MODULO MULTIPIATTAFORMA PER LA GESTIONE DELLO SPINE IEEE 1599

Relatore:

Prof. Luca Andrea Ludovico

Correlatore:

Prof. Adriano Baratè

Elaborato finale di:

Fabrizio Massara

Matricola 773959

ANNO ACCADEMICO 2012 - 2013

Indice

INTRODUZIONE

Il presente elaborato è nato dall'esigenza, da parte del LIM, di fornire un Framework utilizzabile su differenti sistemi operativi. Più precisamente, andremo ad analizzare quanto è stato fatto nella creazione di una parte di tale applicativo, vale a dire per la funzione di Event Inspector, cioè l'analisi della presenza degli eventi all'interno di un documento XML.

In primis verrà fornita una visione d'insieme su cosa sia lo Standard IEEE 1599 e come sia stato utilizzato nello studio di brani musicali da parte dell'Università degli Studi di Milano; si esaminerà poi l'utilizzo di tale standard all'interno di file XML con lo scopo di definirne una struttura univoca.

Nel secondo capitolo si discuterà del progetto open source Mono e delle tecnologie utilizzate nel corso dell'elaborato, descrivendone i pregi e il perché si sia deciso di basarsi su di esso. Sarà scopo del capitolo quello di esporre quanto più sinteticamente e precisamente le funzioni più interessanti dei linguaggi di programmazione C# e GTK+ e dell'IDE Xamarin Studio.

Il terzo capitolo definirà nella maniera più completa possibile la funzione in esame, andando ad analizzare i metodi usati e la struttura del Framework. Inizio e fine saranno incentrati sul lato utente, vale a dire cosa si troverà di fronte e quali saranno i passaggi che dovranno essere eseguiti per poter utilizzare correttamente gli strumenti proposti. La parte centrale del capitolo sarà riservata al codice utilizzato e alle scelte effettuate per rendere il tutto omogeneo e funzionale, con particolare attenzione sul perché siano state adottate particolari politiche d'implementazione invece che altre.

Infine, l'ultimo capitolo sarà riservato a un riassunto di quanto è stato scritto e delle idee sulle possibili implementazioni che potranno essere effettuate sul Framework in futuro, al fine di renderlo maggiormente funzionale e ricco di strumenti, per rendere l'analisi e lo studio di partiture e brani il più accurato possibile.

CAPITOLO 1: STANDARD IEEE 1599

IEEE 1599, divenuto standard nel 2008, ha avuto origine dalle idee innovative e dagli sforzi di un gruppo di studio italiano, guidato dal Prof. Goffredo Haus presso il Laboratorio di Informatica Musicale (LIM) dell'Università degli Studi di Milano[1].
Questo è uno dei tanti progetti realizzati all'interno del LIM per l'analisi di brani e dei più differenti aspetti musicologici. Vogliamo quindi citare un altro software applicativo di particolare interesse, sviluppato al fine di fornire ulteriori strumenti nello studio di brani mediante Reti di Petri, questi prende il nome di Scoresynth[2]. Tale applicativo fornisce la possibilità di disegnare reti e sottoreti, includendo file musicali e partiture, così che, se avviato, l'utente sia in grado di vedere l'evoluzione temporale della rete durante il brano.
Abitualmente si considera la musica prettamente nell'ambito estetico e artistico, ma un diverso approccio permette di analizzare i brani più in profondità, esaminando le differenti parti e le singole note che completano la composizione nella sua complessità.
Lo standard IEEE 1599 nasce proprio per questo studio, codificando un brano in formato XML fornisce all'utente una visione d'insieme della struttura di un'opera delineando tutti gli elementi che ne fanno parte.
Vedere la musica sotto forma di codice astrae dalla normale concezione della stessa e fornisce la base per analizzarla sotto un altro aspetto. Non è più una sequenza di note su un pentagramma, ma una sequenza di eventi su un foglio di testo.
Un concetto fondamentale nella creazione di XML basati su questo nuovo standard è quello di *Layer [3]*. Ne riconosciamo cinque:

- *General*
- *Logical*
    - *Spine*
    - *Logically Organized Symbols (Los)*
- *Notational*
- *Performance*
- *Audio*

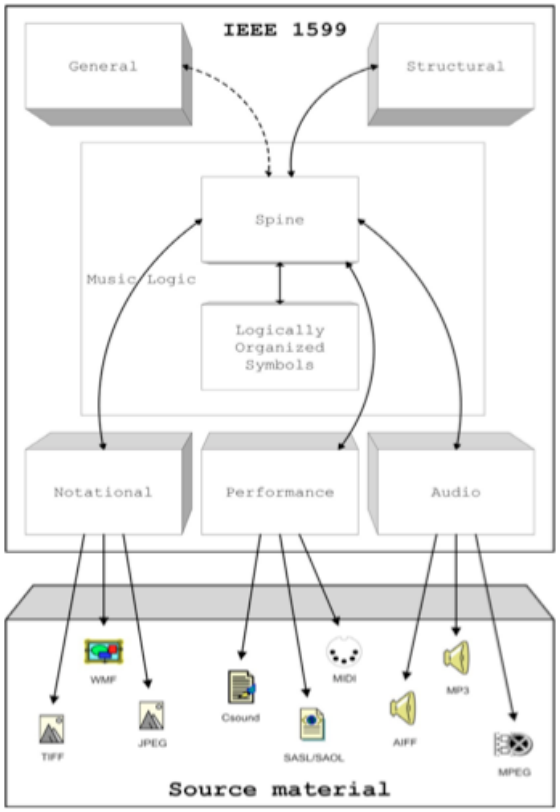


Fig.1: Relazione tra i diversi layer nello standard IEEE 1599

Per l'analisi dei vari layer e per renderne più chiari gli aspetti pratici, prenderemo in considerazione il file Zauberflote.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM
"http://standards.ieee.org/downloads/1599/1599-2008/ieee1599.dtd"[]>
<ieee1599 creator="Finale Plugin" version="1.0">
```

Come si può vedere dal codice di cui sopra, l'intestazione del file XML contiene le informazioni necessarie al programma per poterlo gestire. Quindi la versione di XML in cui è stato scritto il codice e il formato di codifica, in questo caso UTF-8[1].

A seguire viene identificato lo standard IEEE 1599 e il DTD (Document Type Definition) utilizzato, il quale descrive la struttura necessaria perché il documento possa essere considerato valido secondo lo standard utilizzato.

[1] Unicode Transformation Format, 8 bit: codifica dei caratteri Unicode in sequenze di lunghezza variabile di byte.

[2] http://www.lim.dico.unimi.it/ all'interno della sezione Download è presente il plug-in per

Il DTD deve essere incluso obbligatoriamente perché funge da "mappa" per il programma che andrà ad aprire il file XML in questione. Esso può essere definito sia internamente al documento, sia esternamente, l'importante è che la dicitura rimandante a tale definizione sia presente[5].
Tale programma aprirà il documento e verificherà se la struttura utilizzata corrisponde, in tal caso il documento verrà considerato valido e potrà essere utilizzato, in caso contrario si dovrà rendere conforme la struttura a quella definita all'interno del DTD e in seguito il documento dovrà essere validato.
Immediatamente dopo troviamo la dicitura "ieee1599 creator", la quale definisce che il documento è stato creato sfruttando il Plugin apposito per Finale[2].

General

```
<general>
    <description>
      <main_title> Der Holle Rache kocht in meinem Herzen
      </main_title>
      <author type="composer"> Wolfgang Amadeus Mozart
      </author>
      <author type="librettist"> Emanuel Schikaneder</author>
      <work_title> Die Zauberflote</work_title>
    </description>
</general>
```

Si tratta di un livello fondamentalmente descrittivo, in cui vengono specificate le informazioni principali del documento, quali il titolo della raccolta, il nome del compositore, dell'editore e il titolo dell'opera. Queste sono le informazioni di base, a cui se ne possono aggiungere molte altre quali, ad esempio, l'anno in cui è stata pubblicata l'opera.

[2] http://www.lim.dico.unimi.it/ all'interno della sezione Download è presente il plug-in per Finale, utilizzabile sia su Windows che su Mac. Permette l'esportazione delle partiture in un file di testo codificato nel formato IEEE 1599

Spine

```
<spine>
     <event id="Flauti_1_voice0_measure1_ev0" timing="0" hpos="0" />
     <event id="Oboi_2_voice0_measure1_ev0" timing="0" hpos="0" />
     <event id="Fagotti_3_voice0_measure1_ev0" timing="0" hpos="0"/>
     <event id="Corni_4_voice0_measure1_ev0" timing="0" hpos="0" />
     <event id="Trombe_5_voice0_measure1_ev0" timing="0" hpos="0" />
```

Passando poi alla parte d'interesse pratico, con lo spine si entra nel vivo del codice XML inerente al vero e proprio contenuto musicale dell'opera. Come possiamo vedere dalla figura che mostra le diverse relazioni tra i layer (fig. 1), lo spine ha un ruolo centrale, rendendo dipendenti da esso tutti gli altri livelli, alcuni in maniera più marcata di altri. Il cuore dell'XML rimane sempre e comunque questo layer [4].

Ogni *Music Object* (MO) viene rappresentato all'interno del codice come un *Evento* univoco. Per Music Object s'intende qualsiasi figura abbia un valore musicale. Dall'esempio presentato precedentemente prendiamo in considerazione il primo evento.

`Flauti_1_voice0_measure1_ev0`

Rappresenta il MO associato alla parte riservata ai Flauti (identificata dalla posizione 1) e presente per la voce in prima posizione (se la parte di Flauti fosse stata polifonica avrebbe avuto più voci, ma in questo caso è presente una sola voce quindi si identificherà con `voice0`) alla battuta numero 2 (la numerazione delle battute parte da 0, quindi `measure1` indica la seconda), infine `ev0` indica che è il primo evento presente per la parte di Flauti.

Vi sono poi due ultimi elementi riservati a ciascun evento, timing e hpos, i quali rappresentano rispettivamente le coordinate temporali e il posizionamento orizzontale virtuale della nota sul pentagramma.

Il timing viene espresso in *Virtual Time Unit*s (VTU), il cui valore viene indicato all'interno delle specifiche SDML (Signed Document Markup Language). Se, ad esempio, la nota più breve all'interno dello spartito è 1/16 si può definire come VTU la durata di 1/16, in questo modo le note seguenti potranno essere identificate come multipli della VTU.

Hpos, invece, viene misurato in *Virtual PiXels* (VPX) che fornisce la posizione verticale di ogni singolo elemento all'interno delle diverse voci e parti dello spartito. Il valore hpos=0 indica l'allineamento verticale delle note.

Logically Organized Symbols

```
<los>
      <staff_list>
        <staff id="Flauti_1_staff" line_number="5">
          <time_signature event_ref="TimeSignature_Flauti_1_1">
          <time_indication num="4" den="4" />
          </time_signature>
          <key_signature event_ref="KeySignature_Flauti_1_1">
          <flat_num number="1" />
          </key_signature>
          <clef shape="G" event_ref="Clef_Flauti_1_1" staff_step="2"
octave_num="0" />
        </staff>
```

Attraverso il *los* (acronimo del layer in questione) siamo in grado di analizzare più attentamente ogni singolo evento. Come ci suggerisce il nome, in questa porzione di codice viene messo in rilievo l'aspetto dei simboli all'interno della partitura.

Nella prime righe viene reso visibile il nome dell'evento in esame, in questo caso `Flauti_1_staff` cioè il rigo riservato alla parte dei Flauti.

Una caratteristica particolare per questa tipologia di evento è l'inglobamento di altri due eventi, i quali vanno a completare le informazioni relative alla parte che stiamo esaminando, e sono: l'indicazione del tempo rappresentata dall'evento `TimeSignature_Flauti_1_1`, la tonalità identificata da `KeySignature_Flauti_1_1` e infine la chiave denominata, appunto, `Clef_Flauti_1_1`.

Esaminando ciascuna informazione nel suo complesso possiamo notare che, per quanto riguarda il tempo del brano, esso viene rappresentato tramite due parametri che sono numeratore e denominatore. In questo caso la parte sarà in 4/4.

Tra i parametri della tonalità, invece, possiamo vedere che viene indicato `flat_num number="1"` vale a dire che è presente un'alterazione in chiave, più precisamente un bemolle (*flat* infatti è la traduzione di *bemolle*, mentre se fosse stato *sharp* allora avrebbe indicato la presenza di un *diesis*).

Infine, la chiave utilizzata è indicata come *G centrale*, cioè una normale *Chiave di Violino.*

Allo stesso modo per le note si utilizzano parametri come la tonalità, la durata ed eventuali accenti. Ogni singola parte musicale è seguita da tutti gli eventi che la rappresentano, permettendo di analizzarle nella loro interezza.

```
<chord event_ref="p7v1_69">
  <notehead>
    <pitch step="E" octave="5" />
    <duration num="1" den="1" />
  </notehead>
  <notehead>
    <pitch step="G" octave="5" />
    <duration num="1" den="1" />
  </notehead>
  <notehead>
    <pitch step="C" octave="6" />
    <duration num="1" den="1" />
  </notehead>
</chord>
```

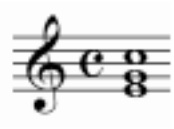


Fig. 2: Codice e rappresentazione grafica di un evento [3]

Per esemplificare prendiamo come esempio la figura al di sopra (Fig. 2) nella quale vengono messi a confronto la parte di codice riguardante un evento all'interno del los e la sua rappresentazione grafica.

In tal caso l'evento sarà un accordo, quindi si dovrà valutare la presenza contemporanea di tre note sulla medesima parte. Vediamo che viene creato un insieme contenente le tre diverse note che formano l'accordo di MI e quindi MI, SOL e DO. Per ciascuna di esse viene identificata l'ottava relativa e la durata, in questo caso è segnato 1 poiché il tempo del brano è in 4/4.

Notational

```
  <notational>
    <graphic_instance_group description="Original">
      <graphic_instance file_name="score/Mozart_Zauberflote_14_Arie-
1.jpg" file_format="image_jpeg" encoding_format="image_jpeg"
position_in_group="1" measurement_unit="pixels">
        <graphic_event event_ref="Flauti_1_voice0_measure1_ev0"
upper_left_x="305" upper_left_y="126" lower_right_x="334"
lower_right_y="150" />
        <graphic_event event_ref="Flauti_1_voice0_measure2_ev0"
upper_left_x="397" upper_left_y="100" lower_right_x="435"
lower_right_y="134" />
```

Il layer notational fornisce informazioni grafiche, sfruttabile per una mappatura grafica della partitura, assegnando ad ogni evento una posizione specifica sullo spartito, evidenziata attraverso un rettangolo di cui si registrano le coordinate partendo dall'angolo in alto a sinistra e dall'angolo in basso a destra, il tutto in termini di pixel.
È possibile esaminare differenti file riguardo alla partitura, ogni file viene mappato graficamente e inserito in un gruppo.
Nell'esempio, il file `score/Mozart_Zauberflote_14_Arie-1.jpg file_format=image_jpeg` codificato attraverso codifica jpeg viene inserito all'interno del gruppo `Original` rendendo più facile l'esame di diverse scansioni dello spartito.

Performance

```
<performance>
    <midi_instance file_name="gottes/gottes.mid" format="1">
      <midi_mapping part_ref="soprano" track="1" channel="1">
        <midi_event_sequence                division_type="timecode"
division_value="1024" measurement_unit="ticks">
          <midi_event event_ref="v1_e0" timing="0"/>
```

Altro layer di mappatura è quello riguardante i file di tipologia MIDI, in cui vengono descritti i vari eventi inerenti alle informazioni tipiche di un file di questo tipo, quindi la traccia, il canale su cui deve essere riprodotto l'evento e il timecode, cioè la velocità di riproduzione, calcolato in numero di ticks.

Audio

```
<audio>
    <track                    file_name="audio/Cristina_Deutekom.mp3"
file_format="audio_mp3" encoding_format="audio_mp3">
      <track_indexing timing_type="seconds">
        <track_event         event_ref="Flauti_1_voice0_measure1_ev0"
start_time="0.79" />
```

Ultimo layer, anch'esso di mappatura, rappresenta la durata all'interno di una registrazione dell'opera di ogni singolo evento.
Per ogni traccia viene messo in evidenza, oltre al nome e al formato di codifica, l'indice temporale che verrà considerato, in questo caso i secondi.

Ogni evento viene registrato in base al minuto in cui compare nel file audio. Ciò permette di esaminare le differenze d'esecuzione di differenti brani.

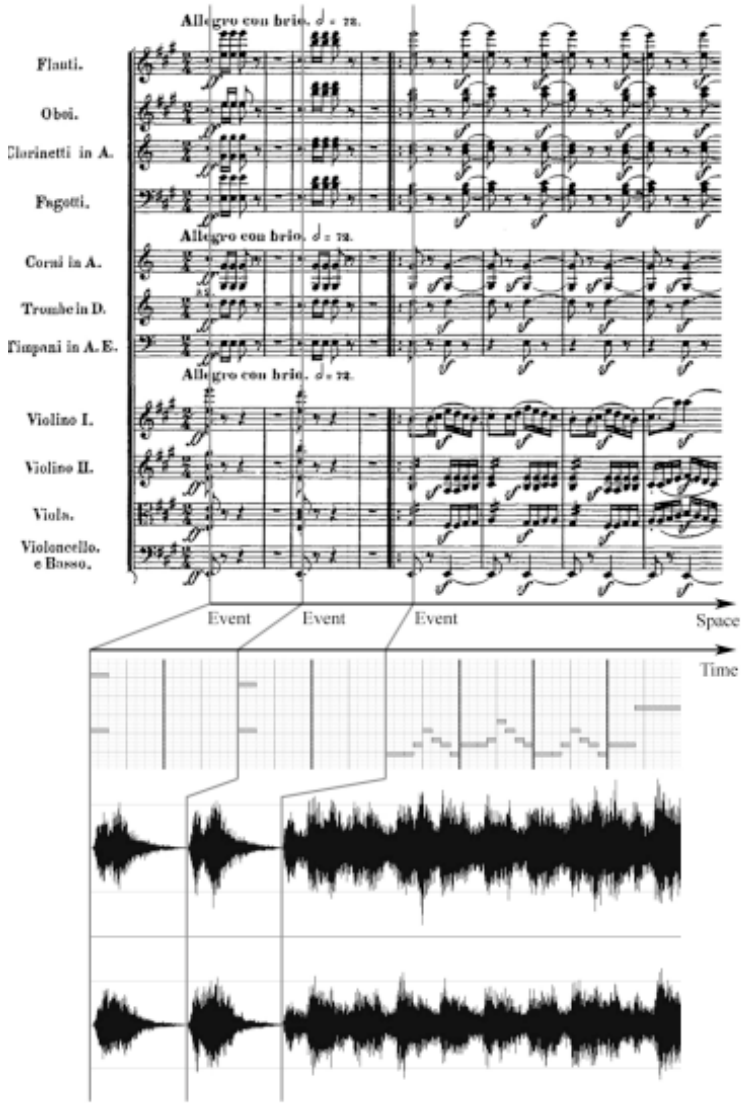


Fig.3: Sincronizzazione dei Layer con lo Spine

Come possiamo notare dalla figura (Fig. 3), i tre layer Notational, Performance e Audio si trovano a cooperare e rappresentare la medesima informazione in differenti forme [3]. Ricordiamo lo schema visto in precedenza (Fig. 1) e qua è possibile dimostrare la centralità dello spine rispetto agli altri layer.

Al di sopra viene rappresentata la sequenza di Eventi dello spine in maniera grafica attraverso la partitura, subito al di sotto troviamo il file MIDI mediante il quale vengono rappresentate solamente il pitch, cioè l'altezza e la durata di ciascuna nota. In ultima analisi vediamo il contenuto spettrale del file audio per cui verranno esaminati gli eventi in ordine di click.

CAPITOLO 2: MONO

Si tratta di una piattaforma software ideata con lo scopo di dare degli strumenti agli sviluppatori per permettergli la creazione di applicazioni cross-platform in maniera più efficiente e rapida [6].

Pone le sue basi sull'idea di open source[3], infatti si tratta del corrispettivo gratuito di Microsoft .NET Framework[7], ma la caratteristica di essere multipiattaforma lo rende maggiormente versatile. I programmatori che sfruttano Mono possono creare applicazioni quasi per qualsiasi elaboratore e sistema operativo in circolazione.

Miguel de Icaza fondò la società Ximian nel 2000, la quale era specializzata in software open source. Attraverso tale società aveva intenzione di costruire strumenti di sviluppo software che rendessero più produttiva la creazione di software multipiattaforma. Non appena la Microsoft presentò il Framework .NET, la Ximian cominciò ad interessarsene e a ideare quello che nel 2001 venne nominato "progetto open source Mono" (al momento la versione corrente è la 2.0).

Differenti strumenti vengono sfruttati per renderlo estremamente competitivo sia dal punto di vista delle capacità che delle funzionalità messe a disposizione dei programmatori. Tra questi strumenti poniamo l'attenzione specialmente sul compilatore C# (sfruttabile anche dal .NET Framework), sul CLR (Common Language Runtime), che permette di sfruttare le librerie di Microsoft, e su Xamarin Studio software che analizzeremo più avanti, e molto altro ancora.

Nei prossimi paragrafi andremo ad esaminare i vari strumenti di rilievo per gli argomenti che affronteremo più avanti, in modo da avere una visione d'insieme e capire perché si è scelto di affidarsi ad esso per il nostro scopo.

L'IDE[4] utilizzato prende il nome di Mono Develop ed è stato creato principalmente per il linguaggio C# e altri linguaggi relativi al .NET Framework, ma supporta anche C, C++, Java, Python e Vala. Si pone come alternativa a SharpDevelop a sua volta nato come concorrente di Microsoft Visual Studio, sfruttando GTK#, un'evoluzione di GTK+, per la creazione delle interfacce e di cui parleremo più approfonditamente nei prossimi capitoli.

---

[3] Software fornito gratuitamente dagli autori, per cui è permessa la modifica e lo sviluppo da parte di tutti coloro che ne entrano in possesso, il codice viene infatti reso disponibile unitamente al programma.

[4] Integrated Development Environment: si tratta di un programma creato per facilitare il compito dei programmatori, mettendo a disposizione un editor per il codice sorgente, un debugger e automazione per la scrittura del codice.

2.1 C# (C Sharp)

Alla base di ogni software vi è il linguaggio di programmazione utilizzato e in questo caso si è scelto di sfruttare le potenzialità offerte da un linguaggio che sta pian piano prendendo più forma e che punta a superare il concorrente Java.
Entrambi fanno parte della categoria riservata alla Programmazione Orientata agli Oggetti (OOP, Object Oriented Programmation), una tipologia che permette di astrarre ulteriormente il concetto di programmazione, andando ad agire sul programma sfruttando le nozioni di Classi, Metodi e, appunto, Oggetti.
C# si sta rivelando molto competitivo in confronto con Java, poiché, sfruttando la sintassi molto simile sia al linguaggio in questione sia ai predecessori C e C++, permette ai programmatori di scrivere un codice in maniera più chiara e più facilmente interpretabile. Il maggior numero di costrutti e di regole però, a detta dei sostenitori di Java, renderebbero il linguaggio C# più difficile da imparare e da scrivere.

Un'importante qualità di questo linguaggio è la possibilità di eseguire operazioni su file XML, il che lo rende molto utile per agire su documenti creati seguendo lo standard IEEE1599 di nostro interesse.
Aprendo una parentesi sull'XML possiamo definirlo un linguaggio molto descrittivo che permette di descrivere, seguendo determinati standard definiti dai DTD, una struttura per ogni tipo di contenuto che si vuole comunicare, per esempio la struttura composta dai vari layer analizzata nel primo capitolo ci permette di trovare immediatamente le varie sezioni d'interesse musicologico e capire come sia strutturato un brano.

Tornando a discutere del linguaggio vero e proprio esso trova origine in ambiente Microsoft nel 2000, circa vent'anni dopo il predecessore C++ [8].
Ma entrambi questi linguaggi hanno un'origine, e questa è il C, ideato agli inizi degli anni Settanta si è rivelato estremamente efficace, sia dal punto di vista dell'apprendimento visto che è risultato molto intuitivo, sia per quanto riguarda le enormi capacità di cui dispone. Si tratta infatti di un linguaggio di alto livello che permette un'astrazione minima rispetto ad altri della medesima tipologia, come Pascal. Permette all'utente di avere un'idea maggiore sul funzionamento dell'Assembly, grazie anche all'introduzione del concetto di *puntatore,* mediante il

quale si affronta il problema di allocazione della memoria e di raggiungimento della posizione del contenuto di una determinata variabile all'interno dello stack o ancora dell'ottenimento vero e proprio dell'indirizzo ad essa associata.
Uno dei problemi che ci si vede costretti ad affrontare e che può complicare la progettazione di applicativi basati su questo tipo di programmazione è individuabile nella necessità di sfruttare un quantitativo maggiore di istruzioni da parte dell'elaboratore e di dispendio di memoria. Applicativi in Java o C++ con notevole contenuto di codice risultano più lenti, comportando anche un rallentamento delle applicazioni in esecuzione sull'elaboratore, nel caso esso non abbia un quantitativo di memoria o una componentistica hardware adeguata. Un altro ostacolo che ci si deve preparare ad affrontare nel momento in cui si decide di approcciarsi alla programmazione ad oggetti è la complessità, dato che si raggiunge come si diceva in precedenza un livello maggiore di astrazione rispetto ai primi linguaggi procedurali.
D'altro canto, il C++ si avvicina alle esigenze dei neofiti in campo di programmazione ad oggetti ma che comunque hanno delle nozioni nei linguaggi di base, dando la possibilità di sfruttare le potenzialità e le caratteristiche del C per facilitare l'apprendimento senza cominciare da zero. Sebbene ciò abbia reso più semplice avvicinarsi a tali linguaggi si è notato che i programmi compilati in C++ soffrivano di una certa instabilità e quindi si è resi più propensi ad utilizzare Java, creato circa un decennio dopo, il quale ancora oggi ha assunto un ruolo centrale nella programmazione ad oggetti.
Questo ruolo si trova ora a vacillare di fronte alla comparsa del C#, l'ultima creazione in campo di linguaggio C che ha apportato delle migliorie notevoli al predecessore ponendosi nettamente al di sopra per intuitività dell'apprendimento e di facilità di utilizzo.
Oltre ad essere relativamente più semplice di altri linguaggi di programmazione ad oggetti, permette un utilizzo più efficiente e meno dispendioso in termini di utilizzo della memoria e delle energie usate dal processore per i calcoli. Questa maggiore efficienza ha portato ad avere programmi più stabili e con una maggiore rendita.

Una delle tattiche messe a disposizione dal C# per mantenere quanto più efficiente l'utilizzo della memoria e non dover eseguire istruzioni che andrebbero a complicare e rallentare l'utilizzo è la possibilità di utilizzare le *struct,* già presente nelle versioni precedenti del C, al posto delle classi.

Un'altra funzione ereditata dal C è la possibilità di generare codice di tipo unsafe[5] per gestire i puntatori, cosa che in Java non è possibile.
Altra differenza è che in C# "tutto è un oggetto", affermazione in questo caso veritiera, poiché rispetto a Java vengono sfruttate due importanti funzioni e cioè boxing e unboxing. Per esemplificare prendiamo, appunto, una scatola che può essere aperta e in cui si possono inserire determinati oggetti, al termine del processo essa viene chiusa e bisognerà riaprirla per ottenere il contenuto.
Allo stesso modo una variabile, quindi definita tramite valore può essere impacchettata (boxing) in una scatola, diventando così un oggetto. Nel momento del boxing viene allocata in memoria la posizione dell'oggetto e il suo valore, tramutando così il tipo da un value type a un reference type. Un esempio di value type può essere una struct, mentre un esempio di reference type può essere una classe.
Se si vorrà tornare a utilizzare l'oggetto per il valore che contiene allora si eseguirà lo spacchettamento della scatola (unboxing).

2.2 GTK+ (GIMP ToolKit)

GTK+, la prima versione, nacque come insieme di strumenti, compresa la relativa libreria, per implementare nuove funzioni all'interno del software di grafica GIMP[6], anch'esso come il resto del progetto Mono è Open Source [9].
Anch'esso, come il C#, trova le sue basi nel linguaggio C, mediante il quale venne sviluppato. Divenne presto indispensabile nella creazione della grafica di applicazioni open source arrivando ad essere uno dei punti centrali nella programmazione dell'ambiente desktop GNOME e su cui tali sistemi si basano ancora oggi.
Per la creazione di widget grafici da aggiungere alle applicazioni, GTK sfrutta la libreria Xlib. Ciò rende il programma più flessibile e permette l'utilizzo su sistemi che non supportano la libreria X Window.

Uno degli strumenti forniti e che è stato implementato all'interno del Framework è il GtkComboBox. Si tratta di un popup che mette a disposizione dell'utente un form con

[5] Si intendono istruzioni che, specificate attraverso la dicitura unsafe, non verranno sottoposte ad alcun controllo da parte del Common Language Runtime (CLR). Sarà responsabilità del programmatore assicurare che ciò non comporti problemi di protezione o errori in memoria e tra i puntatori.
[6] GNU Image Manipulation Program, software open source per la modifica di immagini. Nato come alternativa al colosso di Adobe, Photoshop.

una lista di scelte che possono essere selezionate. Lo stile con cui viene rappresentato questo form dipende dalle impostazioni fissate nel codice, si possono infatti stabilire tutti parametri grafici che andranno ad influire sulla raffigurazione grafica dell'oggetto.

2.3 XAMARIN Studio

Si tratta di un IDE di ultima generazione, nato per permettere ai programmatori di creare applicazioni multipiattaforma sfruttando le potenzialità del C#. È possibile interagire con l'IDE ufficiale di Microsoft, vale a dire Visual Studio, in modo tale da poter implementare funzioni che per il momento Xamarin non mette a disposizione.

Fig.4: Logo di Xamarin

L'aspetto che lo rende un software altamente competitivo e alla portata di tutti è la sua gratuità, nel senso che la versione base, senza tanti fronzoli, viene resa liberamente disponibile a chiunque senza limitazioni di tempo. Già con questa versione è possibile creare applicazioni completamente funzionanti e per qualsiasi sistema. La vera limitazione che viene applicata e per cui bisognerebbe acquistare la versione a pagamento è la dimensione degli applicativi. Nella versione base, infatti, si possono solo creare applicazioni di piccole dimensioni e non è possibile utilizzare librerie esterne a quelle già incluse in Xamarin Studio. Se però non si mira a creare un software con un numero eccessivo di righe di codice, si dimostra un valido strumento per chi si vuole cimentare nella creazione.

Passiamo ad analizzare a cosa si può puntare sfruttando questo IDE e quali sono gli strumenti a nostra disposizione. È importante focalizzare l'attenzione sul fatto che Xamarin nasce per lo sviluppo di applicazioni mobile, quindi permette di creare, anche attraverso template predisposti, applicativi per Android, iOS e Windows Phone. L'idea di base è incentrata sull'ambiente mobile ma le possibilità fornite dall'IDE sono innumerevoli, infatti è possibile creare e pubblicare sull'apposito store anche applicazioni per Mac OS X.

CAPITOLO 3: EVENT INSPECTOR

Dopo aver analizzato in forma generale ciò che lo standard IEEE 1599 e il progetto Mono offrono, possiamo giungere finalmente al succo dell'elaborato in esame, vale a dire il Framework IEEE 1599, o più precisamente la funzione adibita all'analisi degli eventi all'interno di un file XML.

Nel corso del capitolo affronteremo l'analisi innanzitutto dell'aspetto grafico con gli elementi che compongono l'interfaccia, passeremo poi all'esame dello studio dello spine e termineremo la trattazione con la ricerca di un evento utilizzando i comandi messi a disposizione dal Framework.

L'idea è nata con lo scopo di rendere disponibile a chiunque un software in grado di agire su XML, file audio, midi e partiture, permettendo un'interazione tra gli stessi così da analizzare un brano nella sua completezza e favorire gli studi da parte dei musicologi, degli appassionati e degli studenti presso il LIM.

Ed è proprio al LIM che tutto ha origine, a seguito della definizione dello standard IEEE 1599 si è cercata una soluzione per permettere agli utenti di sfruttarne tutte le capacità e poterlo arricchire a loro volta. La risposta a queste esigenze venne individuata nella creazione di un Framework compilato tramite .NET per poter essere utilizzato in ambiente Windows.

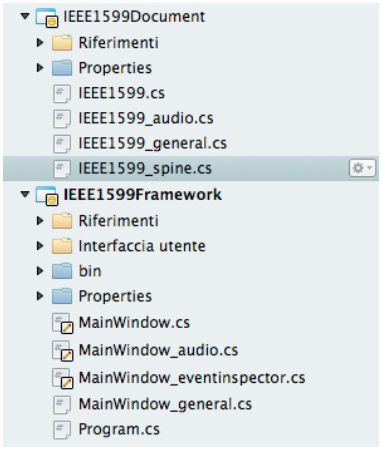
Negli ultimi tempi questa soluzione si è rivelata però poco funzionale perché non permetteva l'utilizzo da parte degli utenti Unix e Mac OS X, dando la possibilità di utilizzarlo unicamente agli utenti di Windows.

Si rivela necessario, quindi, riscrivere il codice del framework rendendolo multipiattaforma e permettere a chiunque di usufruirne in piena libertà, senza dover perciò installare una nuova partizione o una macchina virtuale montante un sistema operativo Windows.

Il progetto Mono viene in contro a questa esigenza, dando la possibilità di sfruttare le stesse capacità del precedente Framework migliorandole e arricchendo il programma con nuove funzionalità.

3.1 INTRODUZIONE AL FRAMEWORK

Per capire esattamente come e su cosa andrà ad agire il nostro Event Inspector, è necessario capire prima la struttura del Framework, dato che la funzione in esame è solo una parte di un progetto ben più grande.



Dalla figura a fianco si nota chiaramente una scissione del contenuto, da una parte il codice relativo alla manipolazione e all'analisi di documenti XML nello standard IEEE 1599, dall'altra quello relativo al funzionamento vero e proprio del Framework.

Fig.5: Lista dei codici sorgente

Più precisamente, il gruppo contenente tutti i sorgenti che iniziano con la dicitura IEEE1599 contengono le dichiarazioni delle classi e dei metodi che agiscono su di esse per rappresentare informazioni precise, permettendo così al programmatore di sapere con esattezza dove reperirle e come utilizzarle.

Nel primo sorgente (IEEE1599.cs) troviamo tre struct, una per i parametri identificativi di ciascun evento, una per le informazioni sugli autori dell'opera e una sulle tracce utilizzate per la mappatura audio. Subito a seguire compare la prima, vera classe, fondamentale per il funzionamento di tutto il Framework perché permette la gestione completa del documento XML standardizzato, agendo sull'apertura, sul salvataggio, sulla modifica e sulla cancellazione sia di singoli nodi sia di tutto il programma. Fornisce inoltre un metodo di identificazione della validità allo standard del documento, fallendo tale verifica il documento non verrebbe aperto comportando la chiusura forzata del programma.

Il secondo nell'elenco è quello riguardante la mappatura audio (IEEE1599_audio.cs), il quale fornisce metodi per la creazione di liste contenenti i differenti formati audio disponibili per l'importazione e l'utilizzo. Permette inoltre di creare una lista di tutte le sorgenti importate e le informazioni di codifica, durata e temporizzazione ottenute direttamente dalla scansione del layer audio all'interno del documento.
Il sorgente subito a seguire (IEEE_general.cs) reperisce, attraverso una serie di metodi, le informazioni presenti all'interno del layer general e le visualizza nella schermata relativa presente nel Framework, permettendo all'utente di cancellare e modificare in base alle proprie esigenze il contenuto. Le modifiche apportate andranno ad agire sul documento originale e verranno applicate unicamente al salvataggio da parte dell'utente sempre attraverso il programma.

Ma la parte che sicuramente ci fornisce le informazioni più importanti per la funzione in esame è l'ultimo file, quello riguardante gli eventi all'interno dello spine (IEEE1599_spine.cs). Nonostante sia più scevro dei precedenti, il metodo che fornisce una lista di tutti gli eventi memorizzando per ciascuno ID, timing e hpos, sarà di notevole utilità per i nostri scopi a seguire.

Se i primi sorgenti permettono la gestione di base dei documenti XML, il secondo insieme fornisce le funzioni più ricche, andando ad agire anche sull'aspetto grafico vero e proprio del programma, tenendo in considerazione ulteriori metodi attivabili attraverso, ad esempio, un semplice click da parte dell'utente.
Anche in questo caso tutti i sorgenti sono identificati dal nome del proprio spazio di lavoro (MainWindow) descrivendo immediatamente a seguire la sezione su cui andranno ad agire, che sia per l'elaborazione del layer General, oppure sulla Mappatura Audio o ancora sull'Event Inspector.
È importante specificare che, al momento, solo queste funzioni sono disponibili ma si è già all'opera per la creazione di numerose altre funzioni, quale ad esempio la gestione del layer Notational attraverso la Mappatura Grafica.

3.2 GRAPHICAL USER INTERFACE

Un software applicativo messo a disposizione del pubblico, di qualsiasi tipo ed età, deve rispondere alle caratteristiche di semplicità e immediatezza. Deve quindi rappresentare le informazioni necessarie al funzionamento e i comandi che ne permettono l'interazione in maniera chiara, facendo attenzione che non sia causa di confusione o fraintendimenti.

A questo scopo, per la creazione del framework si è deciso di adoperare una struttura particolarmente minimale, con pochi comandi e poche schermate.

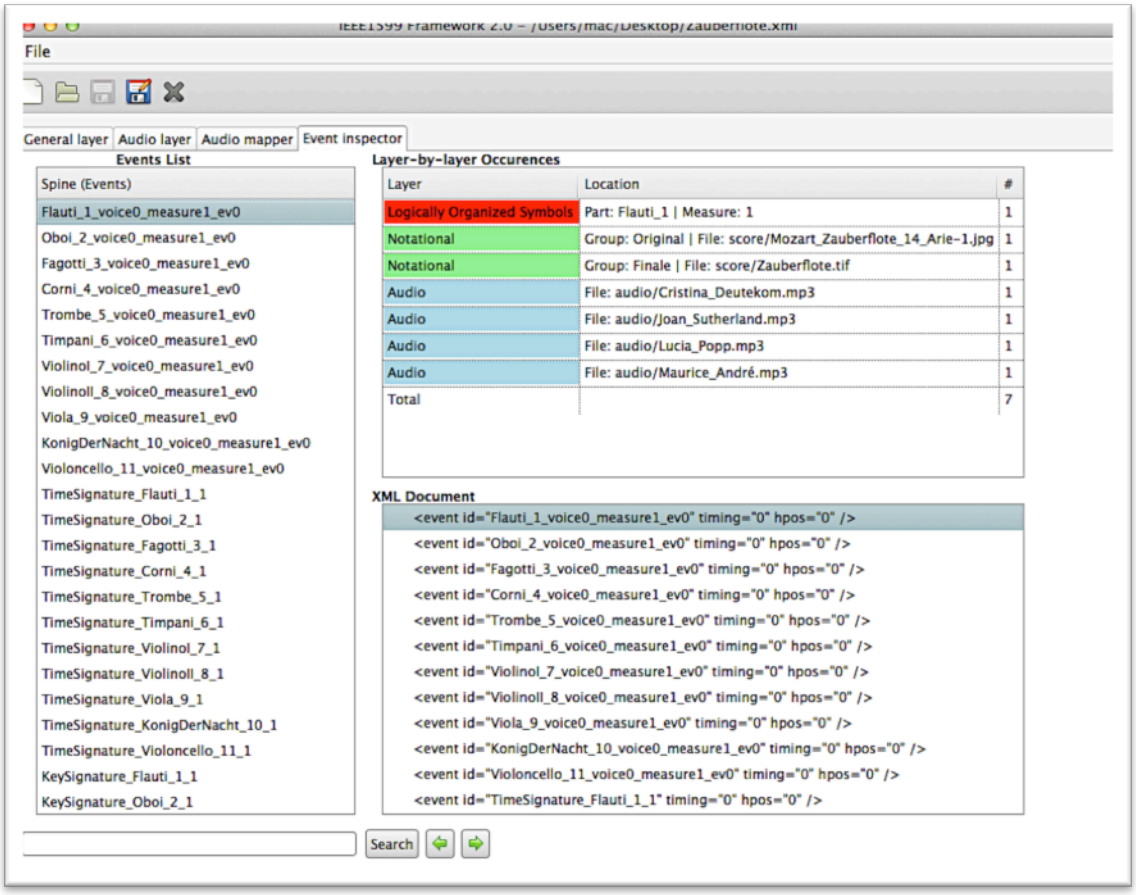


Fig.6: Interfaccia grafica dello strumento Event Inspector

Il tutto è stato creato sfruttando l'UI (User Interface) fornita da Xamarin Studio e dei relativi strumenti.

Inizialmente si è creata una nuova label contenente tutte le funzioni relative all'Event Inspector, così come sono state create le rispettive Label per il General Layer (contente a sua volta le informazioni generali del file XML con la possibilità di modificarle, cancellarle e inserirne di nuove), per l'Audio Layer (visualizzante i file audio collegati con il file XML), l'Audio Mapper (ancora da implementare, il cui

scopo è quello di permettere all'utente di mappare le partiture mediante i file audio importati).
A seguire è stata divisa l'interfaccia in differenti celle in cui inserire, in un secondo momento, gli strumenti necessari all'esecuzione.
Sono state quindi create tre differenti Treeview:

- Una contenente tutti gli eventi individuati all'interno dello spine (`treeviewSpineEvents`);
- Una contenente il numero di occorrenze dei diversi eventi per ciascun layer del file XML (`treeviewOccurences`);
- L'ultima per contenere l'intero codice XML (`treeviewShowXML`).

La scelta di sfruttare questo strumento si è basata sul fatto che permette di selezionare ciascuna riga, necessario per le funzioni che descriveremo in seguito. Per lo stesso motivo si è deciso di utilizzare una Treeview per visualizzare il codice XML e non una semplice casella di testo.
Tutti e tre questi strumenti sono stati inclusi in una finestra a scorrimento così da permettere lo scrolling di tutte le tabelle, visto il notevole contenuto presente nelle stesse.
In basso troviamo, infine, una casella di testo e tre pulsanti. La prima verrà utilizzata come barra di ricerca, in cui l'utente potrà digitare l'evento di suo interesse e a seguire lo potrà individuare nella relativa tabella cliccando sul comando "Search".
Dato che un evento è presente più volte all'interno del file XML, i due pulsanti "Precedente" e "Successivo", identificati attraverso le rispettive frecce, saranno necessari per selezionare e visualizzare le righe contenenti gli eventi e scorrere più velocemente il codice.

Nei precedenti capitoli abbiamo discusso, seppur brevemente, di GTK+ e del fatto che venga utilizzato per creare interfacce grafiche. Nella creazione del nostro Framework assumerà un aspetto rilevante proprio per gli strumenti che mette a disposizione.
Esamineremo quindi alcune istruzioni utilizzate all'interno del codice in esame con lo scopo di comprende ciò che si trova alla base di quanto esposto nel righe precedenti.
Dalla sezione adibita all'Event Inspector dobbiamo passare necessariamente a quella relativa alla MainWindow, la pagina principale contiene appunto le informazioni che verranno ereditate da tutte le rispettive finestre contenenti le funzioni del Framework.

Aprendo il sorgente MainWindow.cs le prime righe che attirano la nostra attenzione sono sicuramente le inclusioni delle librerie e degli assembly che permettono al programmatore di omettere la definizione dell'assembly utilizzato ogni volta che si specifica un comando appartenente ad esso.

Proseguendo con l'analisi del codice si possono notare differenti sezioni in cui compare più e più volte il termine Gtk, quelli saranno i metodi che andremo ad esaminare. Il primo tra questi che ci troviamo ad affrontare è `Gtk.Window` che, come è facile intuire, permette la creazione di una nuova finestra la quale potrà contenere o meno dei widget, nel caso della MainWindow verrà aggiunta una specifica, vale a dire `Gtk.WindowType.Toplevel`, la quale indica il fatto che la finestra in questione sarà ad un livello superiore alle altre e verrà identificata come la principale.
Nel momento in cui viene eseguito il framework si andrà ad attivare la funzione `Build()`la quale richiamerà una serie di parametri creati in automatico dalla UI e impostati all'atto della creazione o della modifica della stessa.
Una volta costruita l'interfaccia vengono eseguiti i metodi per l'aggiunta dei contenuti, distinti all'interno del codice tramite commenti per riuscire a distinguere ciò che viene rappresentato in ogni label e quindi in ogni strumento del Framework.
Attraverso il designer di Xamarin abbiamo creato, come descritto precedentemente, le varie Treeview che andranno a contenere le informazioni di nostro interesse. Ma in questo modo abbiamo solamente comunicato al programma che in quella zona è presente un oggetto Treeview, senza alcun riferimento su cosa debba esserci scritto, quante righe e quante colonne debba avere. Tutto ciò verrà descritto utilizzando i metodo forniti da GTK+. Dato che non è nel nostro interesse studiare il funzionamento completo del Toolkit andremo ad esaminare solamente i metodi di nostro interesse, e cioè quelli che entreranno in funzione al momento dell'avvio dell'Event Inspector.

TreeviewSpineEvents

Partiamo dalla prima in ordine di apparizione all'interno dell'interfaccia e cioè quella riservata alla rappresentazione degli eventi presenti nello spine. Vogliamo creare un'unica colonna in cui ogni riga contenga il nome di ciascun evento. Andiamo quindi ad esaminare, uno per uno, i metodi presenti:

```
Gtk.TreeViewColumn EventInspectorEventsList=new Gtk.TreeViewColumn();
```
Viene creata una colonna di tipo `TreeViewColumn` il cui nome `EventInspectorEventsList` permette di capire subito che si tratta di una lista di eventi.

```
Gtk.CellRendererText SpineEventsCell = new Gtk.CellRendererText();
```
Si imposta il rendering[7] del testo per ciascuna cella, nel caso non vengano definiti dei parametri specifici si utilizzeranno le impostazioni di default per il testo.

```
treeviewSpineEvents.AppendColumn(EventInspectorEventsList);
```
A seguito della creazione di una nuova colonna è possibile aggiungerla alla Treeview così che possa essere visualizzata all'interno dell'interfaccia.

```
treeviewSpineEventsListStore = new Gtk.ListStore (typeof(string));
```
Viene creata una ListStore che permette, come dice il nome, un immagazzinamento delle informazioni sotto forma di lista. Nel momento della creazione devono essere identificati i tipi di informazione che verranno inseriti all'interno della treeview. Nel nostro caso sarà presente una sola colonna, quindi la ListStore conterrà solo una colonna composta da stringhe.

```
treeviewSpineEvents.Selection.Mode = SelectionMode.Single;
```
Attraverso questa istruzione specifichiamo la tipologia di selezione all'interno della Treeview, vale a dire quante righe l'utente può selezionare contemporaneamente. Nel nostro caso si è deciso di permetterla per una e una sola riga.
Oltre alla tipologia Single se ne possono scegliere altre:

- `None`: La selezione sulla treeview è disabilitata.
- `Browse`: È possibile scegliere all'interno di una lista uno e un solo contenuto
- `Multiple`: Si possono selezionare più righe contemporaneamente.

[7] Rendering è una parola inglese utilizzata per indicare l'aspetto di qualcosa, in questo caso si parla di testo quindi si riferisce al colore, alla tipologia (grassetto, corsivo, ecc…), alla dimensione e ad altre caratteristiche.

```
treeviewSpineEvents.Model = treeviewSpineEventsListStore;
```

La treeview principale viene impostata sul modello della `ListStore`, ogni contenuto sarà identificato da un path ben distinto e quindi sarà possibile raggiungere determinati elementi basandosi su questa proprietà. Ciò si rivelerà particolarmente utile durante il procedimento di programmazione del framework.

```
treeviewSpineEvents.CursorChanged += new EventHandler(OnTreeviewSpine
EventsCursorChanged);
```

Un metodo molto interessante di elevata importanza dato che permette di creare e gestire eventi all'interno della treeview. In questo caso viene richiamato il metodo `OnTreeviewSpineEventsCursorChanged` ogni qual volta la selezione viene spostata su un nuovo contenuto della treeview, il che può avvenire o cliccando col mouse su una riga, o spostando la selezione con le frecce direzionali o ancora automaticamente qualora venga selezionata una riga tramite dei metodi.

TreeviewOccurences

Proseguendo in senso orario ci imbattiamo nella treeview che visualizzerà le occorrenze di ciascun evento all'interno del codice XML, identificando il numero di volte in cui compare nei diversi layer.

Per la creazione si è deciso di seguire il codice precedente come esempio, quindi creando una colonna per ciascun parametro da visualizzare includendo i risultati all'interno di determinate Liste.

Nel dettaglio:

- `EventInspectorLayerList:` Lista dei Layer in cui compare l'evento;
- `EventInspectorLocationList:` Specifica in quale sezione all'interno del Layer si trova;
- `EventInspectorNumberOccurencesList:` Numero di volte in cui compare nella sezione definita dal parametro precedente.

```
treeviewOccurences.EnableGridLines = TreeViewGridLines.Both;
```

Una proprietà non comparsa in precedenza ma che si è deciso di utilizzare in questo momento è identificata dall'abilitazione delle righe sulla Treeview. Si tratta di una proprietà che si è voluta utilizzare col solo scopo estetico per rendere più chiara la

visualizzazione dei risultati. Nel nostro caso sono state attivate sia le righe orizzontali che verticali, dando così maggior risalto ai differenti contenuti.

```
treeviewOccurencesListStore = new Gtk.ListStore (typeof(string), type
of(string), typeof(int));
```

Come si diceva nella Treeview precedente, nella creazione di una ListStore è necessario identificare i tipi di informazioni che andranno a riempire ogni singola colonna. Nel primo caso abbiamo visto solamente la tipologia String, visto che dovevano essere visualizzati unicamente gli eventi. Ora abbiamo tre tipologie di dati da visualizzare, cioè Layer e Location che sono informazioni composte da una serie di caratteri di lunghezza variabile e perciò identificati come String, mentre NumberOccurences sarà un numero intero positivo quindi di tipo Int.
Nel caso si volesse cambiare l'ordine delle colonne sarà necessario modificare anche i parametri nella creazione della ListStore.

TreeviewShowXML

Arriviamo quindi all'ultima Treeview in esame, per quanto riguarda l'ambito puramente grafico. La sua funzione sarà quella di visualizzare il codice XML che si vorrà analizzare e di mostrare quali righe contengano un determinato evento.
I metodi, le proprietà e i parametri utilizzati sono analoghi a quelli precedentemente descritti.

```
treeviewShowXML.CursorChanged += new EventHandler(OnTreeviewShowXMLCu
rsorChanged);
```

Apriamo solo una piccola parentesi su questo metodo per porre l'attenzione sul fatto che, anche in questo caso, si è deciso di gestire l'evento relativo allo spostamento del cursore. Analizzeremo più dettagliatamente questa funzione quando parleremo dell'esecuzione del metodo `OnTreeviewShowXMLCursorChanged.`

3.3 STUDIO DELLO SPINE

Dopo aver abbondantemente discusso e analizzato gli strumenti utilizzati durante la creazione dell'elaborato e le basi su cui si fonda, è giunto il momento di esaminare il codice dell'Event Inspector vero e proprio.

Esaminiamo, in ordine di apparizione, i differenti metodi presenti all'interno del codice.

```
private void compileEventInspector(){
        if (document == null)
            return;
        row = new ArrayList ();
        events = new ArrayList ();
        compileSpineEventsList();
        compileXML ();
}
```

Viene richiamato nel momento in cui si apre un nuovo documento all'interno del Framework. Nel caso in cui il documento non sia valido o sia vuoto il metodo termina la sua esecuzione e l'applicativo viene chiuso, in caso contrario vengono avviati altri due metodi, più precisamente `compileSpineEventsList()` e `compileXML()` che descriveremo nei paragrafi successivi.

Si inizializzano, inoltre, due ArrayList, vale a dire `row` ed `events` anch'essi verranno spiegati più dettagliatamente nelle prossime pagine.

```
private void compileSpineEventsList(){
        treeviewSpineEventsListStore.Clear();
        foreach (SpineEventStruct spineevent in document.Logic_spine_
events()) {
          treeviewSpineEventsListStore.AppendValues (spineevent.Id);
          events.Add (spineevent.Id);
        }
        if (document != null && !document.sSaved)
            this.Title = "IEEE1599 Framework 2.0-"+document.Path+"*";
}
```

Il primo metodo richiamato andrà a riempire la Treeview adibita alla creazione della lista degli eventi.

La Treeview, innanzitutto, deve essere liberata dal contenuto e pulita ogni volta che viene aperto un nuovo documento. Se non venisse utilizzata questa funzione gli eventi presenti nel documento XML andrebbero ad aggiungersi alla lista già presente, comportando così una notevole confusione all'interno del programma. Analizzando attentamente il ciclo foreach possiamo vedere che viene creato un nuovo oggetto di tipo SpineEventStruct (quindi una struttura comprendente le informazioni di ogni evento) a cui corrisponde un evento nello spine identificato all'interno del documento.

SpineEventStruct è dichiarata all'interno del file IEEE1599_spine.cs e contiene le informazioni di base di ciascuno evento, vale a dire "Id", "Hpos" e "Timing".

Attraverso la funzione AppendValue l'ID dell'evento viene aggiunto in una riga all'interno della Treeview, inoltre lo stesso ID viene aggiunto all'Arraylist "events".

Un arraylist risulta necessario ai nostri fini, poiché, dato che non è possibile stabilire un numero massimo standard di eventi per ciascun file XML, ci occorre un array di dimensione variabile, e l'arraylist può contenere un numero qualsiasi di elementi, identificati da un indice univoco.

```
private void compileXML(){
        string line;
        StreamReader file = new StreamReader(document.Path);
        treeviewShowXMLStore.Clear();
        while((line = file.ReadLine()) != null){
            treeviewShowXMLStore.AppendValues(line);
            row.Add (line);
            if (line.Contains("</spine>"))
                spinepres=1;
            if (line.Contains("</los>"))
                lospres=1;
            if (line.Contains ("</notational>"))
                notpres = 1;
            if (line.Contains ("</performance>"))
                perfpres = 1;
            if (line.Contains ("</audio>"))
                audiopres = 1;                                          }
        InitalizeLayers ();
        if (document != null && !document.IsSaved)
          this.Title = "IEEE1599 Framework 2.0 -"+document.Path+"*";}
```

Questo metodo, come quello precedente, viene avviato all'apertura di un nuovo documento XML da parte dell'utente.
Permette al Framework di mostrare all'interno dell'apposita Treeview le righe di codice e di selezionarle ad una ad una per dare un grado maggiore di interazione da parte dell'utente con il programma applicativo.
Viene creato un nuovo oggetto di tipo string al quale verranno assegnate le singole righe del codice XML e che verrà utilizzato per determinati metodi che esamineremo più avanti. Inoltre, si crea un oggetto che fungerà da lettore di flusso, ovvero, caricherà il documento di nostro interesse, identificato mediante il percorso salvato nell'oggetto Path, e lo esaminerà riga per riga.
Come per il metodo descritto in precedenza, all'apertura di un nuovo file la Treeview che mostra il codice XML verrà svuotata e ripulita permettendo così di inserire delle nuove righe ripartendo da zero.
Si esamina il documento riga per riga fino a che non si arriva all'ultima, corrispondente a una riga vuota, o per meglio dire, una riga nulla. Il funzionamento è analogo all'apertura di un file in C, dove il ciclo prosegue fino a che non incontra l'EOF, End of File. Ogni riga esaminata sarà assegnata alla stringa temporanea "line".
Viene inserita una nuova riga all'interno della Treeview, contenente quella corrispondente raggiunta dal ciclo while. Lo stesso contenuto viene aggiunto all'Arraylist "row", per lo stesso motivo spiegato precedentemente.
Un if esamina la riga, se all'interno è contenuta la chiusura di un layer, in questo caso lo spine, ne segnala la presenza ponendo a 1 il contatore delle presenze del layer in questione.
Potrebbe sorgere spontaneo il dubbio sul perché utilizzare un simile controllo.
Sfruttando delle variabili globali anziché locali, abbiamo modo di utilizzarle in ogni sezione di codice, senza doverle passare come attributi all'interno dei metodi o dichiararle ogni volta.
In questo caso, una volta terminato il ciclo while di scorrimento del documento, si esegue una verifica su ogni singolo contatore di presenza dei layer. Nel qual caso un layer non sia presente, e quindi il suo contatore sia rimasto a 0, si andrà ad imporre a 1 il contatore di chiusura del layer. Questo valore sarà importante per i nostri scopi e ne spiegheremo il funzionamento quando parleremo del fulcro vero e proprio del codice.

```
protected void OnTreeviewSpineEventsCursorChanged (object sender,
System.EventArgs e){
        TreeSelection selection = (sender as Gtk.TreeView).Selection;
        TreeModel model;
        TreeIter iter;

        if(selection.GetSelected(out model, out iter)){
            ev=model.GetValue (iter, 0).ToString();
        }
        ev=string.Concat(ev, "\"");

        if (ev != "")
            for (i=0; i<row.Count; i++){
            riga = row [i].ToString();
                if (riga.Contains (ev)) {
                    pos = i;
                break;
                }
            }
  treeviewShowXML.Selection.SelectPath (new TreePath (i.ToString()));
  treeviewShowXML.ScrollToCell (new TreePath (i.ToString ()), null,
false,0f,0f);
    searchEvent ();
    }
```

Arriviamo a uno dei metodi definiti in precedenza durante la creazione dell'interfaccia, vale a dire quello che regola cosa avviene una volta che il cursore subisce uno spostamento all'interno della Treeview contenente la lista degli eventi.
Vengono passati due parametri, vale a dire:

- `object sender`: che contiene le informazioni selezionate, sotto forma di oggetto;
- `System.EventArgs e`: in cui sono definite le proprietà della selezione;

Si dichiarano tre oggetti relativi, appunto, alla selezione. Più dettagliatamente troviamo un oggetto per la selezione vera e propria, uno per il modello della treeview e infine uno per l'iter.
TreeSelection permette la gestione, come dice il nome, delle selezioni. Tale oggetto viene creato automaticamente quando una nuova Treeview viene creata e non può esistere separatamente da essa. Vengono creati due differenti metodi, vale a dire

`get_selection` e `set_selection` che forniscono la capacità di selezionare un oggetto e di recuperare le informazioni a riguardo. Attraverso questa capacità è possibile inoltre monitorare lo stato di una selezione, cioè, se ad un certo punto essa dovesse cambiare il metodo emetterebbe una notifica.

TreeModel indica la struttura ad albero della treeview, in cui le informazioni sono disposte su colonne in maniera gerarchica. Nel nostro caso non avremo dei nodi padre e dei nodi figli ma unicamente nodi padre. I valori contenuti all'interno delle colonne saranno ben definiti, infatti ne abbiamo visto la creazione quando abbiamo parlato delle Treeview. Per ognuna di esse, venivano specificati i campi e le tipologie di dato che l'avrebbero riempita in futuro.

TreeIter specifica la posizione del cursore al momento della selezione. L'informazione selezionata è infatti raggiungibile attraverso questa informazione, che fornisce l'indirizzamento completo.

Se una riga viene selezionata si posiziona l'iter su di essa. I parametri `out model` e `out iter` stanno a indicare cosa viene restituito dalla selezione, quindi il modello della treeview e il percorso identificante l'informazione a cui sta puntando in quell'istante il puntatore.

Una volta stabilita la posizione interessata salviamo il contenuto della riga desiderata all'interno di un oggetto di tipo stringa.

Dobbiamo quindi ottenere il valore contenuto all'interno del modello alla colonna 0 della riga in questione. Dopo averlo ottenuto deve essere eseguito l'unboxing (come descritto nel capitolo inerente al C#) trasformando il tutto in un dato di tipologia string.

Abbiamo quindi salvato all'interno dell'oggetto "ev", dichiarato globalmente, l'evento di cui desideriamo ottenere le informazioni.

All'interno del ciclo for vediamo finalmente in azione le Arraylist di cui avevamo già parlato a più riprese nelle pagine precedenti. Al riguardo avevamo descritto tali oggetti come degli Array a dimensione variabile.

Nella porzione di codice che vogliamo esaminare prendiamo in considerazione l'Arraylist "row", che ricordiamo contenere tutte le righe del codice XML. In pratica si tratta del codice visto come una matrice.

Lo scopo del ciclo è di identificare la prima riga contenente l'evento selezionato, vedremo poi nelle prossime istruzioni a cosa servirà.

Controlliamo ogni posizione dell'array, salvando ogni volta il contenuto in un oggetto di tipo stringa denominato "riga", dopo aver eseguito anche in questo caso l'unboxing. Dopo di che controlliamo se effettivamente l'evento, salvato nell'oggetto "ev", è contenuto nella riga. In tal caso verrà salvata la posizione e si chiuderà il ciclo.
Le due istruzioni che seguono, non vanno più ad agire sulla Treeview contenente la lista degli eventi, ma bensì su quella adibita al codice del documento.
La prima istruzione permette la selezione di una determinata posizione, e qui entra in scena l'indice salvato in precedenza. Infatti, ogni riga all'interno della treeview ha una posizione, allo stesso modo dell'array, la scelta però di eseguire la ricerca su quest'ultimo invece che sulla treeview vera è propria è stata effettuata di fronte all'assenza di un metodo adeguato tra quelli forniti da Mono e perciò si è dovuto ripiegare in questo modo per arginare la mancanza.
La seconda istruzione si è rivelata necessaria poiché la selezione veniva eseguita correttamente ma non veniva eseguito in maniera automatica lo scrolling all'interno della finestra con inclusa la Treeview, perciò l'utente avrebbe dovuto cercare la riga selezionata manualmente, rivelandosi una soluzione molto scomoda e poco pratica, soprattutto in caso di documenti composti da numerose righe di codice. ScrollToCell si è osservato essere un'ottima soluzione dato che, inserendo i parametri di cui sopra, sposta in maniera automatica il cursore all'interno della finestra, visualizzando la riga di nostro interesse. Per precisare, i parametri richiesti sono, in ordine, il percorso della riga selezionata all'interno della treeview, la colonna in questo caso null visto che ne è presente una sola, un parametro booleano per definire se si desidera l'allineamento (in questo caso no, quindi false) e due parametri float per l'allineamento di riga e colonna (in questo caso pari a 0 visto che abbiamo deciso di non abilitarlo).
Viene infine avviato il metodo `searchEvent()`, che verrà richiamato più volte nel corso del framework ed è il fulcro dell'Event Inspector poiché mostra i risultati della ricerca di un evento all'interno del codice. Ne parleremo esaurientemente nel prossimo capitolo, lasciando spazio ad alcuni metodi che permettono funzioni simili a quelle viste in precedenza.

```
private void OnButtonSearchClicked(object o, System.EventArgs args){
   filter.Refilter();
   filter.VisibleFunc=new Gtk.TreeModelFilterVisibleFunc(FilterTree);
   treeviewSpineEvents.Model = filter;
}
```

Nel momento in cui il pulsante Search viene cliccato si avvia questo metodo, creato appositamente per permettere all'utente di filtrare gli eventi contenuti all'interno della Treeview apposita. In questo modo si possono visualizzare unicamente determinati risultati, permettendo così una ricerca più rapida ed efficiente.

Il filtro dovrà essere ogni volta resettato. Ogni volta che viene cliccato sul pulsante Search entrerà in funzione questo metodo che eliminerà tutte i parametri con cui tale filtro è stato impostato in precedenza, permettendo così di inserirne di nuovi.

Utilizzando `VisibleFunc` viene applicata la regola di visibilità del filtro, vale a dire quello che effettivamente verrà mostrato. Si crea una nuova funzione adibita a questo ruolo richiamando il metodo FilterTree che vedremo nelle prossime righe e che permetterà di stabilire cosa mostrare e cosa no dei contenuti presenti nella Treeview.

Viene in seguito impostata nuovamente la Treeview utilizzando come modello il risultato del filtro, così che possano essere visualizzati i risultati in maniera corretta.

```
private bool FilterTree (Gtk.TreeModel model, Gtk.TreeIter iter){
        string Event = model.GetValue (iter, 0).ToString ();
        if (entrySearch.Text == "")
            return true;
        if (Event.Contains(entrySearch.Text))
            return true;
        else
            return false; }
```

Tale metodo viene richiamato nell'esecuzione del filtro vero e proprio ottenendo in ingresso come parametri il modello della Treeview sulla quale filtrare e l'iter.

Viene fatta scorrere tutta la treeview riga per riga, e viene salvato il contenuto all'interno della variabile Event.

Si controlla innanzitutto se la casella di testo per la ricerca degli eventi è vuota oppure no; se è vuota si mostra tutto il contenuto della treeview, ad ogni riga il metodo restituirà True, se invece è presente un evento da ricercare si controllerà se è presente all'interno dell'iter e in tal caso si visualizza restituendo anche in questo caso True.

Se invece non fosse contenuto all'interno dell'iter verrà restituito False, impedendo così alla Treeview di mostrare tale riga e si passerà a controllare la prossima.

```
private void OnButtonNextClicked(object o, System.EventArgs args){
        if (ev != "")
            for (i=(pos+1); i<row.Count; i++){
                riga = row [i].ToString();
                if (riga.Contains (ev)) {
                    pos = i;
                    break;
                }
            }
      if (i == row.Count) {
            treeviewShowXML.Selection.SelectPath(new TreePath(pos.ToS
            tring ()));
            treeviewShowXML.ScrollToCell(new TreePath(pos.ToString())
            , null, false, 0f, 0f);
      }
      else {
            treeviewShowXML.Selection.SelectPath(new TreePath(i
      .ToString ()));
            treeviewShowXML.ScrollToCell (new TreePath (i.ToString ()
      ), null, false, 0f, 0f);
      }
}
```

Questo metodo entra in funzione quando viene cliccato il bottone rappresentato dalla freccia rivolta verso destra (sarà presente anche un comando per il bottone con la freccia rivolta verso sinistra, che svolgerà la funzione opposta).

Tale pulsante è necessario, poiché se un evento viene selezionato dalla lista il cursore verrà spostato all'interno del codice fino alla prima occorrenza, ma l'utente potrà voler scorrere il codice per visualizzare le altre. Questo pulsante permetterà lo scorrimento rapido fino alla posizione del prossimo evento. Alla prima ricerca verrà salvata, come abbiamo visto, la posizione all'interno dell'array comprendente tutto il codice XML per i motivi di semplicità e praticità descritti in precedenza.

Un ciclo for partirà proprio dalla posizione successiva a quella corrente e scorrerà l'arraylist fino alla fine del codice, incrementando il puntatore.

Nel momento in cui viene identificata un'altra riga contenente l'evento desiderato si salva la posizione e si chiude il ciclo.
Si controlla poi attraverso un if se l'indice è arrivato a fine array, e quindi non ha trovato nessun altro evento nelle righe successive, o se effettivamente l'evento è stato individuato. Nel primo caso si manterrà la selezione sulla riga in cui ci si trova e anche il cursore non verrà spostato. Se invece l'evento viene trovato si sposterà sia il cursore che la selezione alla riga esatta in cui si trova la prossima occorrenza.
Analogamente il comando per la ricerca dell'occorrenza precedente inizierà dalla riga subito prima a quella in cui ci si trova e il contatore verrà decrementato fino a raggiungere la posizione 0 dell'array, le restante istruzioni verranno ripetute.

```
protected void OnTreeviewShowXMLCursorChanged (object sender, System.
EventArgs e)
```

Metodo analogo al precedente, anche in questo caso viene gestito lo spostamento del cursore all'interno di una Treeview, in questo caso quella che mostra il codice XML, e le azioni che ne conseguono.
Come prima anche in questo caso vengono dichiarati gli oggetti inerenti all'iter, al modello e alla selezione di ogni singolo nodo della Treeview.

3.4 RICERCA EVENTI

In ultima analisi dobbiamo prendere in considerazione i metodi riguardanti la visualizzazione in formato tabellare dei risultati ottenuti dalla ricerca di un evento all'interno del codice.

Il metodo che verrà utilizzato e ricorrerà più volte sarà SearchEvent();

```
protected void searchEvent (){
        treeviewOccurencesListStore.Clear ();
        tempLayer = tempPart = tempMeasure = "";
        tempGroup = tempFile = "";
        tempGroupFile = tempPartMeasure = "";
        tempcountlos = tempcountnot = tempcountaudio = 0;
        totcount = 0;
        losend = spineend = notend = audioend=0;
        for (i=0; i<row.Count; i++){
            riga = row [i].ToString();
            if (riga.Contains("</spine>"))
                spineend=1;
            SearchLosOccurences ();
            if (riga.Contains("</los>"))
                losend=1;
            SearchNotationalOccurences ();
            if (riga.Contains ("</notational>"))
                notend = 1;
            SearchPerformanceOccurences ();
            if (riga.Contains ("</performance>"))
                perfend = 1;
            SearchAudioOccurences ();
            if (riga.Contains ("</audio>"))
                audioend = 1;
        }
  treeviewOccurencesListStore.AppendValues ("Total", ""  , totcount);
      if (document != null && !document.IsSaved)
         this.Title = "IEEE1599 Framework 2.0" + document.Path + "*";
}
```

Come prima operazione, vista già in precedenza più volte per le altre Treeview dovremo ripulire quella delle occorrenze dei suoi contenuti così da poterne inserire di nuovi.

Verranno inizializzate tutte le variabili che conterranno i contatori delle occorrenze e le descrizioni utili per un'analisi rapida di quanto viene mostrato. Si assegneranno poi, sfruttando nuovamente il metodo per l'inizializzazione dei layer, il valore rispettivo alla chiusura di ciascuno di essendosi, basandosi sulla presenza o meno all'interno del documento.

Mediante l'utilizzo di un ciclo for che scorre l'array contenente le righe del codice andiamo a salvare il contenuto di ciascuna di essere in una variabile temporanea di tipologia string. Verifichiamo poi se tale stringa contiene una chiusura di layer, se così fosse modifichiamo il contatore di termine del layer corrispondente ponendolo a "1". Eseguiamo poi la ricerca delle occorrenze per ciascun layer, tale ricerca verrà spiegata a seguire.

Non appena il ciclo giungerà a termine, e quindi si sarà esaminato l'intero codice, verrà aggiunta una riga all'interno della Treeview contenente il totale delle occorrenze, calcolato nel corso delle differenti ricerche.

```
private void SearchLosOccurences(){
    if (riga.Contains ("<part id="))
      tempPart = string.Concat("Part: ", FindSubstring (riga));
    if (riga.Contains ("<staff id="))
      tempPart = "Staff list";
    if (riga.Contains ("<measure"))
      tempMeasure=string.Concat(" | Measure: ",FindSubstring (riga));
    if ((losend == 0) && (riga.Contains (ev) && (spineend == 1))) {
        tempLayer = "Logically Organized Symbols";
        tempcountlos += 1;
        totcount += 1;
        if (tempPart.Contains("Staff"))
            tempMeasure="";
        tempPartMeasure = string.Concat (tempPart, tempMeasure);
        treeviewOccurencesListStore.AppendValues (tempLayer, tempPart
Measure, tempcountlos);
}
```

Partiamo dal layer LOS ed esaminiamolo nel dettaglio, i restanti metodi avranno lo stesso funzionamento di base, ma ovviamente gli output saranno differenti.

Per questa ricerca dovremo salvare quattro informazioni, vale a dire il Layer in cui è stato individuato l'evento, la parte e la misura all'interno della partitura e il numero di volte in cui compare.

L'idea che sta alla base della progettazione di questo metodo consiste nel valutare il codice nella sua struttura ad albero. Se, come nel nostro caso, ci troviamo all'interno del LOS, possiamo usare le nostre conoscenze per definire un codice funzionale. Come ben sappiamo possiamo avere due tipologie principali, vale a dire la Staff List, che contiene tutte le informazioni relative alla durata, le alterazioni di chiave, il tempo ecc… di ciascuna parte, oppure Part, contenente tutti gli oggetti musicali riconoscibili da un identificativo e dalla misura in cui compaiono.
Ciascuna Part contiene un sottolivello denominato Measure, il quale, a sua volta, contiene tutti gli eventi che ne fanno parte con la rispettiva durata.
Ricapitolando avremmo una struttura composta da: Parte, Misura, Eventi, che andranno a chiudersi in ordine inverso. Questa breve descrizione è necessaria per comprendere il funzionamento del codice di cui sopra.
Ogni volta che viene individuato l'inizio di una Parte il contenuto di tale stringa viene inserito in una variabile temporanea, allo stesso modo la Misura. Per entrambe verrà individuala la substringa per poter mostrare solo il contenuto interessato e non tutta la riga.

Esaminiamo di seguito velocemente tale funzione:

```
private string FindSubstring(string str){
        int FirstDelimit = str.IndexOf("\"") + "\"".Length;
        int LastDelimit = str.IndexOf ("\"", FirstDelimit);
        string FinalString = str.Substring(FirstDelimit, LastDelimit
- FirstDelimit);
        return FinalString;
    }
```

Si impostano due delimitatori, uno iniziale corrispondente alle prime virgolette della stringa e uno finale che, partendo dal quello salvato in precedenza esamina la stringa fino ad arrivare alla prima occorrenza del carattere corrispondente alle virgolette.
Viene infine estrapolata la stringa desiderata prendendo un numero di caratteri calcolato togliendo all'indice ottenuto tramite l'ultimo delimitatore, l'indice del primo delimitatore.

Prendiamo ad esempio la stringa relativa ad un'istanza grafica, in cui sono presenti numerose informazioni:

```
<graphic_instance file_name="score/Mozart_Zauberflote_14_Arie-1.jpg" file_format="image_jpg" encoding_format="image_jpg" position_in_group="1" measurement_unit="pixels">
```

Il primo delimitatore corrisponderà alla posizione 28, l'ultimo invece sarà alla posizione 67. Eseguendo la sottrazione otteniamo 39 caratteri, corrispondenti effettivamente alla stringa che ci interessa, vale a dire:

```
score/Mozart_Zauberflote_14_Arie-1.jpg
```

Chiudiamo questa piccola parentesi per tornare a parlare della ricerca delle occorrenze. Abbiamo detto che vengono salvate la Parte e la Misura, ora bisogna controllare se effettivamente è presente l'evento e per questo ci avvaliamo di un if che verifica se ci troviamo nel layer LOS, controllando le variabili di termine layer che abbiamo esaminato precedentemente. Se lo spine è terminato, quindi se spineend=0 verifico che tutti gli altri layer siano aperti, di conseguenza, visto che il documento viene letto dall'inizio alla fine, ci troveremo obbligatoriamente all'interno del layer interessato. Tale controllo avviene anche negli altri metodi, questo per conteggiare e rilevare esattamente l'evento presente in un determinato layer.

Se quindi siamo riusciti a verificare di essere nel los, esaminiamo anche che la riga contenga l'evento desiderato. In tal caso il Layer Temporaneo diverrebbe il "Logically Organized Symbols", il contatore del layer aumenterebbe di 1 e allo stesso modo il contatore delle occorrenze totali.

Nell'eventualità che si tratti di un evento relativo alla Staff List e quindi non contenente informazioni sulla misura in cui è presente, la variabile temporanea non avrà contenuto, mentre la parte verrà nominata "Staff List".

Tutte le informazioni verranno infine inserite all'interno di una nuova riga della Treeview.

3.5 LATO UTENTE

Per poter eseguire il framework sui differenti Sistemi Operativi si deve innanzitutto installare Mono sul proprio elaboratore, in seguito, una volta raggiunta la cartella in cui risiede l'eseguibile, bisognerà digitare da terminale (se non dovesse funzionare il prompt dei comandi usare il terminale di Mono) il seguente comando:

```
mono IEEE1599Framework.exe
```

Se dovessero sorgere problemi con le librerie sarà necessario eseguirlo facendo uso dei seguenti comandi sempre tramite terminale:

```
export
DYLD_FALLBACK_LIBRARY_PATH="/Library/Frameworks/Mono.framework/Ve
rsions/Current/lib:$DYLD_FALLBACK_LIBRARY_PATH:/usr/lib"

exec /Library/Frameworks/Mono.framework/Versions/Current/bin/mono
"/Users/mac/Desktop/IEEE1599 Framework
2.0/IEEE1599Framework/IEEE1599Framework/bin/Release/IEEE1599Frame
work.exe"
```

(il path assoluto del secondo comando dovrà essere modificato inserendo il percorso dove si trova effettivamente l'applicativo).

Verrà quindi avviata l'esecuzione dell'applicativo e sarà visibile la schermata principale, che permetterà l'apertura o la creazione di un nuovo documento XML (creazione non ancora implementata).

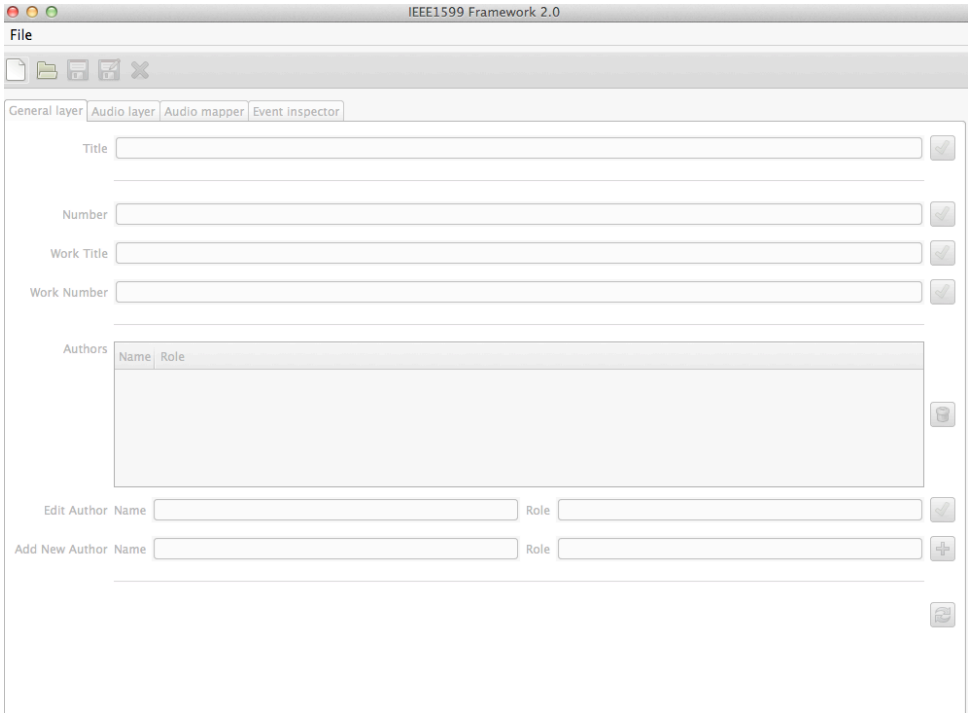


Fig.7: Finestra principale del Framework IEEE 1599 all'avvio

Apriamo dunque il documento che più ci aggrada all'interno del nostro computer, potremo anche scegliere dalla lista dei documenti recentemente aperti tramite il Framework.
Se il file XML è valido si aprirà senza complicazioni, in caso contrario il Framework verrà chiuso e sarà necessario riavviare tutto.
Nel caso in cui vada tutto a buon fine avremo la seguente schermata:

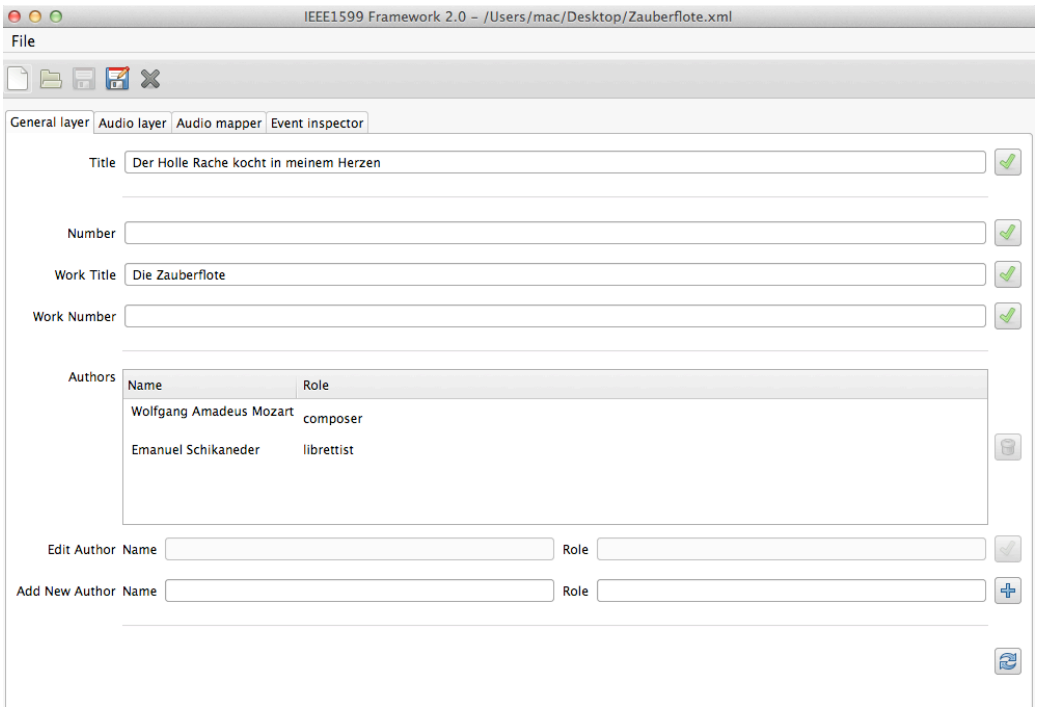


Fig.8: Finestra principale del Framework IEEE 1599 all'apertura di un file XML

Possiamo notare che vengono reperite le informazioni dal Layer Generale e saranno visualizzati nelle rispettive sezioni. Potranno quindi essere modificate o cancellate e tutte le variazioni apportate verranno poi salvate sul documento cliccando sul comando "Salva".

Al di sopra della schermata principale sono visibili dei comandi, che rimanderanno a differenti finestre relative alle funzioni del Framework. Cliccando sulla label nominata "Event Inspector" si aprirà la schermata relativa alla funzione di nostro interesse.
Tale finestra sarà la medesima rappresentata in precedenza quando si è parlato dell'interfaccia grafica.

CONCLUSIONI E SVILUPPI FUTURI

Il Framework IEEE1599 ha le potenzialità per diventare uno strumento fondamentale negli studi musicologici al fine di studiare in maniera approfondita alcuni aspetti differenti da quelli usuali.

Lo scopo dell'elaborato consisteva nell'impostare le prime funzioni all'applicativo, così da fornire la base ai programmatori per i possibili sviluppi futuri.

Qualora si volesse implementare una nuova funzione sarà possibile aggiungere una label identificativa e lavorare su quella nuova finestra, in maniera analoga a quanto è stato fatto per l'Event Inspector. Al riguardo sono già state prese in considerazione alcune aggiunte di questo strumento. Durante la scrittura dell'elaborato ci si è trovati di fronte a un dilemma riguardante la possibilità di modificare o meno il codice XML direttamente dalla Treeview apposita. Il problema principale consisteva nel fatto che, al fine della modifica, sarebbe stato necessario validare nuovamente il documento così da renderlo conforme al DTD standardizzato IEEE 1599. Ma, qualora non fosse stato fatto nella maniera corretta, la modifica e la validazione errata avrebbero potuto compromettere il documento in questione oppure causare il blocco dell'applicativo.

Si è deciso dunque di affrontare in futuro questo problema, esaminando la possibilità di inserire un comando per la validazione di un file XML all'interno del Framework, in maniera tale da poter eseguire tutte le operazioni direttamente da tale applicativo senza dover dipendere da altri software. In questo modo si aggiungerebbero le potenzialità del Framework MX a quello in esame, rendendolo maggiormente funzionale con la possibilità di utilizzarlo anche come editor di codice XML.

Un'ulteriore implementazione potrebbe essere la comunicazione tra Event Inspector e altri strumenti del Framework, per esempio il layer di mappatura grafica o quello di mappatura audio, nel senso che sarebbe interessante aggiungere una piccola finestrella di anteprima delle note relative agli eventi oppure un comando che, selezionando la riga apposita all'interno della Treeview delle occorrenze, mostri la sezione di partitura in cui è incluso l'evento o anche mandi in esecuzione il brano in cui l'evento è stato mappato. Attraverso questi sviluppi sarebbe quindi possibile ottimizzare le funzionalità del Framework riducendo i movimenti che l'utente dovrà eseguire all'interno di esso.

In conclusione si può dire che l'Event Inspector al momento risulta funzionante per quel che è stato progettato. L'elaborato si è rivelato proficuo sotto diversi punti di vista, tra cui la possibilità di fornire agli utenti un software multipiattaforma, open source, sfruttando le potenzialità di altrettanti applicativi gratuiti. In tal modo si è mantenuta la medesima etica di base, anche dal punto di vista dei linguaggi, garantendo un dispendio limitato di energia e istruzioni da parte del processore.

BIBLIOGRAFIA

[1]Goffredo Haus, Luca Andrea Ludovico, Adriano Baratè, *IEEE 1599: a New Standard for Music Education,* Edizioni Nuova Cultura, 2009.

[2]Goffredo Haus, Luca Andrea Ludovico, Adriano Baratè, *Music representation of score, sound, MIDI, structure and metadata all integrated in a single multilayer environment based on XML*, Information Science Reference, 2008.

[3]Denis L. Baggi, Goffredo Haus, *Music Navigation with Symbols and Layers: Toward Content Browsing with IEEE 1599 XML Encoding,* Wiley-IEEE Computer Society Press, 2013

[4]Eric Lease Morgan, *Getting Started with XML: A Manual and Workshop,* 2004

[5]Goffredo Haus, Luca Andrea Ludovico, Adriano Baratè, *Music Analysis and Modelling through Petri Nets,* Springer, 2005.

[6]Edd Dumbill, Niel M. Bornstein, *Mono: A Developer's Notebook,* O'Reilly Media, 2004.

[7]Don Box, Chris Sells, *Essential .NET, Volume I: The Common Language Runtime,* Addison-Wesley Professional, 2002

[8]M. De Benedittis, *Manuale di C# 2005 Guida alla programmazione di base e all'ambiente di sviluppo,* Hoepli, 2007.

[9]Andrew Krause, *Foundations of GTK+ Development*, Apress, 2007.