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

Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

PLUGIN MULTIPIATTAFORMA PER SLIDESHOW A PARTIRE DA DOCUMENTI IEEE 1599

Elaborato finale di:

Francesco Lodovici

Matricola 738909

Relatore:

Prof. Luca Andrea Ludovico

Correlatore:

Dott. Adriano Baratè

Anno Accademico 2010 – 2011

Indice

0. Introduzione 1
1. Descrizione multistrato dell'informazione musicale 3
1.1 Concetti chiave dello standard IEEE 1599 3
1.2 I Layers 4
1.2.1 Il layer General 6
1.2.2 Il layer Audio 8
2. Tecnologie utilizzate: Mono, Gtk#, SDL.NET 9
2.1 Origini di Mono, derivazione da .NET 9
2.1.1 Differenze tra Mono e .NET 10
2.1.2 Monodevelop 10
2.2 Origini di Gtk+ e Gtk# 10
2.3 SDL 11
2.3.1 SDL.NET 11
2.3.2 - SdlDotNet.Audio 12
3. Plugin multipiattaforma per slideshow a partire da documenti IEEE 1599 14
3.1 TripleEye e TripleEyeEngine 14
3.1.1 Related Files 15
3.1.2 Clip Audio/Video 16
3.2 Individuazione delle classi e degli algoritmi necessari 21
3.2.1 Fetching delle informazioni relative ai related files 21
3.2.2 La classe MediaElement 24
3.2.3 La classe relatedFileWindow 24
3.2.4 L'algoritmo di creazione dello slideshow 26
3.2.5 Distruzione dello slideshow 27
3.3 Sviluppo dell'interfaccia grafica 28
3.3.1 Descrizione della GUI 28
3.3.2 Aspetti implementativi dell'interfaccia grafica 30
4. Test dell'applicazione e analisi dei risultati 34
4.1 Test numero 1, singolo file audio, precisione del software 34
4.2 Test numero 2, più file audio, con temporizzazioni diverse dei related files 35
5. Conclusioni, considerazioni, sviluppi futuri 37
6. Bibliografia e sitografia 38
6.1 Bibliografia 38
6.2 Sitografia 38
7. Ringraziamenti 39

0. Introduzione

Da sempre vivo di musica. La mia passione per la musica in tutte le sue possibili sfacettature, in tutti i suoi aspetti, matematici e non, nasce davanti al pianoforte negli anni dell'infanzia, quando ancora tutti gli aspetti algoritmici dell'atto compositivo mi erano estranei, e l'unico elemento che mi guidava era l'istinto, la curiosità di trovare "tasti che suonassero bene insieme".

La passione per l'informatica invece è più recente. Ricordo con un sorriso il mio primo computer, un Pentium III assemblato alle medie insieme al classico amico "smanettone". Ricordo i primi esperimenti da adolescente con Cubase, privo di qualsivoglia nozione sull'audio, tantomeno di concetti quali la differenza tra suono analogico e suono digitale, quantizzazione, rumore. Semplicemente, attaccavo il microfono, armavo la traccia, e registravo, affascinato dalla possibilità di interagire con i suoni, riascoltarli, modificarli, combinarli. Per la prima volta l'esperienza musicale diventava per me non solo un gesto istintivo ed estemporaneo fine a se stesso, ma un modo per comprendere e razionalizzare l'atto compositivo. Nel corso degli anni ho poi spostato il mio interesse sempre più verso l'informatica in quanto tale. Le prime esperienze negli studi di registrazione sono state formative ed importanti, sì, ma quantomeno deludenti. Purtroppo o per fortuna gli studi da me frequentati erano, nella maggior parte dei casi, non luogo di ricerca e analisi, ma luoghi dove si produce. Per forza di cose l'interesse nei confronti dell'utilizzo di un elaboratore in un contesto di produzione si focalizza più sul fine (il prodotto) che sul mezzo (le macchine utilizzate per produrlo). Per fare un esempio banale, nella maggior parte dei software dedicati al recording/mixing/editing esistono svariati algoritmi di rimozione del rumore. Ma pochi degli utilizzatori di tali algoritmi in un contesto di produzione si pongono il problema di come funzionano. E la mia curiosità restava incolmata. Poi ho scoperto dell'esistenza di questo corso. E durante questi anni ho appreso nozioni fondamentali in svariati campi, ho scoperto la programmazione e come essa possa essere stimolante tanto quanto l'atto creativo. Scoprire un algoritmo, vederlo funzionare, correggerlo là dove sbaglia, porta la stessa sensazione di soddisfazione che un musicista prova nel riascoltare la propria composizione. Ho avuto inoltre la fortuna di incontrare compagni e professori nei cui occhi e nelle cui parole ho visto e sentito lo stesso interesse ed entusiasmo che mi hanno portato fino a qui.

Il presente lavoro, conclusione di un percorso formativo articolato e multisfacettato, scaturisce dal mio interesse verso la programmazione evolutosi durante questi anni, e verso il progetto IEEE 1599, che reputo quantomeno affascinante e pionieristico, e rappresenta solo una piccolissima parte di un progetto di ben più ampio respiro. In particolare il seguente elaborato si propone di illustrare le possibilità del framework Mono, implementazione open source del framework .NET di Microsoft,

nella realizzazione di un programma (attualmente stand-alone, ma pensato per diventare un plugin del visualizzatore di documenti IEEE 1599 correntemente in sviluppo presso il Laboratorio di Informatica Musicale) multipiattaforma, che permetta di visualizzare i file immagine contenuti nel campo <related_files>, all'interno del layer General, in finestre temporizzate indipendenti. Dopo un primo capitolo introduttivo al framework MX e ai concetti fondamentali di codifica dell'informazione musicale multistrato, seguono rispettivamente un capitolo sulle tecnologie utilizzate, e il capitolo sullo sviluppo vero e proprio del software. Il quarto capitolo è riservato ad un'analisi del funzionamento del software, per verificarne la correttezza operativa. Seguono le conclusioni e le riflessioni su eventuali sviluppi futuri.

1. Descrizione multistrato dell'informazione musicale

In questo capitolo si illustra, in maniera funzionale alla comprensione del capitolo relativo al software, lo stardard conosciuto con il nome di IEEE 1599. IEEE 1599 è un formato basato sull'XML, sviluppato a partire dal 2000 presso il LIM (Laboratorio di Informatica Musicale - Dipartimento di Informatica e Comunicazione - Università degli Studi di Milano), che ha come scopo la descrizione e sincronizzazione di una serie eterogenea di contenuti musicali. Durante la prima fase del progetto sono stati analizzati e valutate le caratteristiche di tutti formati già esistenti volti alla descrizione della musica, in particolare SMDL, NIFF e XML. Seguirono nel 2001 incontri con altri gruppi di ricercatori in ambito accademico e commerciale. Un primo prototipo del formato è stato realizzato nel 2002, sotto il nome di MAX (Musical Application using XML). Questo prototipo è stato discusso nel corso di MAX 2002, la prima conferenza internazionale su applicazioni che usano l'XML, organizzata da IEEE CS TC. Nel Luglio 2008 tale formato è stato riconosciuto appunto come standard internazionale, sotto il nome di IEEE 1599.

1.1 Concetti chiave dello standard IEEE 1599

L'obbiettivo finale di tale progetto è quello di fornire all'utente una rappresentazione altamente integrata della musica, laddove contenuti multimediali di varia natura possano essere fruiti nel loro insieme.

All'interno dello stesso file tutti i simboli musicali, le tracce audio, metadati di catalogo, testi ed elementi grafici inerenti una singola parte musicale sono collegati e sincronizzati reciprocamente, in una struttura multi-strato che supporta la codifica in diversi formati ed un numero variabile di elementi per ogni strato.

Le caratteristiche salienti di tale formato possono essere schematizzate nei seguenti punti:

- ricchezza nella descrizione dei contenuti multimediali relativi allo stesso brano musicale. Tutti i contenuti, siano essi simbolici, testuali, grafici, audio, video, possono essere codificati in un unico documento.
- possibilità di descrivere e collegare tra loro media dello stesso tipo, ovvero molte performance dello stesso brano, o diversi spartiti appartenenti a diverse edizioni.
- pieno supporto per la sincronizzazione tra contenuti basati sul tempo. In un player dedicato, i contenuti audio e video possono essere sincronizzati con l'avanzamento dello spartito mentre la musica è riprodotta, anche cambiando fra una particolare esecuzione e un'altra.

- facile interazione con i contenuti musicali. In un navigatore IEEE1599 l'utente può selezionare una parte dello spartito e l'audio viene automaticamente spostato al punto corrispondente.

1.2 I Layers

Per descrivere, collegare e sincronizzare le diverse informazioni musicali, è stata ideata una architettura multi-livello, costituita di 6 diversi strati (da qui in avanti layer): General, Logic, Structural, Notational, Performance, Audio. Tutti i layer, fatta esclusione per il General, sono collegati insieme dallo Spine, una struttura astratta collocata all'interno del layer Logic che rappresenta e gestisce la relazione spazio-temporale intrinseca nella musica. Ognuno dei vari layer può contenere al proprio interno diverse descrizioni del brano musicale, ad esempio il layer Audio può contenere diverse registrazioni dello stesso brano codificate in diversi formati (WAV, MP3,...) oppure il layer Structural può fornire analisi differenti dello stesso brano.
Vediamo adesso una breve panoramica dei vari layer, cui seguirà un approfondimento sui layer General e Audio, in quanto fulcro dell'elaborato.

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

Lo schema seguente rappresenta l'interazione tra i diversi layer. Si noti come il layer Logic, e in particolare lo Spine, rappresenti il vero e proprio motore di sincronizzazione di tutto il framework.

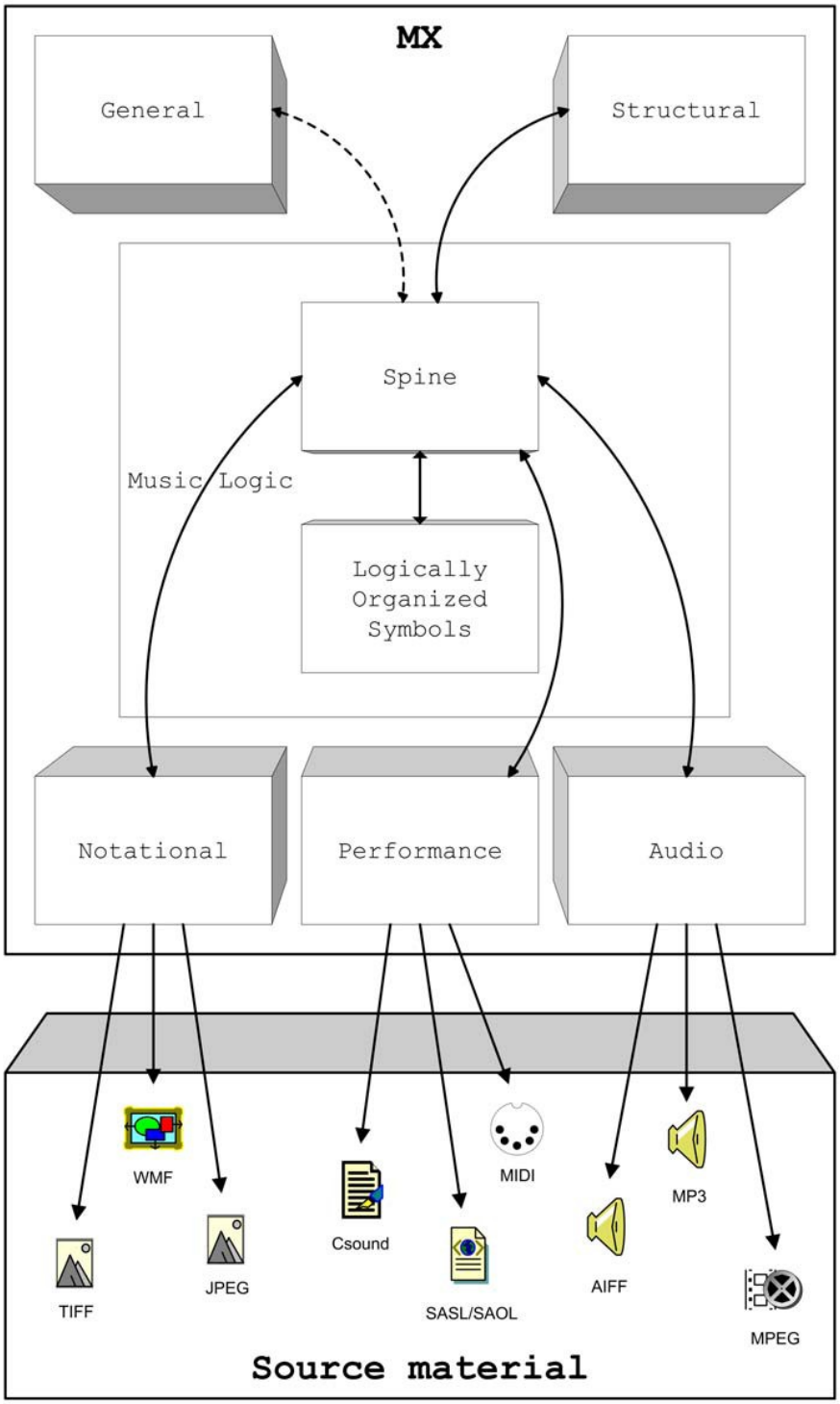


Figura 1 - Schema di interazione tra i layer IEEE 1599

1.2.1 Il layer General

Come già accennato precedentemente, l'obbiettivo principale del layer General è quello di descrivere informazioni di catalogo sul brano. Inoltre in esso possono essere descritte ulteriori informazioni, come quelle, ad esempio, sui supporti, analogici o digitali, e quelle dedicate alle annotazioni e alla gestione dei diritti (DRM). Tuttavia esso non è direttamente legato ai contenuti audio o alla partitura. Ciò è ben evidente nello schema in Figura 1, in cui il layer General è connesso agli altri blocchi da una linea tratteggiata, che rappresenta una connessione debole. Tale connessione è comunque sufficiente a garantire la temporizzazione, ove necessaria, di alcuni suoi sotto-elementi. Allo scopo della presente trattazione, saranno di seguito analizzati in maniera sistemica i sotto-elementi del layer General con maggior interesse invece verso un sotto-elemento specifico: il campo *related files*.

- Description: è il più importante sotto-elemento, fornisce informazioni sul brano, il suo autore e il genere. In particolare l'elemento Description contiene a sua volta 6 sotto-elementi, ovvero *Work title* (titolo del'opera complessiva, ad esempio il nome della sinfonia di cui il file rappresenta un solo movimento, o il nome dell'album da cui è tratta la canzone considerata) , *Work number* (permette l'identificazione di un numero di catalogo convenzionale all'interno dell'intera produzione musicale di un autore, più spesso utilizzato nella catalogazione della musica classica), *Movement title* (elemento in cui è descritto il titolo del brano in questione), *Movement number* (la posizione del brano all'interno di una sequenza ordinata di brani che costituisce l'intera composizione, precedentemente definita all'interno dell'elemento *Work title*), *Genre* (dedicato alla descrizione del genere musicale, secondo una tassonomia gerarchica attualmente in sviluppo presso il LIM) e *Author* (contiene il nome il nome o i nomi degli autori, l'attributo type permette di specificare il tipo di autore tra compositore, revisore, librettista, ecc.)
- Casting: elemento dedicato alle informazioni sul cast e sugli esecutori. Può essere presente o meno.
- Analog media: comprende informazioni relative ai supporti analogici, come importazione/esportazione, restauro, catalogazione, ecc. Attualmente non implementato.
- Notes: elemento riservato alle annotazioni testuali generice sul brano in questione.
- Rights: specifica i diritti di proprietà intellettuale e le informazioni sulla sicurezza dell'intero pezzo, attualmente in fase di sviluppo.
- Related files: è un elemento riservato alla descrizione dei file dati collegati alla musica, in riferimento a tutti i layer. Rappresenta l'elenco dei file multimediali coinvolti a corredo

dell'informazione musicale, e permette quindi di collegare degli oggetti esterni che non sono direttamente correlati al brano (ad esempio foto dell'autore, copertine di CD, suonerie per cellulari create a partire dal brano originale ecc.). I file possono essere linkati con il loro percorso relativo o con il loro percorso assoluto, permettendo di linkare sia file locali, sia file remoti che si trovano sulla rete. Virtualmente è supportato qualunque tipo di formato di encoding: dato che esistono solo connessioni deboli tra i related files e il brano in questione, non è richiesta alcuna interazione con il contenuto musicale, ed ogni tipo di formato di file può essere utile allo scopo di aggiungere informazioni, purchè si disponga di un'applicazione in grado di gestire quel tipo di formato. Possono essere impiegati due attributi in particolare, chiamati start_event_ref e end_event_ref, entrambi riferiti ad eventi presenti all'interno dello Spine, per indicare quando i related files dovrebbero essere aperti e chiusi dall'applicazione in merito.

Vediamo di seguito un esempio di Layer General di un file IEEE 1599 (si noti la presenta degli attributi start_event_ref ed end_event_ref all'interno del primo related_file).

```
<general>
    <description>
      <main_title>E lucevan le stelle</main_title>
      <author type="composer">Giacomo Puccini</author>
      <author type="librettist">Giuseppe Giacosa</author>
      <author type="librettist">Luigi Illica</author>
      <work_title>Tosca - Atto III</work_title>
    </description>
    <related_files>
      <related_file file_name="tosca/iconografico/manifesto_1.jpg"
      file_format="image_jpeg" encoding_format="image_jpeg" description="Manifesto"
      start_event_ref="intro_1" end_event_ref="p18v1_52"/>
      <related_file file_name="tosca/iconografico/manifesto_2.jpg"
      file_format="image_jpeg" encoding_format="image_jpeg" description="Manifesto"/>
      <related_file file_name="tosca/iconografico/bozzetto_lettera.jpg"
      file_format="image_jpeg" encoding_format="image_jpeg" description="La scena della
      lettera (bozzetto)"/>
    </related_files>
</general>
```

1.2.2 Il layer Audio

Il layer Audio descrive le proprietà del materiale musicale audio. I formati per la rappresentazione dell'informazione audio si possono suddividere in 2 categorie: compressi e non compressi. Tra questi ultimi troviamo PCM/WAV, AIFF e MuLaw. A loro volta, i formati compressi possono presentare perdita di informazione (formati lossy) o meno (formati lossless). Tra i formati compressi con perdita ricordiamo MPEG e DOLBY, tra quelli senza perdita SHN e ADPCM.
Per poter collegare un generico file audio allo spine è necessario estrarre le caratteristiche degli eventi musicali relative alla loro localizzazione temporale. Si tratta di informazioni che non dipendono dal particolare formato di codifica nè dall'eventuale compressione. La localizzazione degli eventi musicali all'interno di clip (audio o video, in quest'ultimo caso comunque si fa riferimento alla traccia audio presente nel video) è ottenuta mediante una struttura di indexing basata su temporizzazione assoluta. Ogni clip è composta da 3 parti principali:

1) Clip general: elemento che descrive i metadati riguardanti la performance musicale.
2) Clip format: elemento che memorizza dati tecnici sul tipo di formato audio/video
3) Clip indexing: elemento che contiene una lista di index, basati su temporizzazione assoluta, rappresentanti l'esatta di eventi musicali in un preciso istante della traccia audio.

L'elemento clip indexing rappresenta la parte più importante della descrizione dell'audio in MX.
In particolare, ogni suo singolo sotto-elemento index contiene la temporizzazione assoluta, in secondi, di quel preciso index (attributo start_time) assieme al riferimento all'evento musicale presente nello spine (attributo event_ref).
Vediamo ora un esempio di Layer Audio di un file IEEE 1599:

```
<audio>
    <track file_name="audio\por_una_cabeza_fisa.mp3" file_format="audio_mp3"
    encoding_format="audio_mp3">
      <track_indexing timing_type="seconds">
        <track_event event_ref="Violino_1_1_voice0_measure1_ev0" start_time="0" />
        <track_event event_ref="Violino_2_2_voice0_measure1_ev0" start_time="0" />
        <track_event event_ref="Violino_3_3_voice0_measure1_ev0" start_time="0" />
        <track_event event_ref="Violino_4_4_voice0_measure1_ev0" start_time="0" />
        <track_event event_ref="Cello_5_voice0_measure1_ev0" start_time="0" />
                                    ...
      </track_indexing>
    </track>
</audio>
```

2. Tecnologie utilizzate: Mono, Gtk#, SDL.NET

A seguito di un'attenta valutazione, visti i requisiti di compatibilità multipiattaforma del software, si è scelto come ambiente di sviluppo il framework Mono, porting open-source del .NET framework di Microsoft. L'intero progetto è stato realizzato sul sistema operativo Ubuntu, utilizzando come IDE MonoDevelop. Per la gestione della parte audio del software è stato invece utilizzato SDL.NET, un insieme di binding .NET per la celebre libreria di sviluppo di videogames SDL. Nei seguenti paragrafi sarà spiegato nel dettaglio l'insieme delle componenti utilizzate nel corso della realizzazione del software.

2.1 Origini di Mono, derivazione da .NET

Mono è un progetto open source coordinato da Xamarin (precedentemente da Novell, e originariamente da Ximian) per creare un insieme di strumenti compatibili con il framework .NET, che comprende un compilatore C# e un Common Language Runtime. Lo scopo di tale progetto non è solo di essere in grado di far girare applicazioni .NET su più piattaforme, ma anche di portare strumenti di sviluppo software migliori verso la piattaforma Linux. Mono, nella sua versione corrente, può essere installato ed eseguito su Android, BSD, iOS, Linux, Mac OS X, Windows, Solaris, e altri sistemi Unix, così come su alcune console come Playstation 3, Wii e Xbox 360. La storia di Mono comincia nel Giugno del 2000, quando Microsoft annunciò per la prima volta il suo .NET framework descrivendolo come "una nuova piattaforma basata sugli standard Internet".

Nel Dicembre dello stesso anno fu pubblicata come standard aperto la sottostante Common Language Interface, ponendo così le basi per implementazioni indipendenti dello standard. La società Ximian si convinse che .NET avrebbe avuto le potenzialità per aumentare la produttività dei programmatori e cominciò a lavorare su un porting per Linux. Rendendosi però conto che un team così esiguo di persone non sarebbe riuscito a sviluppare un prodotto completo, pubblicarono il progetto come open source, sotto il nome di Mono, il 19 Luglio 2001. Dopo 3 anni di sviluppo, Mono 1.0 fu rilasciato il 30 Giugno 2004. Il progetto si è poi evoluto da semplice strumento di sviluppo di applicazioni desktop Linux fino a supportare una moltitudine di architetture e sistemi operativi diversi.

Attualmente il progetto è mantenuto e supportato da Xamarin, e la sua ultima versione (2.10.8, rilasciata nel Marzo 2012) fornisce agli sviluppatori le core API del .NET framework così come il supporto a Visual Basic.NET e C# (nelle versioni 2.0, 3.0, 4.0), LINQ, XML e SQL.

2.1.1 Differenze tra Mono e .NET

L'aspetto più controverso dell'esistenza del progetto Mono è legato all'implementatazione di alcune parti del .NET framework di Microsoft che risultano coperte da brevetti. Sul sito ufficiale del progetto si legge quanto segue:

"*Il Framework .NET si divide in due parti: le tecnologie coperte dall'ECMA/ISO e le altre tecnologie sviluppate sopra di esse come ADO.NET, ASP.NET e Windows.Forms. Mono implementa le parti ECMA/ISO, essendo questo un progetto che permette l'implementazione di blocchi a livelli più alti come ASP.NET, ADO.NET e Windows.Forms. Gli elementi controversi sono i sottoinsiemi ADO.NET, ASP.NET e Windows.Forms. Questi sono convenienti per quanti necessitano piena compatibilità con la piattaforma Windows, ma non sono richiesti per la piattaforma open source Mono, così come l'integrazione con l'attuale ottimo supporto per Linux. La strategia di Mono riguardo queste tecnologie è la seguente: (1) aggirare i brevetti usando tecniche di implementazione differente che mantengono la API, ma cambiano i meccanismi. Se ciò non fosse possibile, vorremmo (2) rimuovere le parti di codice coperte da brevetti, oltre che (3) trovare dei precedenti che possano rendere inutilizzabili i brevetti stessi.*"

2.1.2 Monodevelop

MonoDevelop è un IDE (Integrated Development Environment) open source progettato per essere utilizzato con C# e altri linguaggi del framework .NET, che punta ad essere una piattaforma estensibile su cui costruire qualunque tipo di strumento di sviluppo software. Il progetto mira inoltre a diventare l'IDE di riferimento per lo sviluppo di applicazioni Mono/.NET su piattaforma Unix, così come Visual Studio di Microsoft lo è in ambiente Windows. Presenta caratteristiche atte a velocizzare il processo di programmazione, come l'autocompletamento del codice, un browser per le classi, un designer di interfacce grafiche, ecc. Questo IDE nasce per essere utilizzato primariamente per lo sviluppo del progetto GNOME, uno degli ambienti desktop più utilizzati in campo Unix, ed è scritto facendo utilizzo di Mono e delle librerie grafiche Gtk (binding di Gtk+ verso il linguaggio C#).

2.2 Origini di Gtk+ e Gtk#

Gtk+ è un toolkit multipiattaforma per creare interfacce grafiche, sviluppato a partire dal 1998 come un set di widget per il programma GIMP (GNU Image Manipulation Program), ma accresciutosi fino a diventare uno dei più utilizzati toolkit grafici nell'ambiente open source. Sviluppato in C, supporta nativamente l'ambiente grafico X Window System e Microsoft Windows. Numerose

applicazioni, in ambito commerciale e non, fanno uso di questo toolkit, ad esempio il sopracitato progetto GNOME, che è interamente basato su Gtk+. Questo toolkit è stato progettato fin dall'inizio per supportare diversi binding verso altri linguaggi, oltre al C e al C++ (tra cui Perl, Python, C#, ecc). Gtk# nasce appunto per permettere di sviluppare facilmente applicazioni Mono con interfaccia grafica pensate per il desktop GNOME, ma la sua natura multipiattaforma permette di sviluppare per una serie eterogenea di sistemi e architetture.

2.3 SDL

Simple DirectMedia Layer (SDL) è un'API gratuita multi-piattaforma (Windows, Linux, Unix compreso Mac OS X) per lo sviluppo multimediale. E'una libreria progettata per fornire accesso a basso livello a audio, tastiera, mouse, joystick, hardware 3d via OpenGL, framebuffer video 2D. Scritta in C, supporta nativamente anche il C++, e ha binding verso numerosi altri linguaggi, tra cui Ada, C#, Erlang, Euphoria, Go, Guile, Haskell, Objective C, Pascal, Perl, PHP, Pike, Python, solo per citarne alcuni. SDL è pensato per lo sviluppo di videogiochi, pertanto espone allo sviluppatore delle funzioni per il controllo del video, per il controllo degli eventi da mouse e tastiera, per il multithreading, e per l'audio. In particolare, la libreria permette, per quanto riguarda l'audio, un playback a 8 o 16 bit, mono o stereo, con conversione ottimale laddove il formato non è supportato dall'hardware. L'audio inoltre gira indipendentemente in un thread separato, controllato per mezzo di un meccanismo di callback controllato dall'utente. L'esistenza di binding verso C# (detti SDL.NET dall'unione dei nomi delle due tecnologie) e la natura multipiattaforma di questa libreria ci ha fatto optare per la sua scelta nella parte di gestione audio del nostro software.

2.3.1 SDL.NET

SDL.NET è un insieme di binding .NET per la libreria SDL che permettono l'accesso a molte delle funzioni di tale libreria utilizzando i linguaggi .NET, e in particolare C#.
In realtà consiste di 2 librerie distinte, ovvero la libreria Tao.sdl, una collezione di binding e librerie pensate per facilitare lo sviluppo di videogiochi multipiattaforma utilizzando Mono e le piattaforme .NET, e la libreria SDL.NET vera e propria. Tao.sdl contiene i binding a basso livello verso la libreria SDL, scritta in C, e tutte le librerie correlate, mentre SDL.NET non fa altro che costruire un'interfaccia orientata agli oggetti attorno alla libreria Tao. Per poter utilizzare SDL.NET è necessario comunque avere installato sul proprio sistema operativo la libreria SDL, così come una versione del .NET framework uguale o superiore alla 2.0, oppure Mono.

Il progetto è articolato e suddiviso in diversi pacchetti (che corrispondono ai namespaces da richiamare, tramite la direttiva "using", all'interno dell'IDE di riferimento), ognuno dedicato a un particolare aspetto della programmazione di videogiochi (la parte grafica, la parte audio, ecc).
Il software oggetto dell'elaborato utilizza il pacchetto SdlDotNet.Audio per la riproduzione di file multimediali. Saranno perciò ora analizzate nel dettaglio alcune caratteristiche peculiari della parte dedicata all'audio di questa libreria.

2.3.2 - SdlDotNet.Audio

SdlDotNet.Audio presenta numerose classi atte ad avvantaggiare lo sviluppatore nella creazione del comparto audio di un videogioco, tuttavia è possibile, senza limitazioni, utilizzare le stesse classi nello sviluppo di applicazioni multimediali.
L'aspetto sicuramente più interessante è la possibilità di interagire con molteplici formati audio, sia compressi (.mp3, .ogg Vorbis, ecc.) che non (wav, PCM). Vediamo ora alcune delle classi principali, e tra queste evidenzieremo quelle utilizzate nello sviluppo dell'applicativo:

- Classe Music: rappresenta un frammento musicale. Può ospitare formati compressi e non. Il metodo costruttore Music(string fileName) permette di inizializzare un oggetto Music caricandolo da un file. Altri metodi degni di nota sono il metodo Play, che presenta 3 override diversi, ovvero Play(), senza parametri, che riproduce il file caricato, Play (bool continuous) che permette una riproduzione infinita del frammento musicale, e Play(int numberOfTimes) che permette una ripetizione programmata, secondo un numero di volte stabilito. Viene utilizzata all'interno del progetto (viene in realtà utilizzata la classe equivalente MusicPlayer) per caricare i file audio da riprodurre.
  Un semplice esempio di utilizzo della classe potrebbe essere:

```
Music techno = new Music("techno.mp3");
Music jazz = new Music("jazz.mid");
techno.Play();
```

- Classe Sound: è interessante come SdlDotNet faccia distinzione tra questa classe e quella precedentemente descritta: un suono non è una musica, e appartengono a due classi distinte. Questo perchè generalmente in ambito videoludico per "musica" si intende una musica di sottofondo, mentre per "suono" si intendono tutti quegli elementi sonori legati all'ambiente, al personaggio giocante, spesso hanno il ruolo di essere un feedback verso l'utente (si pensi al suono di SuperMario che spacca un mattone) più che un elemento indipendente dal gioco

in sè. La classe presenta comunque metodi simili alla precedente, con il costruttore Sound (string file) che inizializza un oggetto Sound da un file, e diversi override del metodo Play che permettono la ripetizione della riproduzione.

- Classe MusicCollection: rappresenta una collezione di frammenti musicali memorizzati all'interno di un dizionario come una coppia chiave-valore, come nell'esempio:

```
MusicCollection tunes = new MusicCollection();
tunes.Add("techno", "techno.mid");
tunes.Add("jazz.mid");
tunes["techo"].Play();
tunes["jazz.mid"].Play();
```

Tale dizionario può essere scorso attraverso i metodi classici utilizzati per le collezioni. Quest'ultima classe, sebbene non ancora implementata, potrebbe essere utilizzata in futuro in ulteriori sviluppi dell'applicazione oggetto del presente trattato, permettendo ad esempio di memorizare il file MIDI di un brano (sebbene appartenga al layer Performance, non oggetto di questo elaborato) in una coppia di valori assieme ad una delle tracce audio presenti, per lasciare all'utente la possibilità di far suonare il file MIDI piuttosto che la traccia audio.

3. Plugin multipiattaforma per slideshow a partire da documenti IEEE 1599

Il software oggetto dell'elaborato ha come obbiettivo la creazione di slideshow temporizzati a partire dai related files di tipo immagine contenuti nel layer General di un documento IEEE1599.
Esso tuttavia è stato sviluppato con l'intento di essere un'estensione di un progetto ben più ampio, ossia un visualizzatore completo, multipiattaforma, di documenti in formato IEEE 1599, correntemente in fase di sviluppo presso il LIM con il nome in codice "TripleEye".
Pertanto il lavoro è cominciato direttamente sul codice del progetto "TripleEye", di cui sarà data una panoramica nel paragrafo seguente.

3.1 TripleEye e TripleEyeEngine

TripleEye si propone di essere un visualizzatore completo di documenti IEEE 1599 tramite cui poter accedere ed interagire con tutti i layer, lasciando all'utente la possibilità di spostarsi all'interno del brano da un particolare momento ad un altro e tra un layer e l'altro. Allo stato attuale si trova ancora in una fase di sviluppo embrionale, e si presenta come una serie di finestre separate, visualizzate a seconda dei layer presenti nel file IEEE1599 aperto dalla finestra principale, ognuna dedicata ad un differente layer. Il tutto è gestito da un motore che permette l'estrapolazione delle informazioni dai documenti IEEE1599 e la loro sincronizzazione, chiamato TripleEyeEngine. Attualmente il visualizzatore delle score (che mostra i contenuti del layer Notational) è l'unico implementato e correttamente funzionante, e permette lo spostamento tra i vari eventi musicali all'interno della partitura, e di cambiare partitura laddove ne fosse presente più d'una. Di seguito uno screenshot del software eseguito su sistema operativo Ubuntu.

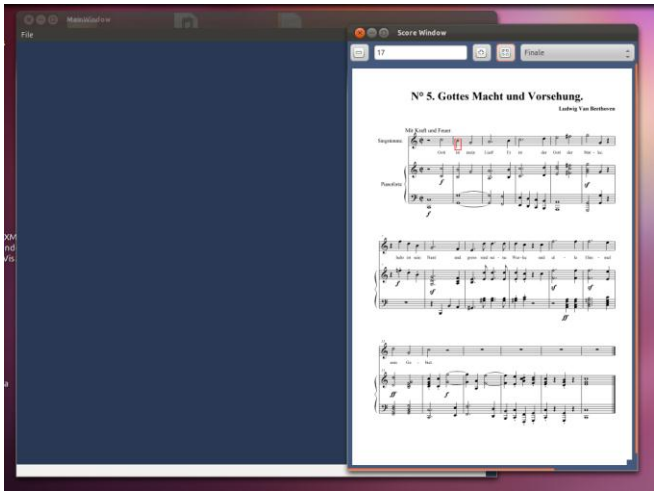


Figura 3.1 - TripleEye in esecuzione su una macchina Ubuntu

Il TripleEyeEngine rappresenta il vero e proprio cuore dell'applicazione, in quanto estrapola ed organizza i contenuti del file IEEE 1599 mediante la tecnologia LINQ (Language Integrated Query, un componente del .NET Framework Microsoft, integrato in Mono, che aggiunge ai linguaggi .NET la possibilità di effettuare interrogazioni su oggetti utilizzando una sintassi simile a SQL).
Il file TripleEyeEngine.cs ospita al suo interno la dichiarazione della classe TripleEyeEngine, oltre ad una serie di strutture atte a memorizzare i dati estrapolati dall'XML.
La classe TripleEyeEngine presenta un metodo costruttore che si occupa di caricare come oggetto Xdocument il file IEEE1599 aperto dall'applicazione, e di effettuare un parsing dell'XML mediante delle query LINQ specifiche, atte ad estrapolare le informazioni dai vari layer. Qualunque applicazione utilizzi il TripleEyeEngine avrà un metodo che si occupa di creare un oggetto engine, istanza della classe, che carica al suo interno il documento IEEE 1599 selezionato dall'utente. Vedremo nel paragrafo 3.2 come questo accada all'interno del nostro software.
Le varie query sono effettuate singolarmente per ogni layer e per ogni tipo di informazione contenuta nel documento (Voices, Spine, Related Files, Lyrics, Scores, Audio/Video Clips) e vanno ad inserire i dati estrapolati all'interno di dizionari contenenti le strutture precedentemente descritte.
Di seguito saranno analizzate le strutture e le corrispettive query LINQ relative ai Related Files e ai clip Audio/Video, in quanto il visualizzatore reimplementa queste sezioni specifiche del TripleEyeEngine per ottenere i dati necessari all'esecuzione dello slideshow.

3.1.1 Related Files

```
public struct relatedFilesStruct
{
       public string Description;
       public string StartEventRef;
       public string EndEventRef;

       public relatedFilesStruct(string description, string startEventRef,
                                                          string endEventRef)
       {
              Description = description;
              StartEventRef = startEventRef;
              EndEventRef = endEventRef;
       }
}
```

La struttura relatedFilesStruct è atta ad ospitare nei suoi 3 campi i valori Description, StartEventRef ed EndEventRef di ogni related file contenuto nel documento. Si noti che i campi StartEventRef ed EndEventRef sono di tipo string, questo perchè fanno riferimento all'attributo event_ref presente nei singoli eventi musicali, piuttosto che alla temporizzazione assoluta.
Il costruttore prende in ingresso i 3 parametri direttamente dalla query LINQ e li assegna ai 3 campi, come si vede nella porzione di codice seguente.

```
// --- Related Files ---
RelatedFilesInstances = new Dictionary<string, relatedFilesStruct>();
var relatedFiles = from relatedFile in ieee1599Doc.Descendants("related_file")
                   where (relatedFile.Attribute("encoding_format") != null &&
                   relatedFile.Attribute("encoding_format").Value.StartsWith("image"))
                   select new
                   {
                     FileName =   relatedFile.Attribute("file_name").Value,
                     Description = (relatedFile.Attribute("description") != null) ?
                     relatedFile.Attribute("description").Value : "",
                     StartEventRef = (relatedFile.Attribute("start_event_ref") != null)?
                     relatedFile.Attribute("start_event_ref").Value : null,
                     EndEventRef = (relatedFile.Attribute("end_event_ref") != null) ?
                     relatedFile.Attribute("end_event_ref").Value : null
                   };

if (relatedFiles.Count() > 0)
{
  foreach (var relatedFile in relatedFiles)
    if (RelatedFilesInstances.ContainsKey(relatedFile.FileName))
      errorsLogInsertWarning("Repeated related_file filename: " + relatedFile.FileName);
    else
      RelatedFilesInstances.Add(relatedFile.FileName, new
      relatedFilesStruct(relatedFile.Description, relatedFile.StartEventRef,
                                                  relatedFile.EndEventRef));
}
```

Viene creato un oggetto Dizionario, chiamato RelatedFilesIstances, le cui chiavi sono di tipo string e memorizzano il percorso dei file, mentre il corrispettivo valore di ogni chiave è una relatedFilesStruct. Viene quindi effettuata la query all'interno del documento IEEE1599 tra gli elementi related_file: si noti come il risultato della query sia filtrato tramite una clausola where in cui si richiede esplicitamente che l'attributo encoding_format non sia nullo e sia di tipo immagine. Vengono così ottenuti i valori degli attributi file_name, description, start_event_ref ed end_event_ref di ogni related file presente nel documento IEEE1599. Infine, tramite un ciclo foreach, viene aggiunta al Dizionario una voce per ogni corrispondente related file, inserendo come chiave il contenuto dell'attributo file_name, e come valore una relatedFilesStruct i nei cui campi Description, StartEventRef ed EndEventRef vengono inseriti i corrispettivi contenuti degli attributi description, start_event_ref ed end_event_ref del related file.

3.1.2 Clip Audio/Video

La parte di codice necessaria ad estrapolare i dati dal layer Audio risulta più complessa in quanto coinvolge e va ad inserire i valori ottenuti in 2 strutture differenti, una relativa all'indexing del singolo file audio, chiamata clipEventStruct, ed una atta a memorizzare le varie istanze di clip Audio/Video (nell'ottica di permettere all'utente di sceglierne una tra le tante, in caso se ne presenti più d'una), chiamata ClipStruct. Vediamo le 2 strutture nel dettaglio:

```
public struct clipEventStruct
{
    public double StartTime;
    public double EndTime;
    public clipEventStruct(double startTime, double endTime)
    {
        StartTime = startTime;
        EndTime = endTime;
    }
}
```

La struttura clipEventStruct ospita nei suoi 2 campi i valori StartTime e EndTime di ogni evento musicale contenuto all'interno dell'indexing di ogni clip presente nel layer audio. Il costruttore prende in ingresso i 2 parametri dalla query LINQ e li assegna ai 2 campi, come si vedrà in seguito.

```
public struct clipStruct
{
    private bool isOnlyAudio;
    public bool IsOnlyAudio
    {
        Get
        {
            return isOnlyAudio;
        }
    }
    public bool IsVideo
    {
        get
        {
            return !isOnlyAudio;
        }
    }
    public Dictionary<string, Dictionary<string, clipEventStruct>> DictionaryVoices;
    public MediaElement MediaElement;
    public clipStruct(bool _isAudio, Dictionary<string, Dictionary<string,
             clipEventStruct>> dictionaryVoices, MediaElement mediaElement)
    {
        isOnlyAudio = _isAudio;
        DictionaryVoices = dictionaryVoices;
        MediaElement = mediaElement;
    }
}
```

La struttura clipStruct ha 2 campi boolean read-only, IsOnlyAudio e IsVideo, che permettono di categorizzare la singola clip come solo audio o audio + video, e un campo Dizionario, chiamato DictionaryVoices, le cui chiavi sono di tipo string e i cui valori costituiscono un ulteriore dizionario annidato con chiave di tipo string e valore di tipo clipEventStruct. Presenta inoltre un campo chiamato MediaElement, destinato a contenere oggetti audio appartenenti alla classe MediaElement, che sarà introdotta e spiegata nel paragrafo successivo. Il costruttore clipStruct prende in ingresso i parametri direttamente dalla query LINQ e li assegna ai 3 campi isOnlyAudio, DictionaryVoices e MediaElement.

Sarà ora spiegato nelle 2 porzioni di codice seguenti (di cui una mostra la query vera e propria, e l'altra l'inserimento dei valori all'interno dei dizionari) come le informazioni vengano ottenute.

```
MediaInstances = new Dictionary<string, clipStruct>();
var audioEv = from audioEvent in ieee1599Doc.Descendants("track_event")
              where spine["_all"].ContainsKey(audioEvent.Attribute("event_ref").Value)
              select new
              {
                EventRef = audioEvent.Attribute("event_ref").Value,
                StartTime = (audioEvent.Attribute("start_time") != null) ?
                Convert.ToDouble (audioEvent.Attribute("start_time").Value.Replace('.',',')) :
                -1,
                EndTime = (audioEvent.Attribute("end_time") != null) ?
                Convert.ToDouble(audioEvent.Attribute("end_time").Value.Replace('.', ',')) :
                (audioEvent.Attribute("start_time") != null) ?
                Convert.ToDouble(audioEvent.Attribute("start_time").Value) :
                -1,
                Filename = (audioEvent.Ancestors("track").First().Attribute("file_name")!= null) ?
                audioEvent.Ancestors("track").First().Attribute("file_name").Value : ""
              };
```

Viene creato un oggetto Dizionario, chiamato MediaIstances, le cui chiavi sono di tipo string e memorizzano il percorso dei clip audio, e il corrispettivo valore di ogni chiave è una clipStruct. Viene quindi effettuata la query all'interno del documento IEEE 1599 tra gli elementi track_event: in questo caso la condizione where viene utilizzata unicamente per filtrare i risultati ottenuti mantenendo solo quelli che sono presenti nello spine. Il risultato della query sono i valori Event_ref, StartTime, EndTime e Filename. La parte di codice che segue è piuttosto elaborata, e sarà suffragata, assieme alla spiegazione, da uno schema esemplificativo che mostra il risultato dell'operazione di estrazione dei dati necessari.

```
var groupedByFileName = from audioEvent in audioEvents group audioEvent by audioEvent.Filename;
foreach (var filename in groupedByFileName)
{
   if (MediaInstances.ContainsKey(filename.Key))
   {
       errorsLogInsertWarning("Repeated media filename: " + filename.Key);
       continue;
   }
      Dictionary<string, Dictionary<string, clipEventStruct >> evDictionary = new Dictionary<string,
                                                                  Dictionary<string, clipEventStruct>>();

     evDictionary["_all"] = new Dictionary<string, clipEventStruct>();
     foreach (string voice in voices)
       evDictionary[voice] = new Dictionary<string, clipEventStruct>();
     foreach (var audioEvent in filename)
     {
       if (evDictionary["_all"].ContainsKey(audioEvent.EventRef))
          errorsLogInsertWarning("Repeated track_event: " + audioEvent.EventRef);
       else
       {
           evDictionary["_all"].Add(audioEvent.EventRef, new clipEventStruct(audioEvent.StartTime *
                                                                  1000, audioEvent.EndTime * 1000));
              if (spine["_all"][audioEvent.EventRef].Voice != null)
                evDictionary[spine["_all"][audioEvent.EventRef].Voice].Add(audioEvent.EventRef, new
                          clipEventStruct(audioEvent.StartTime * 1000, audioEvent.EndTime * 1000));

       }

     }
   bool isOnlyAudio;
   var mediaType = from media in ieee1599Doc.Descendants("track")
                   where media.Attribute("file_name").Value == filename.Key
                   select media.Attribute("encoding_format").Value;
   if (mediaType.First().StartsWith("audio"))
       isOnlyAudio = true;
   else
       isOnlyAudio = false;
   string relFilePath=filename.Key.Replace("\\",System.IO.Path.DirectorySeparatorChar.ToString());
                                                  //rimpiazza i backslash nel filename a seconda dell'OS
   string filepath = Path.Combine (path, UnixRelFilePath);   //crea percorso assoluto dei file_name
   MediaElement me = new MediaElement();
   MediaInstances.Add(filepath, new clipStruct(isOnlyAudio, evDictionary, me));
}
```

Tutti gli eventi audio vengono raggruppati in base al filename, e viene effettuato un ciclo foreach che scorre i gruppi e per ognuno di essi crea un Dizionario chiamato evDictionary (dizionario degli eventi). Le chiavi di quest'ultimo sono di tipo string e rappresentano le varie voci all'interno dello spine, mentre i valori sono a loro volta dizionari con chiavi string e valori clipEventStruct. Viene poi inserito un particolare elemento in evDictionary, la cui chiave è "_all" (ossia contiene eventi relativi a tutte le voci all'interno dello Spine). Mediante un foreach annidato al primo che scorre i vari eventi audio all'interno di ogni gruppo, viene inserita in evDictionary["_all"] una coppia di valori costituita dall'EventRef restituito dalla query come chiave, e da una clipEventStruct, i cui campi StartTime ed EndTime sono i valori numerici StartTime ed EndTime restituiti dalla query. (Si ricordi che ogni elemento all'interno di evDictionary è a sua volta un dizionario). A questo punto, sempre all'interno del primo foreach, una volta creata la variabile boolean isOnlyAudio viene effettuata un'ultima query tra i vari elementi "track" all'interno del layer audio per controllare se siano solo audio o audio + video (attributo encoding_format). La variabile isOnlyAudio assume perciò il valore true se la track corrente è un file audio o false in caso contrario.

Infine viene aggiunta al dizionario MediaInstances la coppia di valori costituita da percorso del file audio e dalla clipStruct i cui campi IsOnlyAudio, DictionaryVoices e MediaElement assumono rispettivamente i valori delle variabili isOnlyAudio, evDictionary, me.

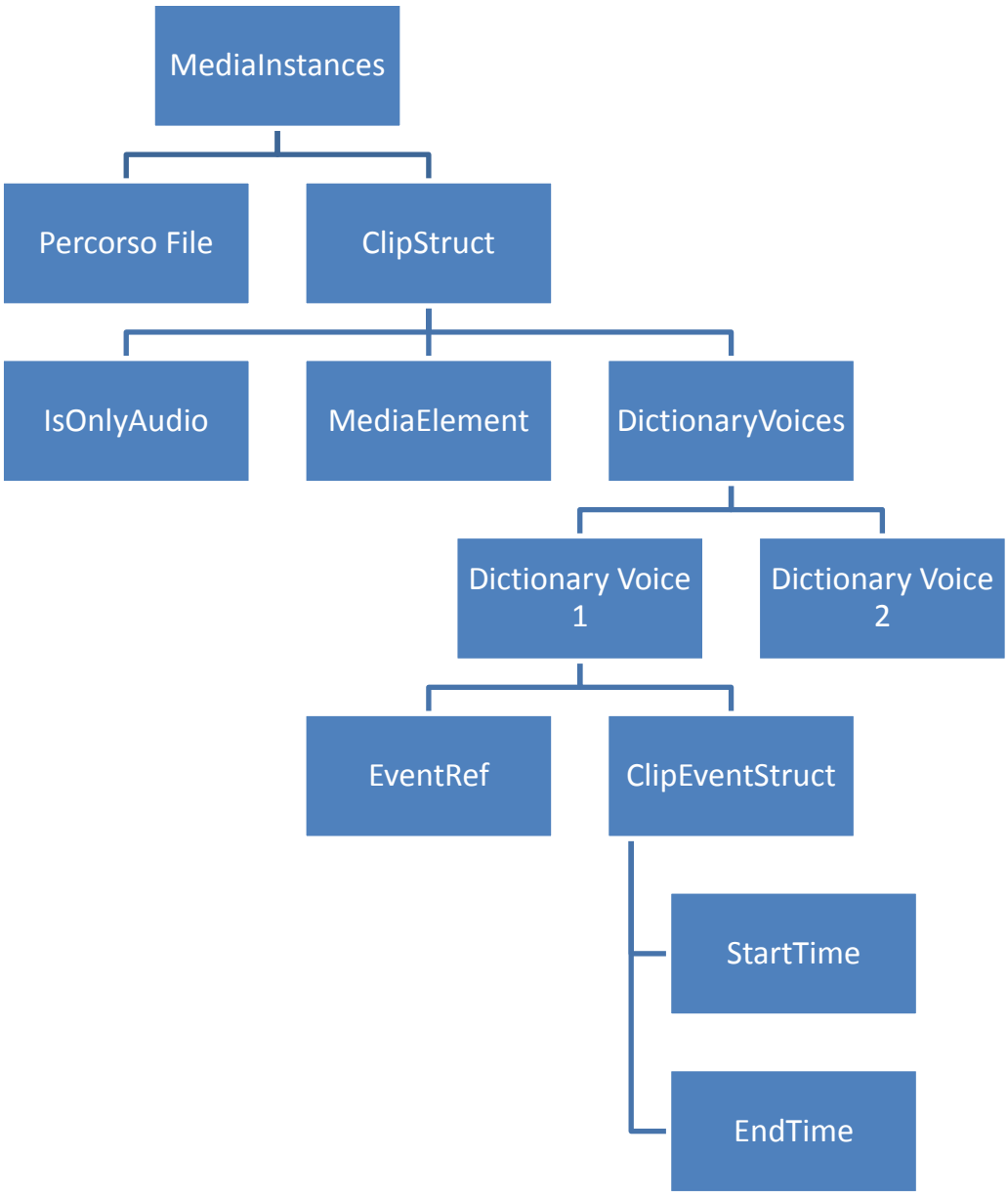


Figura 3. 1 - Schema di immagazinamento dei dati Audio nel TripleEyeEngine

3.2 Individuazione delle classi e degli algoritmi necessari

Il software realizzato permette, allo stato attuale, di selezionare una tra le molteplici clip audio presenti in un file IEEE1 599, e tramite il tasto Play, di visualizzare uno slideshow che mostra i i vari related files di tipo immagine, contenuti all'interno del layer General, tramite una serie di finestre indipendenti temporizzate. Alla fine del brano, o alla pressione del tasto Stop, lo slideshow si ferma.
Si rimanda al paragrafo 3.3 per una descrizione dell'interfaccia grafica del software, e al capitolo 4 per un'analisi dettagliata del suo comportamento.
Il presente paragrafo è invece dedicato alla descrizione dello sviluppo vero e proprio dell'applicazione, e all'implementazione delle caratteristiche fondamentali.
Il programma utilizza la parte di codice presente nel TripleEyeEngine finora descritta, unita ad una serie di soluzioni ad hoc sviluppate per permettere la temporizzazione corretta delle immagini.
A seguito di un'analisi delle problematiche da affrontare nella realizzazione del progetto sono emersi più quesiti, le cui risposte costituiscono i punti fondamentali attraverso cui si è svolto lo sviluppo del software:

- Come recuperare ed utilizzare i dati relativi alla temporizzazione dei related_files
- Come permettere la riproduzione di un file audio a scelta tra i file audio disponibili
- In che modo visualizzare le immagini contenute nei related files
- Sviluppare un algoritmo capace di sincronizzare lo slideshow con la riproduzione dell'audio
- Sviluppare un algoritmo in grado di fermare l'esecuzione dello slideshow secondo la volontà dell'utente

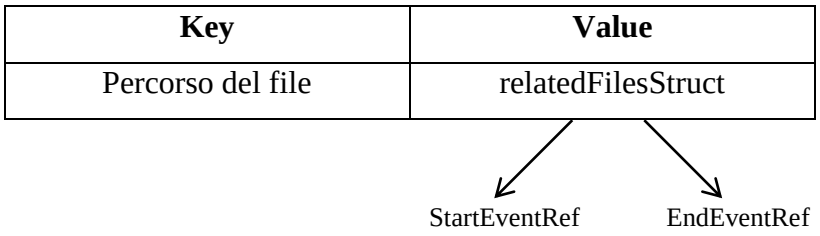
I sottoparagrafi seguenti sono dedicati alla descrizione delle soluzioni a ciascuna delle problematiche elencate.

3.2.1 Fetching delle informazioni relative ai related files

Il visualizzatore di slideshow utilizza il TripleEyeEngine, creando all'apertura del documento IEEE 1599 un'istanza della classe TripleEyeEngine, chiamata engine.

```
TripleEyeEngineLite engine;
engine = new TripleEyeEngineLite(fcd.Filename);
```

Il paragrafo 3.1 ha mostrato come di fatto sia possibile, tramite il TripleEyeEngine, ottenere le infomazioni relative ai related_files e immagazzinarle all'interno del dizionario RelatedFilesIstances. Analizzando una coppia di valori all'interno del dizionario, ci si trova di fronte alla seguente situazione:



Di fatto si è in grado di determinare il percorso del file, e gli eventi musicali in cui il file deve rispettivamente apparire e scomparire dallo slideshow. Tuttavia manca completamente il concetto di temporizzazione assoluta, in quanto non ci si sta riferendo a dei particolari istanti di tempo, ma a degli eventi logici contenuti nell'indexing della clip audio, che a loro volta sono riferimenti ad eventi contenuti nello Spine. Questo, come si vedrà, comporta il vantaggio di poter temporizzare diversamente uno stesso evento a seconda della traccia audio correntemente in esecuzione.

Per poter temporizzare un related file in maniera assoluta, ottenendo dei valori in secondi da poter passare ad un algoritmo che generi lo slideshow, dobbiamo necessariamente fare riferimento al layer Audio.

Ogni singolo sotto-elemento index dell'elemento track_indexing presenta infatti, come già analizzato nel capitolo 1, sia il corrispettivo riferimento all'evento musicale presente nello Spine (attributo event_ref) sia la temporizzazione assoluta di quell'evento rispetto alla traccia audio che stiamo prendendo in considerazione (attributo timing).

Tali attributi sono memorizzati dal TripleEyeEngine nel dizionario MediaInstances, come visto nel paragrafo precedente, ed è possibile utilizzarli per "tradurre" i valori StartEventRef ed EndEventRef nei corrispettivi valori numerici di tempo.

L'algoritmo messo a punto per eseguire tale operazione utilizza una serie di ArrayList in cui vengono trasferiti, mediante dei foreach, tutti i percorsi dei related files, gli StartEventRef ed EndEventRef, e tutti i dati provenienti dal dizionario MediaInstances relativi agli EventRefs e ai timing. La scelta degli ArrayList è motivata dal fatto che, a differenza dei dizionari, essi sono scorribili tramite un indice numerico.

L'ArrayList eventrefs contiene perciò tutti gli id degli eventi audio relativi alla traccia considerata, mentre l'ArrayList times contiene tutti i tempi assoluti.

```
ArrayList relfilename = new ArrayList();
ArrayList start_time= new ArrayList();
ArrayList start_event_ref = new ArrayList();
ArrayList end_time= new ArrayList();
ArrayList end_event_ref = new ArrayList();
ArrayList times= new ArrayList();
ArrayList eventrefs= new ArrayList();

foreach (var item in engine.RelatedFilesInstances)
{
    string filename= item.Key.ToString();
    string RelFilePath= filename.Replace("\\",System.IO.Path.DirectorySeparatorChar.ToString());
    string fullfilepath = System.IO.Path.Combine(engine.FullPath, RelFilePath);
    relfilename.Add(fullfilepath);
    start_event_ref.Add(item.Value.StartEventRef);
    end_event_ref.Add(item.Value.EndEventRef);
}

foreach ( var item in engine.MediaInstances[curRow].DictionaryVoices["_all"])
{
    times.Add(item.Value.StartTime);
    eventrefs.Add(item.Key);
}
```

Tramite due cicli for annidati si scorrono e si confrontano i valori dell'ArrayList eventrefs con i valori dell'ArrayList start_event_ref ed end_event_ref, e laddove essi coincidono, il valore corrispondente dell'ArrayList times viene aggiunto all'Arraylist start_time, oppure ad end_time.

```
for (int i=0;i< relfilename.Count;i++)
{
    for (int i2=0;i2< eventrefs.Count;i2++)
    {
        if ( start_event_ref[i].ToString()==eventrefs[i2].ToString())
        {
            start_time.Add(times[i2]);
        }
        else if (end_event_ref[i].ToString()==eventrefs[i2].ToString())
        {
            end_time.Add(times[i2]);
        }
    }
}
```

Si hanno così a disposizione 3 ArrayList che contengono rispettivamente tutti i percorsi dei related files, tutti gli istanti in cui essi devono essere visualizzati e tutti gli istanti in cui essi devono scomparire. Mediante un indice è possibile scorrere parallelamente i 3 Arraylist ed ottenere, per l'n-esimo file, l'istante in cui farlo apparire e scomparire.

3.2.2 La classe MediaElement

Questa classe nasce allo scopo di poter rappresentare e riprodurre i clip audio contenuti nel documenti IEEE 1599. Era presente in origine nel progetto TripleEye, ma risultava vuota e non ancora implementata. Nell'implementazione attuale utilizza alcune funzioni di SDL.NET per riprodurre il file selezionato. I metodi più importanti presenti in questa classe, oltre al costruttore, sono il metodo Play() e il metodo Stop(), che rispettivamente fanno partire e fermano la riproduzione del brano selezionato tramite l'utilizzo della classe MusicPlayer contenuta in SDL.NET. Una volta caricato il file audio da riprodurre come un'istanza della classe Music, tramite l'utilizzo della classe MusicPlayer è possibile far partire la riproduzione. Qui di seguito, a scopo esemplificativo, una dichiarazione della classe che oltre al costruttore presenta i soli metodi Play() e Stop(), e il campo boolean IsPlaying, in sola lettura, che restituisce il valore true qualora il brano sia in esecuzione.

```
public class MediaElement
{
    Music music;

    public MediaElement(string _source)
    {
       music = new Music(_source);
       Source = music.FileName;
    }


    public void Play()
    {
       MusicPlayer.Load(music);
       MusicPlayer.Play();
    }

    public void Stop()
    {
    MusicPlayer.Stop();
    }
    public bool IsPlaying
    {
        get
         {
            return MusicPlayer.IsPlaying;
         }
    }
}
```

3.2.3 La classe relatedFileWindow

Questa classe nasce come risposta alla domanda su come rappresentare le immagini contenute nei i related files. Il problema principale della rappresentazione delle immagini era costituito dal fatto che è possibile che l'end_event_ref di un dato related file n sia consecutivo allo start_event_ref del related file successivo n+1.

Pertanto la rappresentazione su finestra singola risulterebbe impossibile, dal momento che le 2 immagini si sovrapporrebbero per un tempo t= end_time[n] – start_time[n+1].
La soluzione a tale problema è stata l'implementazione di una classe di finestre Gtk# temporizzate, ciascuna delle quali ospita al suo interno una sola n-esima immagine, e viene visualizzata sullo schermo per un tempo t= end_time[n] – start_time[n].
Vediamo ora la dichiarazione della classe, seguita da una più dettagliata spiegazione:

```
public partial class relatedFileWindow : Gtk.Window
{

    Gtk.Image relfileimage;

    public relatedFileWindow (string RelFilePath, double start_time, double end_time ) :
           base(Gtk.WindowType.Toplevel)
    {

        uint window_duration = Convert.ToUInt32 (end_time - start_time);
       GLib.Timeout.Add (window_duration, new GLib.TimeoutHandler (window_close));
       relfileimage = new Gtk.Image(RelFilePath);
       this.Add(relfileimage);
       this.Build ();
    }

    bool window_close ()
    {
        this.Destroy();
        return true;
    }
}
```

Il costruttore della finestra ha in ingresso 3 parametri, che sono rispettivamente il percorso del related file, lo start_time e l'end_time. La variabile interna window_duration calcola l'intervallo di tempo per cui la finestra deve essere visualizzata, ossia la differenza tra end_time e start_time.
Alla finestra è aggiunta come contenuto un'immagine (Gtk.Image) il cui percorso corrisponde al primo parametro in ingresso.
Insieme al metodo Build() che costruisce la finestra viene fatto a questo punto partire un timer, di intervallo uguale a window_duration, allo scadere del quale sarà eseguito il metodo handler window_close e la finestra sarà distrutta tramite il metodo Destroy(). Si noti l'utilizzo di Glib.Timeout piuttosto che di un timer classico di .NET (System.Timers). Questo perchè Glib.Timeout viene invocato direttamente nel thread dell'interfaccia grafica, diminuendo i tempi di risposta da parte delle finestre.
Il metodo utilizzato come handler da questo tipo di timer, oltre ad eseguire eventuali istruzioni, restituisce un valore boolean. Se il valore restituito è true, il timer viene fatto ripartire, viceversa se il valore restituito è false il timer viene fermato.

3.2.4 L'algoritmo di creazione dello slideshow

A questo punto resta da definire un algoritmo che mostri le varie finestre temporizzate in concomitanza con la riproduzione del brano. Ciò è realizzato per mezzo di un ulteriore timer Glib.Timeout, inizializzato con l'inizio della riproduzione della traccia audio, come segue:

```
GLib.Timeout.Add (10, new GLib.TimeoutHandler (update_status));
```

Tramite l'inizializzazione di uno StopWatch controlliamo lo scorrere del tempo a partire dall'inizio della riproduzione.

```
Stopwatch st = new Stopwatch();
st.Start();
```

Il timer ha una durata di 10 millisecondi, allo scadere della quale esegue il metodo update_status(), riportato qui di seguito:

```
int i=0;

bool update_status ()
{
    if((media.isPlaying ==true) &&  (i < relfilename.Count))
    {
        if (st.ElapsedMilliseconds < Convert.ToDouble( start_time[i]))
         {
             return true;
         }
         else
         {
             rfw = new relatedFileWindow(relfilename[i].ToString(),
             Convert.ToDouble(start_time[i]), Convert.ToDouble( end_time[i]));
              rfw.Show();
              relfilewindows.Add(rfw);
              i++;
              return true;
         }
    }
    else
    {
        i=0;
         return false;
    }
}
```

Inizializzata la variabile int i=0, (initializzata in realtà all'interno del metodo OnPlayClick(), ma qui riportata per praticità) tale metodo effettua ogni 10 millisecondi un controllo su più condizioni tramite 2 if annidati. Se il file audio è in riproduzione, e la variabile i è minore del conteggio totale dei related files, il metodo passa alla valutazione successiva, ovvero valuta se il tempo trascorso (restituito attraverso la proprietà ElapsedMilliseconds della classe StopWatch) sia minore dell'i-esimo valore contenuto nell'ArrayList start_time. In caso affermativo il metodo restituisce il valore

true, ossia viene rieseguito. Viceversa, nel caso in cui il tempo trascorso sia uguale all'i-esimo elemento di start_time, viene costruita una finestra appartenente alla classe relatedFileWindow, avente come parametri in ingresso il percorso dell'i.-esimo elemento dell'ArrayList relfilename, l'i-esimo start_time e l'i-esimo end_time. La finestra viene quindi inserita in un ArrayList chiamato relfilewindows (dichiarato altrove) che tiene conto di tutte le finestre aperte. La variabile i viene incrementata di 1 e il metodo restituisce ancora un boolean true, continuando ad essere eseguito. Nel caso invece non si verifichino le condizioni nella prima valutazione, ossia il file audio non sia più in riproduzione o la variabile i abbia raggiunto il conteggio totale dei related files, il metodo azzera la variabile i e restituisce un boolean false, fermando il timer e smettendo così di essere eseguito.

3.2.5 Distruzione dello slideshow

L'algoritmo per la distruzione dello slideshow, che entra in gioco nel momento in cui la riproduzione dell'audio venga interrotta (tipicamente alla pressione del tasto Stop da parte dell'utente), non fa altro che distruggere tutte le finestre create (ed aggiunte all'Arraylist relfilewindows) dall'algoritmo precedente attraverso un foreach.

```
if (relfilewindows.Count!= 0)
{
    foreach (relatedFileWindow rfwin in relfilewindows)
    {
        rfwin.Destroy();
    }
    relfilewindows.Clear();
}
```

Possiamo quindi schematizzare il funzionamento del programma nel seguente modo:

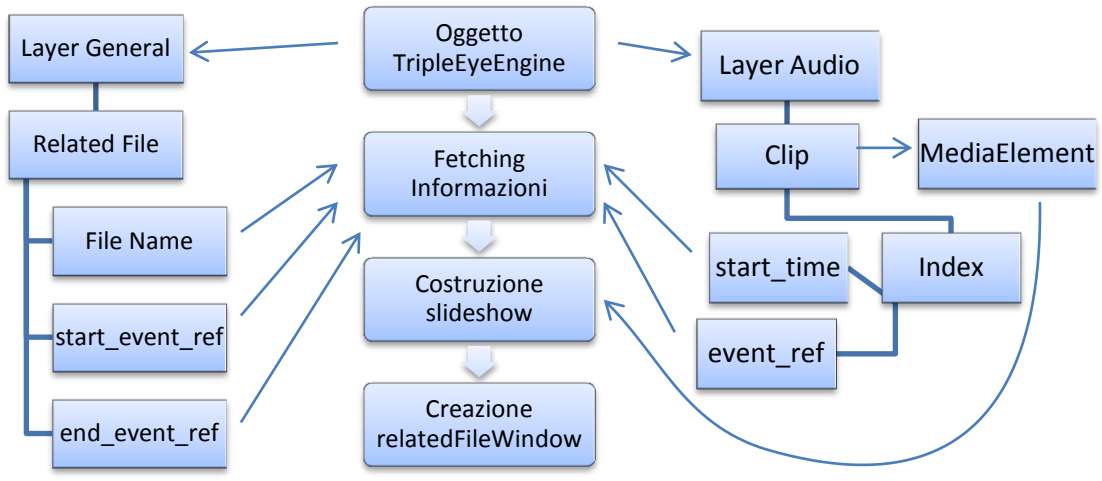


Figura 3.2 – Schema del funzionamento dell'applicazione

3.3 Sviluppo dell'interfaccia grafica

La creazione dell'interfaccia grafica (o GUI, acronimo per il termine inglese Grapical User Interface) dell'applicazione si è svolta di pari passo con lo sviluppo del codice, tramite il designer Gtk# Stetic, incluso in MonoDevelop. Seguono adesso una descrizione del layout dell'applicazione, eseguita in ambiente Ubuntu, ed un approfondimento su alcuni aspetti implementativi dell'interfaccia dell'applicazione.

3.3.1 Descrizione della GUI

Al primo avvio l'applicazione si presenta come una semplice finestra, il cui unico menù File ha le sole opzioni Open ed Exit, che rispettivamente permettono di aprire un documento IEEE 1599 e di chiudere l'applicazione:

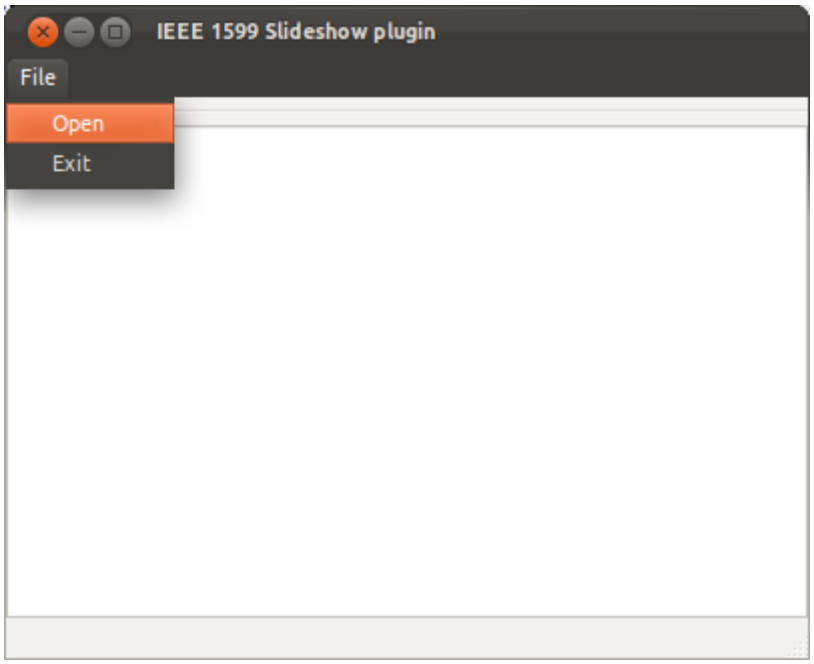


Figura 3.3 - Finestra del programma all'apertura

All'apertura di un documento IEEE 1599 l'applicazione mostra all'utente alcune informazioni.
In particolare possiamo distinguere nella figura: una label (1) in basso, che informa l'utente dell'avvenuto caricamento del file, e una lista (2) dei file audio disponibili per essere riprodotti.

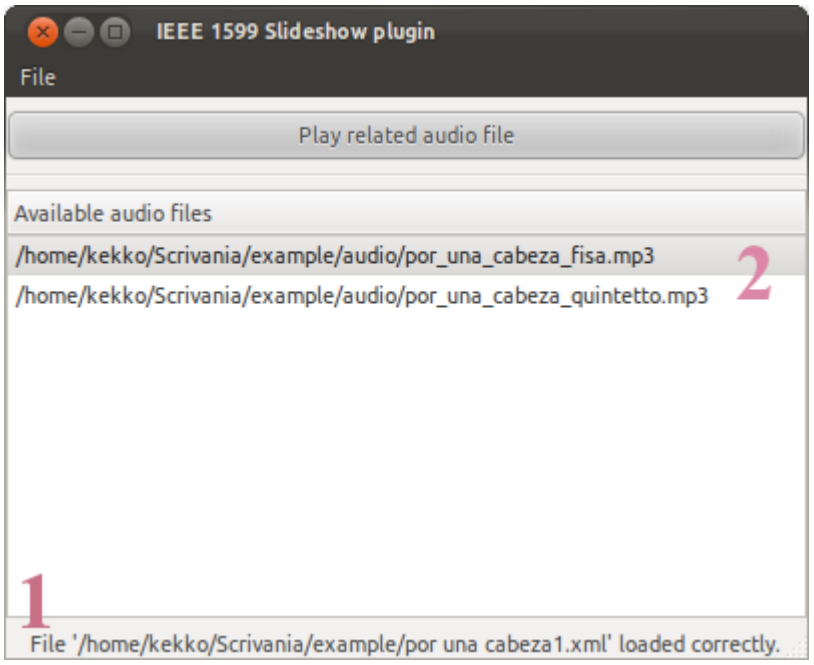


Figura 3.3 - Finestra del programma all'apertura di un file

Facendo doppio click su uno dei file audio presenti nella lista, l'utente viene informato da un messaggio che il file è stato selezionato ed è pronto per la riproduzione.

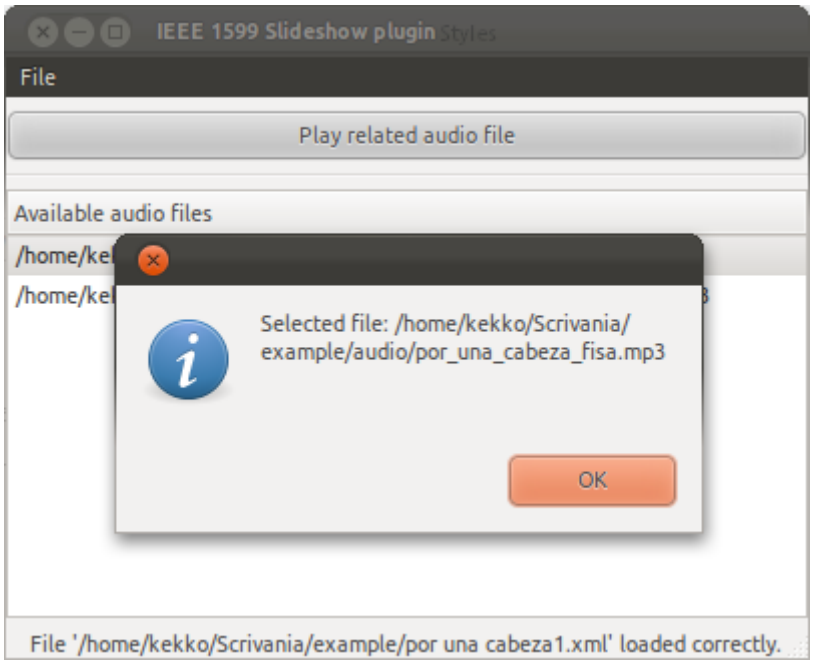


Figura 3.4 – Finestra di dialogo del programma che mostra il file audio selezionato

A questo punto, cliccando sul pulsante "Play related audio file", l'utente fa partire la riproduzione del file audio selezionato, assieme allo slideshow.

Possiamo vedere una schermata del software in esecuzione nell'immagine seguente:

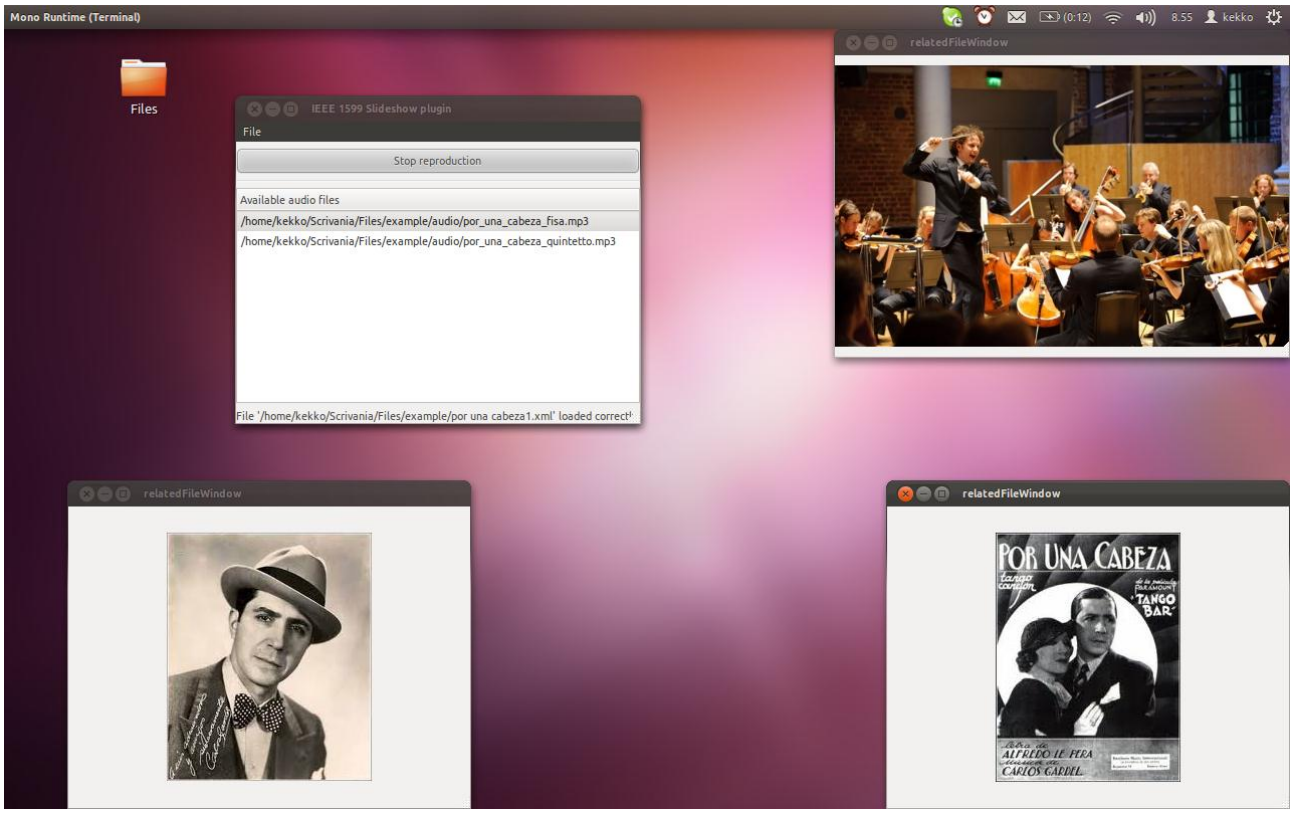


Figura 3.5 - Slideshow in esecuzione

Le immagini presenti nei related files vengono visualizzate in forma temporizzata e mostrate all'interno di finestre temporizzate separate.

3.3.2 Aspetti implementativi dell'interfaccia grafica

La GUI della finestra principale è stata realizzata utilizzando il designer Gtk#.
Sotto questo aspetto le librerie Gtk si rivelano versatili ed immediate, e l'IDE MonoDevelop offre allo sviluppatore un workflow simile a Visual Studio. Così come nell'IDE di Microsoft, anche qui abbiamo la possibilità di trascinare widget da una toolbar a lato dello schermo per inserirli nella finestra, e di assegnare ad ogni widget determinati eventi (ad esempio la pressione di un bottone, la spunta di una checkbox, e così via) che vengano gestiti da un handler.
Di seguito un'immagine che raffigura il progettatore durante le fasi di creazione della finestra principale del software:

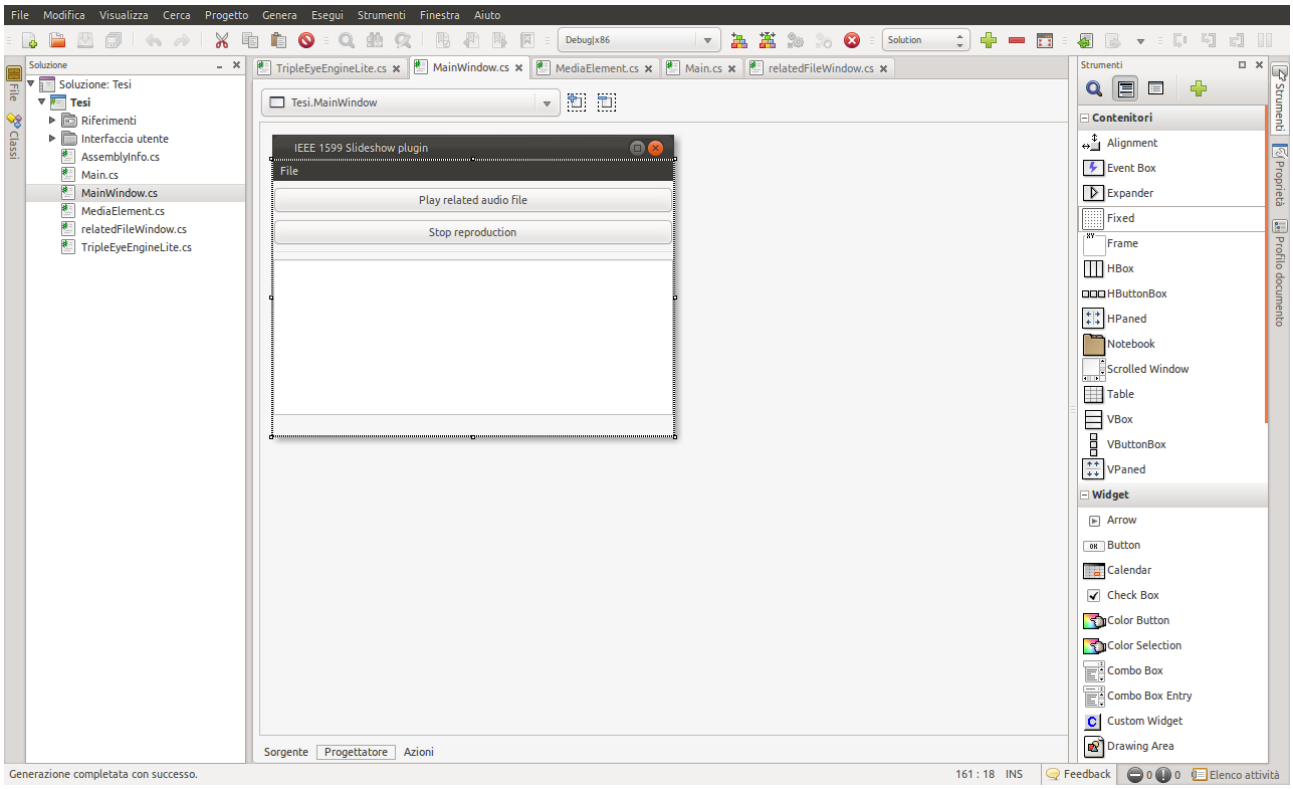


Figura 3.6 - Finestra del progettatore GTK#

Una particolarità menzionabile di Gtk è sicuramente il modo in cui i vari widget sono posizionati all'interno di una finestra. Ogni widget della libreria Gtk deve essere logicamente incapsulato all'interno di un contenitore. Il widget andrà così ad occupare all'interno di quest'ultimo tutto lo spazio che ha a disposizione. In Gtk la finestra stessa è considerata un contenitore, pertanto inserendo, ad esempio, un bottone direttamente all'interno della finestra vedremo il bottone espandersi fino ad occupare tutta la superficie disponibile. Per mezzo dei contenitori è però possibile controllare il layout dei vari widget inseriti. Possono essere infatti utilizzati particolari tipi di contenitore, chiamati VBox e Hbox (e loro varianti), che permettono di suddividere la superficie

della finestra in rispettivamente righe e colonne, come se fosse una tabella, e di collocare i widget all'interno delle celle, di cui si può controllare la dimensione.
Per la realizzazione dell'interfaccia del software è stata utilizzata una Vbox in cui sono stati inseriti tutti i widget.
I bottoni Play e Stop hanno i corrispettivi eventi associati OnPlayClick() e OnStopClick(), che implementano nell'interfaccia grafica gli algoritmi visti nel paragrafo 3.2. In particolare nel metodo OnPlayClick sono avviati il timer che controlla la temporizzazione dello slideshow, e la riproduzione del file audio, mentre il metodo OnStopClick stoppa la riproduzione e contemporaneamente esegue la distruzione dello slideshow. I due pulsanti inoltre si escludono mutualmente attraverso la proprietà Visible.

```
protected void OnPlayClick (object sender, System.EventArgs e)
{
    playButton.Visible=false;
    stopButton.Visible=true;
    GLib.Timeout.Add (10, new GLib.TimeoutHandler (update_status));
    media.Play();
}
```

```
protected void OnStopClick (object sender, System.EventArgs e)
{
    playButton.Visible=true;
    stopButton.Visible=false;
    media.Stop();
    elapsed_time= 0;

    if (relfilewindows.Count!= 0)
    {
        foreach (relatedFileWindow rfwin in relfilewindows)
        {
            rfwin.Destroy();
        }
        relfilewindows.Clear();
    }
}
```

La lista dei file Audio disponibili è stata invece creata utilizzando un widget Treeview, inserito tramite il designer ed inizialmente vuoto, a cui vengono aggiunti contenuti al momento dell'apertura del file attravero il metodo populate_treeview(), che scorre il dizionario MediaIstances tramite un foreach e va ad appendere alla Treeview il contenuto delle chiavi (ovvero i percorsi relativi dei file multimediali), filtrati in base alla proprietà IsVideo del corrispettivo valore.

```
private void populate_treeview()
{
    Gtk.TreeViewColumn FileColumn = new Gtk.TreeViewColumn ();
    FileColumn.Title = "Available audio files";
    treeview1.AppendColumn (FileColumn);
    Gtk.ListStore fileListStore = new Gtk.ListStore (typeof (string));
    treeview1.Model = fileListStore;
    foreach(var item in engine.MediaInstances)
    {
        if (item.Value.IsVideo == true)
        {
        }
        else
        {
            fileListStore.AppendValues(item.Key);
        }
    }
    Gtk.CellRendererText fileNameCell = new Gtk.CellRendererText ();
    FileColumn.PackStart (fileNameCell, true);
    FileColumn.AddAttribute (fileNameCell,"text",0);
    media = null;
}
```

Il doppio click su una delle righe della treeview genera l'evento treeview1_RowActivated, che implementa l'algoritmo visto nel paragrafo 3.2.1 (che non sarà nuovamente riportato) per il fetching dei dati relativi alla temporizzazione dei related files, oltre ad inizializzare l'oggetto MediaElement media (il file audio da riprodurre) utilizzando il percorso del file selezionato dalla treeview.

```
protected void treeView1_RowActivated (object o, Gtk.RowActivatedArgs args)
{
    TreeIter iter = new TreeIter();
    treeview2.Model.GetIter(out iter, args.Path);
    string curRow =treeview2.Model.GetValue(iter, 0).ToString();
    media = new MediaElement(curRow);
    ...
}
```

4. Test dell'applicazione e analisi dei risultati

Il presente capitolo è dedicato all'analisi del comportamento del software sviluppato, per verificare il suo corretto funzionamento. I test sono volti ad analizzare 2 casistiche particolari (singolo file audio, e presenza di file audio multipli), e si vuole soprattutto testare il diverso comportamento del programma in merito alla temporizzazione dei related files in presenza di più file audio.
I test seguenti sono stati eseguiti per mezzo di messageDialog che stampano a video i valori che assume la variabile di tipo Stopwatch (System.Diagnostics.Stopwatch) st, che controlla il tempo di riproduzione del brano audio a partire dall'esecuzione del metodo OnPlayClick, ogni volta che viene visualizzata una finestra contenente un related file.
Un ulteriore stopwatch, opportunamente posizionato all'interno della classe relatedFileWindow, ci permette inoltre di misurare, tramite altri messageDialog, il tempo per cui la finestra permane sullo schermo prima di essere distrutta.

4.1 Test numero 1, singolo file audio, precisione del software

Per effettuare il test seguente sarà utilizzato un file IEEE1599 appositamente modificato, in cui sono stati inseriti dei related files con una precisa temporizzazione conosciuta.
In questo modo si può verificare che effettivamente il software riproduca correttamente, nell'ordine giusto e con la giusta temporizzazione i file.

Questo è il campo <related files> all'interno del layer General del file IEEE 1599 utilizzato.

```
<related_files>
      <related_file file_name="relatedfiles/gardel.jpg" file_format="image_jpeg"
      encoding_format="image_jpeg" description="Carlos Gardel"
      start_event_ref="Cello_5_voice0_measure3_ev0"
      end_event_ref="Violino_1_1_voice0_measure5_ev1"/>
      <related_file file_name="relatedfiles/Aurora_Orchestra.jpg" file_format="image_jpeg"
      encoding_format="image_jpeg" description="Aurora Orchestra"
      start_event_ref="Violino_1_1_voice0_measure4_ev1"
      end_event_ref="Violino_1_1_voice0_measure5_ev1"/>
      <related_file file_name="relatedfiles/locandina.jpg" file_format="image_jpeg"
      encoding_format="image_jpeg" description="Locandina"
      start_event_ref="Violino_1_1_voice0_measure4_ev3"
      end_event_ref="Cello_5_voice0_measure5_ev1"/>
</related_files>
```

Analizzando il layer Audio del documento si possono ritrovare gli event_ref e i relativi timing, ed è possibile compilare una tabella in cui compaiono il nome del file, lo start_event_ref ed il suo tempo assoluto Ts, l'end_event_ref ed il suo tempo assoluto Te.

Nome del file	start_event_ref	Ts (s)	end_event_ref	Te (s)
Gardel.jpg	Cello_5_voice0_measure3_ev0	3,98 s	Violino_1_1_voice0_measure5_ev1	8,46 s
Aurora_Orchestra.jpg	Violino_1_1_voice0_measure4_ev1	6,18 s	Violino_1_1_voice0_measure5_ev5	9,37 s
locandina.jpg	Violino_1_1_voice0_measure4_ev3	6,62 s	Cello_5_voice0_measure5_ev1	8,24 s

Per ogni related file possiamo ottenere inoltre il tempo di permanenza Tp della finestra, sottraendo i 2 valori Te – Ts.
Si ha quindi:

Nome del file	Tp (s)
Gardel.jpg	8,46-3,98 = 4,48 s
Aurora_Orchestra.jpg	9,37-6,18=3,19 s
locandina.jpg	8,24-6,62=1,62 s

Sottoponendo l'applicazione al test si ottengono per i 3 file, come si può vedere dall grafico sottostante, raffigurante sulla sinistra i valori Ts e sulla destra i valori Tp, risultati che sono lontani dalla perfezione ma si avvicinano ai valori aspettati.

Nome del file	Ts (s)	Tp (s)
Gardel.jpg	3,99 s	4,46 s
Aurora_Orchestra.jpg	6,21 s	3,18 s
locandina.jpg	6,62 s	1,61 s

Test successivi hanno riportato risultati molto simili a quelli appena illustrati. I valori oscillano sempre in un intervallo di 1-3 centesimi di secondo attorno al valore aspettato. L'applicazione è molto lontana dalla precisione che sarebbe reputata necessaria in un contesto di sincronizzazione audio realtime, ma dobbiamo ricordare che, così come la connessione tra il layer General e lo Spine è molto debole, così anche le esigenze di realtime sono meno pesanti. E' infatti tollerabile che, in uno slideshow di immagini non strettamente correlate alla traccia, ma che portino informazioni aggiuntive, possano presentarsi delle imprecisioni dell'ordine di centesimi di secondo, purchè questo non appesantisca o peggiori l'esperienza d'uso dell'utente finale.

4.2 Test numero 2, più file audio, con temporizzazioni diverse dei related files

Nel caso in cui il documento IEEE 1599 presenti più file audio, e quindi diversi indexing, tramite il software è possibile selezionare uno dei file audio a scelta. Questo test ha lo scopo di verificare se effettivamente la temporizzazione dei related files cambi in maniera corretta. Prendendo lo stesso documento IEEE 1599 utilizzato nel test precedente, ma selezionando questa volta una diversa traccia audio, possiamo ricavare nuovamente una tabella simile a quella in figura, ma contenente i nuovi valori per la nuova clip.

Nome del file	start_event_ref	Ts (s)	end_event_ref	Te (s)
Gardel.jpg	Cello_5_voice0_measure3_ev0	5,15	Violino_1_1_voice0_measure5_ev1	10,17
Aurora_Orchestra.jpg	Violino_1_1_voice0_measure4_ev1	7,21	Violino_1_1_voice0_measure5_ev5	11,24
locandina.jpg	Violino_1_1_voice0_measure4_ev3	8	Cello_5_voice0_measure5_ev1	9,88

E'possibile calcolare i Tp per la permanenza della finestra anche in questo caso, ottenendo la seguente tabella.

Nome del file	Tp (s)
Gardel.jpg	10,17-5,15 = 5,02 s
Aurora_Orchestra.jpg	11,24-7,21=4,03 s
locandina.jpg	9,88-8= 1,88 s

Sottoponendo nuovamente l'applicazione al test si ottengono per i 3 file le seguenti coppie di valori

Nome del file	Ts (s)	Tp (s)
Gardel.jpg	5,16 s	5,01 s
Aurora_Orchestra.jpg	7,24 s	4,06 s
locandina.jpg	8 s	1,91 s

Ancora una volta ci troviamo davanti a valori che sono vicini ai risultati aspettati. L'applicazione risulta quindi funzionare correttamente, modificando il timing delle finestre a seconda della clip selezionata, pur presentando ancora margini di migliorabilità nell'ambito della precisione della temporizzazione.

5. Conclusioni, considerazioni, sviluppi futuri

Il presente lavoro dimostra le potenzialità del framework Mono in ambito multimediale, e di come esso possa essere utilizzato, in unione con la tecnologia SDL.NET, per lo sviluppo di applicazioni altamente portabili. Lo scopo primario di questo lavoro era infatti, prima ancora della realizzazione stessa del software, la ricerca di nuovi spunti per lo sviluppo del progetto multipiattaforma TripleEye. Si è appurato che SDL e Gtk# sono strade percorribili nello sviluppo di software compatibile con il framework .NET ma indipendente dalla piattaforma Windows.

Tutto ciò permette di sfruttare la potenza e la flessibilità del linguaggio C# in un ambiente molto più vasto dell'ecosistema Microsoft, e contemporaneamente pone le basi per un futuro in cui il formato IEEE 1599 possa essere lo standard de-facto della fruizione multimediale, sia essa online od offline, indipendentemente dal dispositivo utilizzato per fruire informazioni.

Sicuramente, assieme ad un perfezionamento, e ad un ampliamento delle possibilità di interazione con i related files, uno degli sviluppi futuri possibili per questa applicazione sarà l'implementazione completa all'interno del ben più ampio progetto TripleEye.

Gli algoritmi implementati nel software ed argomentati nel corso di questo elaborato sono perfezionabili, come ha dimostrato il capitolo relativo ai test, ma saranno, spero, spunto di ricerca e miglioramento, dentro e fuori dal Laboratorio di Informatica Musicale, in un percorso di cui queste pagine hanno segnano solo un piccolissimo passo.

6. Bibliografia e sitografia

6.1 Bibliografia

[1] *Ludovico, L.A.*: An XML Multi-layer Framework For Music Information Description. Ph.D. diss., Università degli Studi di Milano, Milan (2006)

[2] *Ludovico, L.A.*: Key concepts of the IEEE 1599 Standard. In: Baggi, D., Haus, G. (eds.) Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects. pp. 15-26. IEEE CS, Lugano (2008), ISBN: 88-7595-010-5

[3] *Ludovico, L.A.*: Outline of the MX Standard. In: Avrithis, Y., Buitelaar, P., Falcidieno, B., Kompatsiaris, I., Spagnuolo, M. (eds.) Semantic multimedia : Second International Conference on Semantic and Digital Media Technologies, SAMT 2007, Genoa, Italy, December 5-7, 2007 : proceedings. LNCS, vol. 4816, pp. 196-199. Springer-Verlag, Heidelberg / Berlin (2007),
ISBN: 978-3-540-77033-6

[4] *John Sharp*: Microsoft Visual C# 2008 Step by Step, Microsoft Press (2007),
ISBN: 8861141137

[5] *Edd Dumbill*, Niel M. Bornstein: Mono: a Developer's Notebook, O'Reilly (2004),
ISBN: 0596007922

[6] *Mark Mamone*: Practical Mono, Apress (2005), ISBN: 978-1-59059-548-0

6.2 Sitografia

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

http://www.mono-project.com/

http://docs.xamarin.com/

http://cs-sdl.sourceforge.net/

7. Ringraziamenti

Mi sento in dovere di ringraziare tutte le persone che mi sono state vicino e mi hanno sostenuto nello sviluppo del software e di questo elaborato finale.
In primis vorrei ringraziare il Professor Ludovico, il Dottor Baratè e tutto il personale del Laboratorio di Informatica Musicale per la disponibilità, la collaborazione e l'interesse dimostrato nei confronti del mio lavoro.
Ringrazio i miei genitori, Andrea e Roberta, che mi sono sempre stati vicino quando ero lontano da casa per parecchi mesi, e mi hanno aiutato economicamente e psicologicamente, nonostante le avversità della vita, nel corso di questi anni di studio, e durante lo svolgimento dell'elaborato finale, con una parola di conforto anche nelle giornate più grigie.
Ringrazio tutti i colleghi e compagni di questa singolare avventura, che hanno riempito di bei momenti le ore di lezione e le maratone di studio pre-esame: Banzi, Limo, Sam, Ale, Jack, Ste, Simon...è stato un onore, oltre che uno spasso, conoscervi.
Ringrazio mia cugina Simonetta, per avermi ospitato e fatto contemporaneamente da mamma, sorella e migliore amica nei primi due anni di vita a Milano.
Ringrazio Giuliana e Valentina, per tutti gli aiuti ricevuti nei primi mesi di vita nella città, e per le belle, semplici serate fatte di tisane e risate davanti alla tv.
Ringrazio Ingha, per le assurde serate, la tenerezza, le litigate, e l'amore di cui ha saputo circondarmi e riempirmi. Ci saranno altre gru su cui arrampicarci per ammirare estasiati il panorama, ne sono certo.
Ringrazio Laura e Marco per essere stati degli ottimi coinquilini, ma soprattutto dei degni avversari nelle serate passate a massacrarci le braccia giocando alla Wii.
Ringrazio Matteo per essere l'amico "smanettone" sopracitato. Se sono qui è anche un po' suo merito (o colpa?).
Un grazie di cuore va a Fil, Cipo, Artu e Giovà, che si sono rivelati il mio gruppo in tutti i sensi. E quando dico gruppo, intendo molto di più che una semplice band.
Ringrazio Fausto, che ha creduto in me e nei nostri progetti così tanto da aver camminato al mio fianco fino a qui. E il bello deve ancora venire.
Infine un enorme grazie va a mio fratello Giagi, che purtroppo oggi non è più qui, ma di cui non ho mai scordato i sorrisi e i consigli. Gli ho fatto una promessa, una volta. E, costi quel che costi, quella promessa diventerà realtà.