Relazione per “JWave”

Dario Cantarelli, Alessandro Martignano, Aleksejs Vlasovs

31 Maggio 2016

Sommario

JWave è un un'applicazione open source, realizzata in linguaggio Java, orientata alla riproduzione e all'editing di file audio.

L'applicazione e questa relazione sono stati realizzati come progetto per l'esame di Programmazione ad Oggetti del corso di Ingegneria e Scienze Informatiche dell'Università di Bologna, A.A. 2015/2016.

Indice

1 Analisi **2**
1.1 Requisiti 2
1.2 Analisi e modello del dominio 3

2 Design **7**
2.1 Architettura 7
2.2 Design dettagliato 8

3 Sviluppo **18**
3.1 Testing automatizzato 18
3.2 Metodologia di lavoro 19
3.3 Note di sviluppo 20

4 Commenti finali **22**
4.1 Autovalutazione e lavori futuri 22

A Guida utente **24**
A.1 Player mode tutorial 24
A.2 Editor mode tutorial 25

Capitolo 1

Analisi

1.1 Requisiti

Obiettivo del programma è offrire la possibilità di caricare file audio dal proprio computer, attraverso una interfaccia semplice ed intuitiva. Questi file, che tipicamente prendono il nome di canzoni, tracce o brani musicali, potranno essere organizzati in gruppi detti "playlist" e riprodotti a discrezione dell'utente. Inoltre si vuole permettere all'utente la modifica di un file caricato in modo da crearne uno nuovo derivato dall'originale, diverso per struttura e/o contenuto.

L'applicazione è stata concettualmente suddivisa in tre blocchi principali: il player fornisce le funzionalità di riproduzione e di organizzazione di una libreria musicale; l'editor permette di modificare una canzone appartenente alla libreria musicale e l'interfaccia grafica (GUI) permette all'utente di interagire con i due componenti citati.

Player

- Si dovrà disporre di un riproduttore che permetta all'utente di leggere e spostarsi liberamente all'interno di diversi tipi di file audio.
- L'utente potrà inoltre modificare i metadati dei file con estensione ".mp3".
- Si vuole anche permettere l'inserimento di effetti, componenti che modificano il flusso di dati del riproduttore, attivabili e modificabili dall'utente durante la riproduzione.
- Per gestire la libreria musicale sarà necessario offrire all'utente la possibilità di poter riorganizzare a piacimento la disposizione dei bra-

ni all'interno delle playlist e di salvare queste modifiche nel proprio computer.

Editor

- Per orientarsi all'interno della traccia da modificare l'utente dovrà averne una vista fedele in un formato intuitivo.
- Per apportare modifiche alla traccia l'utente avrà inoltre bisogno di un insieme di controlli che permettono di comunicare le varie modifiche da effettuare sulla traccia e eventuali parametri necessari a queste modifiche.
- Una traccia potrà essere infine salvata nella sua versione modificata sul computer dell'utente.

User Interface

- L'applicazione dovrà essere dotata di un'interfaccia utente semplice e intuitiva.
- Al contempo, l'interfaccia dovrà permettere all'utente di effettuare operazioni relativamente complesse quali spostarsi con precisione lungo una traccia per poterla modificare.
- L'interfaccia dovrà essere il più possibile dinamica e costantemente aggiornata sullo stato della riproduzione, indipendentemente dall'intervento dell'utente.

1.2 Analisi e modello del dominio

Il sistema software realizzato è costruito intorno ad un modello di file audio, che costituisce l'elemento di base dell'intero sistema. Tutte le entità del sistema sono volte ad esporre, in modi diversi, questa entità all'utente finale.

Uno dei principali problemi da risolvere in fase di progettazione è disaccoppiare i vari componenti per ridurre al minimo le interdipendenze. L'entità player avrà il compito di riprodurre le canzoni e non dovrà dipendere da altri componenti se non dalla canzone. Allo stesso modo l'entità che si occupa di organizzare le canzoni in delle liste (playlists) dovrà dipendere anch'essa soltanto dall'elemento che organizza, ovvero le canzoni. L'entità editor che permetterà di modificare le canzoni dovrà dipendere soltanto da queste e non da altri componenti esterni. Ad unire questi elementi indipendenti ci

dovrà essere un'altra entità di tipo controllore, che permetterà di gestire i vari componenti ed esporre le loro funzionalità all'utente in modo interattivo attraverso un'interfaccia grafica.

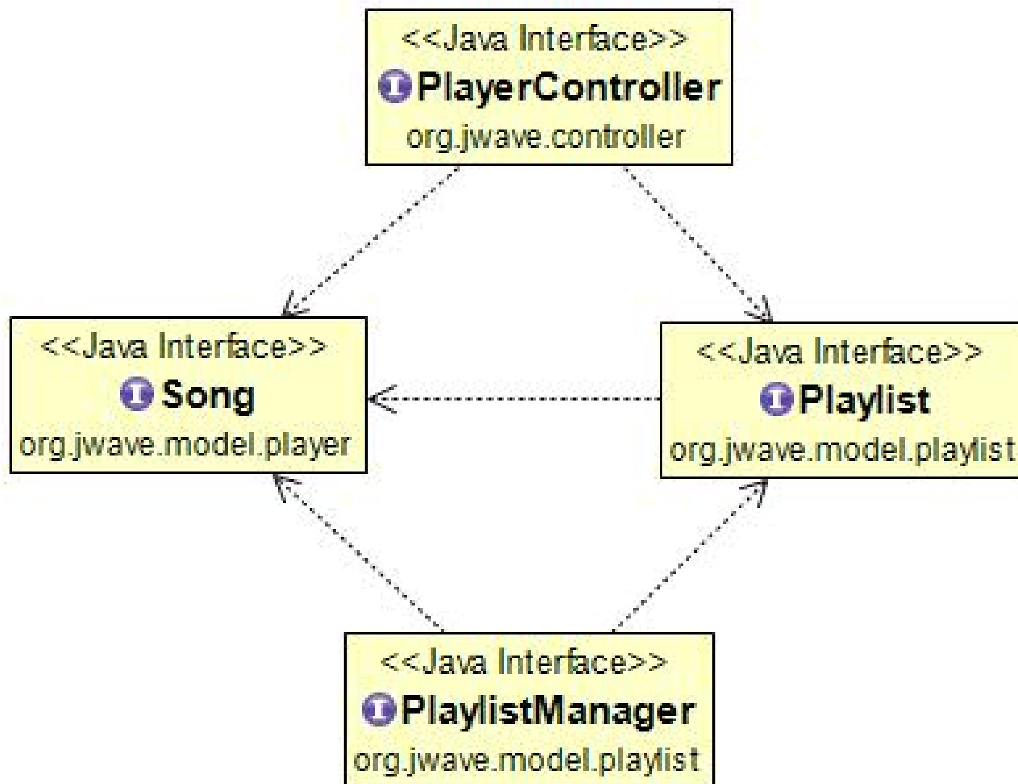

Figura 1.1: Schema UML del modello di dominio del Player.

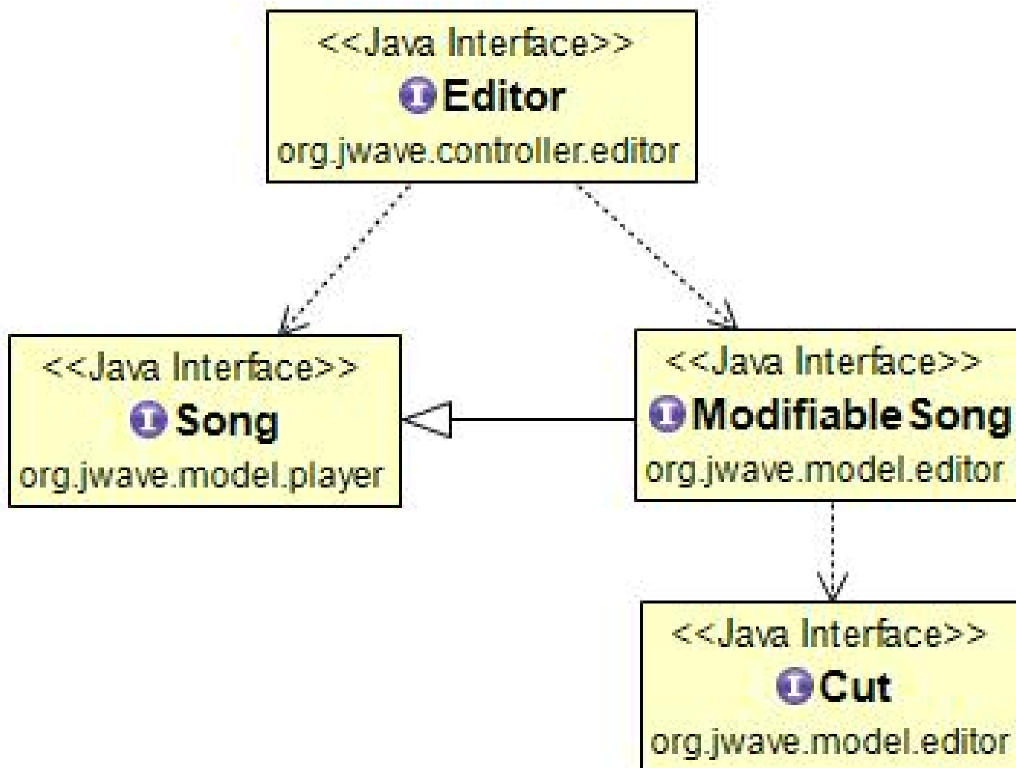

Figura 1.2: Schema UML del modello di dominio dell' Editor.

Player

Il player dovrà separare la parte di riproduzione del singolo file audio dalle modalità di gestione e navigazione delle playlist. Il gestore di playlist dovrà essere in grado di navigare le suddette secondo diverse modalità che l'utente potrà scegliere. La parte impegnativa sta nel decidere come memorizzare efficacemente le playlist in maniera permanente, cosicchè l'utente possa ritrovarsele disponibili all'apertura di una nuova sessione.

Per quanto riguarda la gestione degli effetti audio, sarà impegnativo far aderire componenti che svolgono funzioni diverse come un riproduttore audio ed un effetto ad un protocollo comune che permetta loro di scambiarsi informazioni secondo il modello della "sound chain", dove diversi componenti che processano il flusso di dati proveniente dal riproduttore sono disposti a catena; l'utente può scegliere quali elementi modificano il flusso di dati e quali devono essere "bypassare". La complessità di questo problema e l'impossibilità di risolverlo entro il monte ore a disposizione hanno fatto sì che questa "feature" possa essere sviluppata in futuro.

Editor

Il problema più difficile da risolvere legato al componente dell'editor sarà quello di riuscire a concrettizzare il concetto di modifica. Una modifica implica un cambiamento della canzone, di struttura o di comportamento, esteso ad un certo segmento arbitrario. Le modifiche strutturali devono permettere di cambiare la struttura della canzone, togliendo segmenti o prendendo segmenti da una posizione e spostarli in altre posizioni. Le modifiche comportamentali dovranno permettere invece di modificare come viene riprodotto un certo segmento della traccia, alterando il suo volume o aggiungendo effetti particolari.

La seconda problematica riguarda la riproduzione di un brano musicale modificato. La soluzione, e quindi anche la complessità di questo problema saranno legate alla scelta della libreria da utilizzare per la riproduzione audio e alla soluzione implementata per risolvere la problematica precedente inerente la rappresentazione concreta delle modifiche.

Nonostante la similitudine tra modifiche di tipo strutturale e comportamentale, ognuna di esse richiede una soluzione implementativa diversa, e poiché creare una nuova tipologia di modifica necessita anche la realizzazione di un modo per effettuare questa modifica sulla traccia, un modo per riprodurre la traccia contenente questo nuovo tipo di modifica e un modo per salvare la traccia modificata contentente la modifica, risolvere questo problema, dato il lavoro preesistente, richiederebbe una quantità di ore significativa oltre il monte previsto. Dunque è stato deciso di implementare questa feature in futuro.

User Interface e Controller

La UI dovrà fornire, assieme al controller, un accesso semplice alla modifica del modello e alla lettura di esso, costantemente aggiornata. Le due entità, UI e controller, dovranno lavorare efficacemente insieme ma al tempo stesso essere completamente indipendenti e isolate l'una dall'altra, in modo da poter cambiare facilmente l'entità o il numero delle loro relazioni. Il controller deve infatti garantire la massima espandibilità e accessibilità, sia da parte di componenti grafici che non. La difficoltà principale sarà, nel senso opposto, permettere un accesso alle componenti dell'interfaccia da parte del controller che sia uniforme indipendentemente dal tipo e dall'implementazione della stessa.

Particolare attenzione si dovrà porre alla possibilità di aumentare in futuro la tipologia e il numero delle componenti connesse al controller, permettendo comunque un comune accesso al modello e una corretta sincronizzazione.

Capitolo 2

Design

L'obiettivo dal punto di vista del design è quello di realizzare un programma che riduca al minimo le interdipendenze tra i tre componenti precedentemente citati (player, editor e GUI) e di conseguenza limiti la necessità di modifiche a cascata. Si è quindi deciso di utilizzare il pattern MVC (Model View Controller), dove l'interazione tra i componenti si basa solamente sulle interfacce esposte, senza le implementazioni sottostanti.

2.1 Architettura

Il pattern MVC permette di disaccoppiare totalmente il player/editor dall'interfaccia grafica. Le interfacce dei tre componenti espongono all'esterno solamente i metodi necessari per svolgere le funzionalità principali richieste, tutta l'implementazione invece viene nascosta nelle classi che implementano le suddette interfacce.

2.2 Design dettagliato

Player: Dario Cantarelli

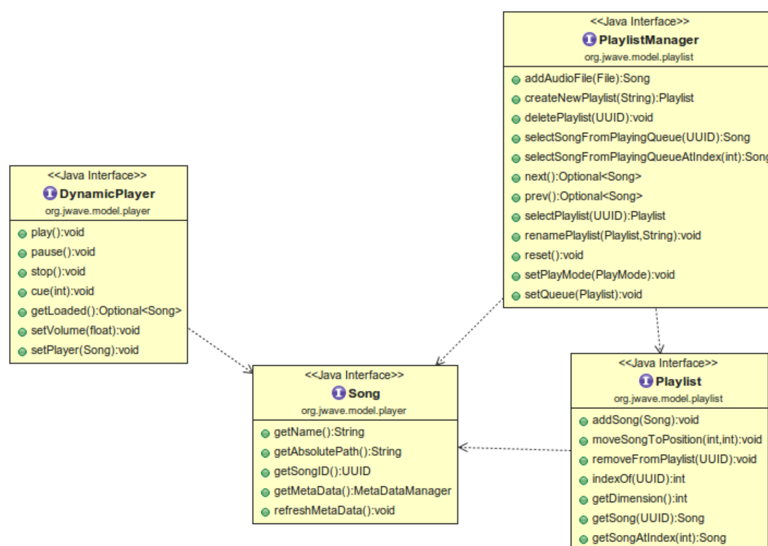

Figura 2.1: Schema UML del modello del player. Il Player è costituito principalmente dalle seguenti parti: Un DynamicPlayer è un oggetto su cui possono essere caricate delle Song. Le Song modellano il concetto di file audio, e possono essere aggregate in collezioni, le playlist. Il PlaylistManager si occupa di creare playlist, di settarle come coda di riproduzione e contiene la strategia per navigarle, ossia la sequenza con cui selezionare i brani contenuti nella playlist corrente. DynamicPlayer è un facade che nasconde l'implementazione sottostante che fa uso di una libreria, tuttavia si è partiti da una modellazione che potesse essere quanto più generica e riutilizzabile possibile, apportando piccoli aggiustamenti solo in caso di necessità.Per poter cambiare in modo agile la strategia di navigazione, si è deciso di adottare il pattern Strategy, delegando il compito ad un oggetto dedicato, il PlaylistNavigator: queta entità viene settata adeguatamente dal PlaylistManager ogni volta che si cambia la PlayMode, una enumerazione che definisce la sequenza di con cui i brani contenuti nella playlist possono essere caricati automaticamente nel DynamicPlayer da un agente esterno presente nel controllore. La costruzione effettiva del navigator avviene attraverso un SimpleFactory di navigator contenuto nel PlaylistManager. La memorizzazione effettiva delle playlist comporta delle operazioni di I/O e viene demandata al controller, più precisamente ad una utility class PlaylistController che si occupa di scrivere sul file system le playlist in una directory di default che viene creata in caso di necesità all'avvio del programma. La riproduzione di brani automatica viene gestita da un'altra classe del controller, il ClockAgent, che si occupa di tenere in relazione un DynamicPlayer e un PlaylistManager e di caricare automaticamente il brano successivo una volta che la riproduzione del brano corrente è terminata. In generale ci si riferisce alle entità del dominio attraverso l'astrazione delle interfacce, secondo i principi di buona programmazione.

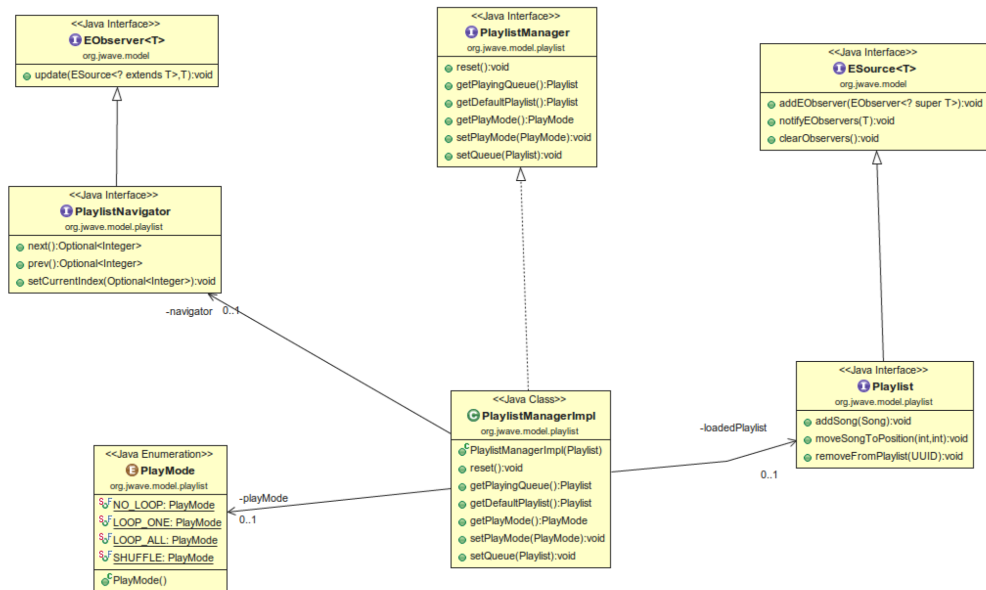

Figura 2.2: Schema UML del attern Observer Un oggetto Playlist quando impostato come coda di riproduzione viene collegato al playlist navigator utilizzando il pattern Observer. Ogni volta che vengono aggiunti/rimossi brani dalla playlist, questa notifica il cambiamento di stato al navigator che può lavorare di conseguenza.

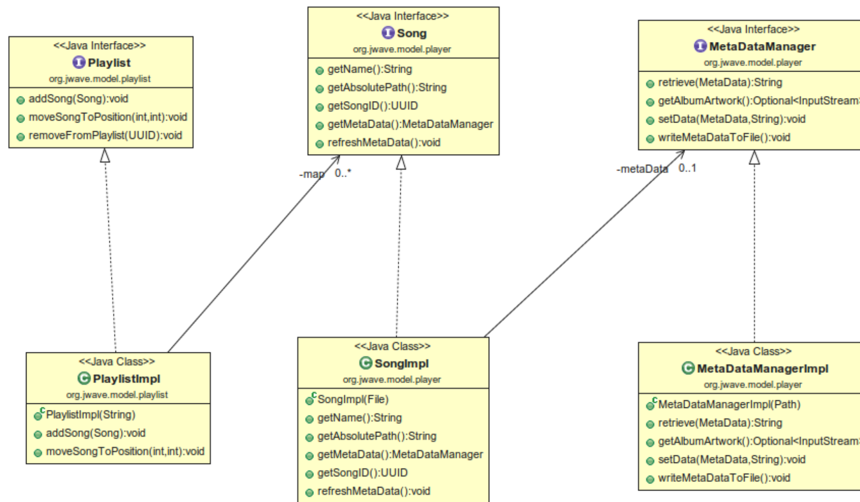

Figura 2.3: Schema UML di una playlist. Una Song è un decoratore di File che contiene una serie di informazioni dette metadati. I metadati vengono gestiti da un MetadataManager contenuto nella Song, che permette all'utente di leggere e riscrivere tali dati direttamente nel file originale. Una Playlist è una collezione di Song, costruita come un decoratore delle Java Collection. In questo modo sia Song che Playlist sfruttano il pattern Decorator per favorire l'aggiunta o la modifica di funzionalità, potendo delegare alcuni compiti agli elementi decorati senza doverle riscrivere e lasciando la possibilità di estendere le implementazioni sottostanti.

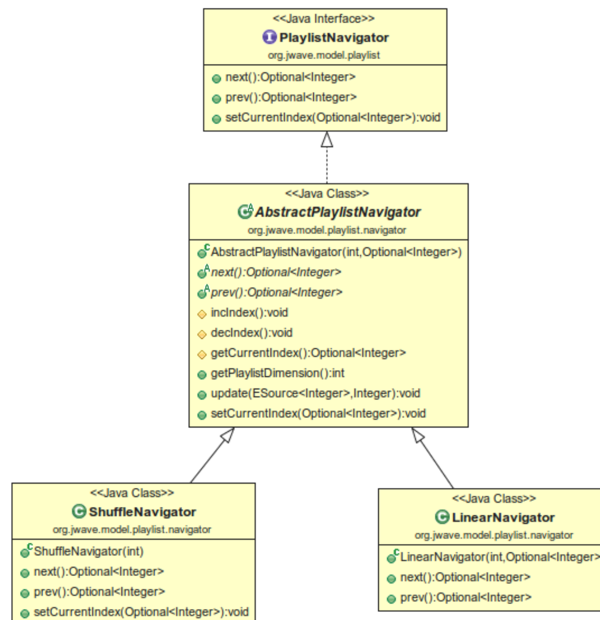

Figura 2.4: Schema UML dello strategy adottato per la navigazione delle playlist. A livello concreto le strategie sono due, LinearNavigator che scorre i brani sequenzialmente, come sono disposti dentro la playlist, e ShuffleNavigator, che sceglie il brano successivo in modo casuale, mantenendo lo storico delle sue scelte.

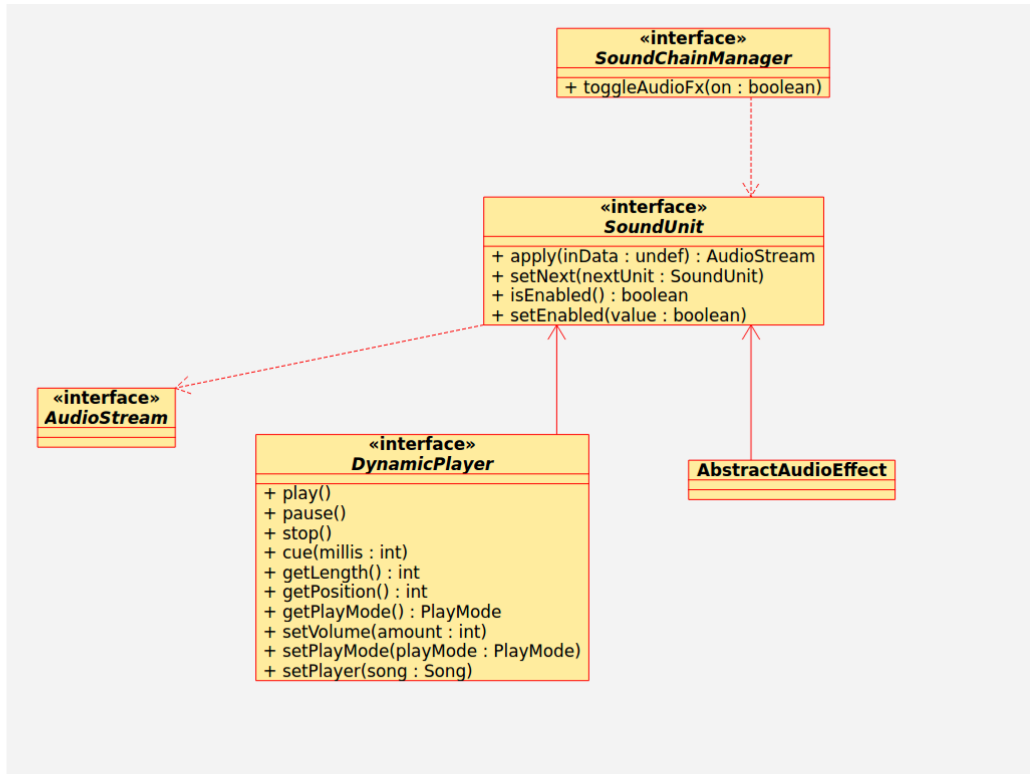

Figura 2.5: Schema UML della parte di gestione degli effetti audio. Si sarebbe adottato il pattern Chain Of Responsibility, che ben si adatta all'esigenza della sound chain, dove ogni componente può decidere se elaborare il flusso di dati in input o passarlo senza modifiche al membro successivo della catena. L'attuale design del Player permetterebbe di implementare questa feature in futuro senza troppe modifiche, facendo sì che DynamicPlayer e tutti gli elementi della sound chain diventino un sottotipo di SoundUnit, creando così un protocollo di comunicazione comune. Un oggetto SoundChainManager si sarebbe occupato di memorizzare la catena e di collegare I suoi componenti, in quanto nel pattern Chain of Responsibility ogni elemento non sa da chi è il successivo. Anche in questo caso SoundChainManager è disaccoppiato dal resto dei componenti, motivo per cui una sua eventuale implementazione non impatterebbe quasi per niente il resto del sistema. Nello schema è riportata la struttura della sound chain. Un SoundChainmanager contiene più SoundUnit collegate fra di loro e può selezionare quali elementi della catena agiscono attivamente sull'AudioStream in input.

Editor: Aleksejs Vlasovs

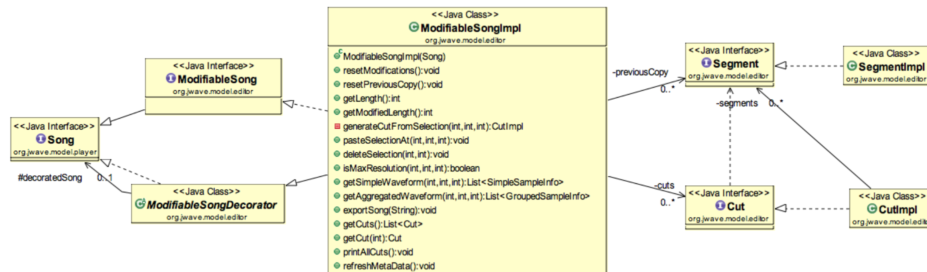

Figura 2.6: Schema UML delle classi che decorano e che servono ad implementare l'interfaccia Song, creando così il decorator, l'interfaccia e la corrispondente implementazione ModifiableSong, tramite il pattern decorator. Questo schema descrive la soluzione implementata per avere canzoni modificabili. Una canzone modificata, oltre alle funzionalità di Song, deve tenere traccia delle modifiche strutturali e comportamentali ad essa effettuate. La motivazione di decorare Song invece che creare una classe nuova è stata presa per riutilizzare il codice già scritto nel componente Player/Playlist, poiché una canzone modificabile è una naturale estensione di una canzone normale. Oltre a riutilizzare Song però, questo permette di utilizzare anche la classe DynamicPlayer, dopo un'altra leggera decorazione, per la riproduzione di brani modificati, aumentando ulteriormente il riuso. La soluzione per apportare modifiche strutturali è stata quella di vedere la canzone come un insieme di Cut relativi alla canzone modificata, e Segment, di cui i Cut sono composti, relativi alla canzone originale, in modo da riuscire ad estrarre dati audio.

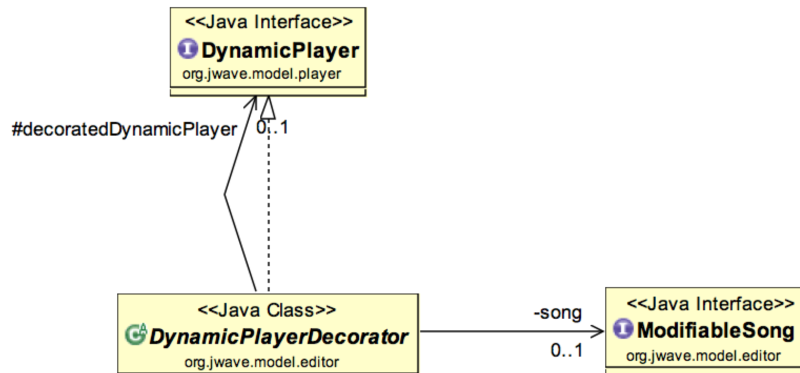

Figura 2.7: Schema UML che mostra le classi coinvolte nella riproduzione di una canzone modificata. Poiché è già stato progettato e implementato un DynamicPlayer per la riproduzione di canzoni normali non modificate (Song), è stato deciso di decorare questo tramite il pattern decorator invece di progettare e implementare un riproduttore apposito per le canzoni modificabili (ModifiableSong, estensioni decorate di Song). La classe DynamicPlayer ha dei metodi che vengono chiamati da un clock con intervalli regolari che verificano lo stato della canzone riprodotta. È stato possibile decorare questi metodi in modo da aggiungere un algoritmo di controllo dei tagli e segmenti. Poiché il DynamicPlayer era già in grado di riprodurre una Song, e poiché una canzone modificabile era composta di un insieme di segmenti mappabili 1 a 1 con una canzone non modificata, si è potuto far riprodurre una canzone modificata a partire dall'audio non modificato.

Interfaccia grafica e controller: Alessandro Martignano

Individuati i due principali ambiti di impiego dell'applicazione, un lettore e un editor, si è deciso di suddividere in altrettante parti le relative interfacce grafiche e controller, quasi ad avere 2 applicazioni diverse che sono comunque in grado di condividere le risorse e i dati di cui necessitano. Per ognuna delle 2 strutture è stata scelta un'implementazione del pattern observer, con una gui che chiama direttamente il controller al verificarsi di un evento da parte

dell'utente, ma è stata posta particolare attenzione alla parte dinamica di aggiornamento di quest'ultima, che può dipendere da eventi non scatenati dall'utente.

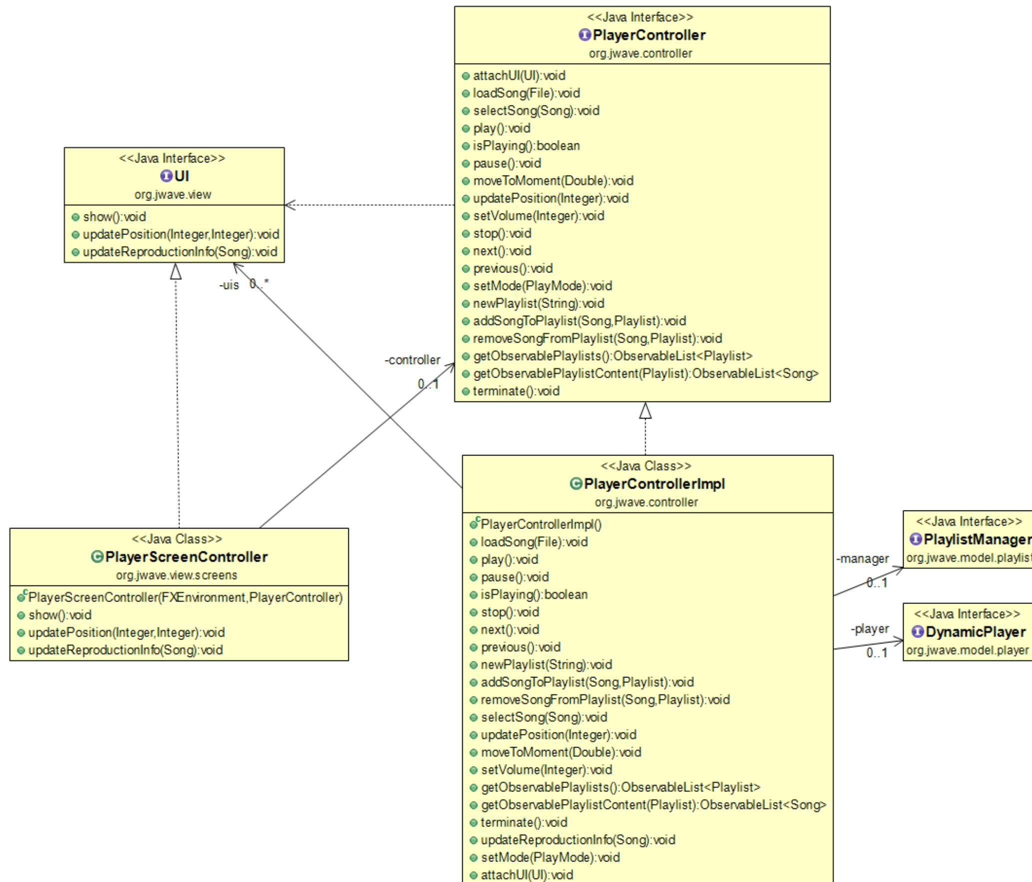

Figura 2.8: Lo schema mostra l'interazione della classe che gestisce la GUI del riproduttore (PlayerScreenControllerImpl) con il relativo controller. Il controller espone tutti i metodi di cui l'interfaccia può aver bisogno ai fini della riproduzione e gestione delle tracce e viene agganciato alla stessa in fase di costruzione, in modo che sia possibile collegare più interfacce grafiche, anche di tipo diverso, allo stesso controller. Per gestire invece le callback che necessitano che la GUI si aggiorni in funzione di eventi indipendenti da essa il controller mette a disposizione delle collezioni osservabili, alle quali le view possono registrarsi senza bisogno di ulteriori chiamate e restare costantemente aggiornate sullo stato interno del modello di loro competenza quali le playlist caricate, le canzoni contenute in esse e i dati relativi allo stato della riproduzione.

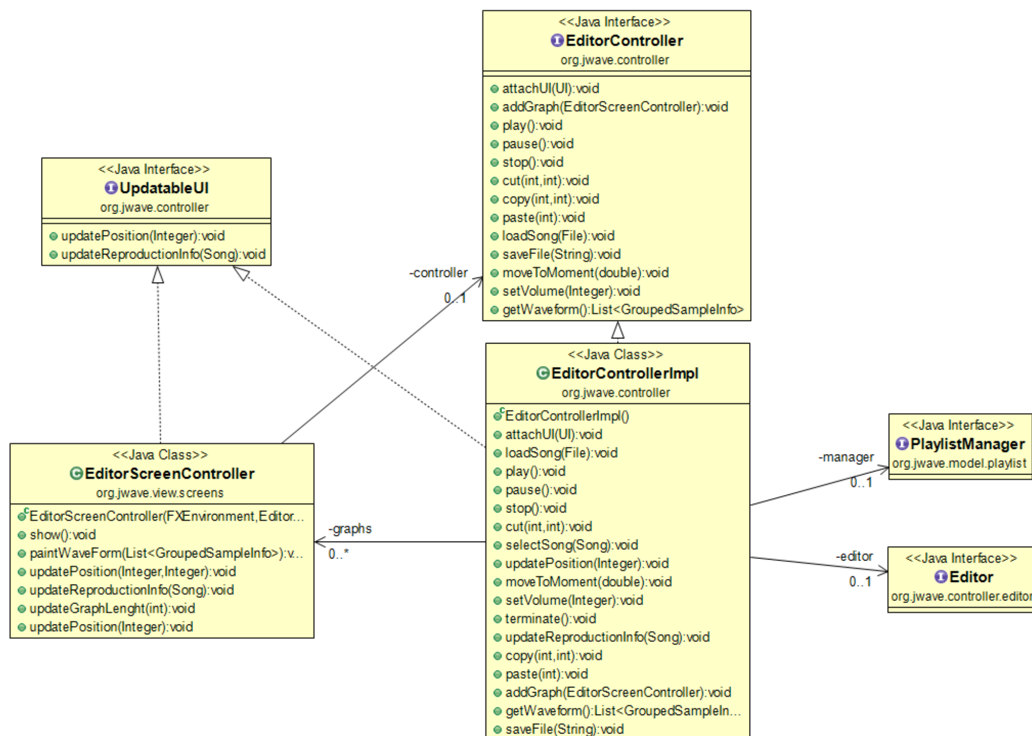

Figura 2.9: Lo schema speculare, per quanto riguarda invece la parte di editing. La struttura generale rimane la stessa, cambiano tutte le funzionalità offerte dall'entità di controllo.

Come già detto in precedenza, uno degli obiettivi principali dell'architettura MVC da noi adottata è quello di rendere totalmente indipendenti le 3 componenti principali. Compatibilmente con ciò, l'aspettativa d'uso del comparto grafico rimane consistente, anche in funzione dei tipi di dato in gioco.

Per coniugare queste esigenze è stato scelto l'impiego della libreria JavaFX, per la parte di View, che è in grado di fornire un'ampia gamma di strumenti di buon livello ad accesso relativamente facile, per gestire una situazione che in certi casi può rivelarsi non banale, senza dimenticare poi l'aspetto didattico di poter conoscere nuovi strumenti.

A fronte di una tale complessità e varietà di strumenti, e anche per via di fattori dovuti alla struttura intrinseca della libreria, si è deciso di adottare un pattern facade per mantenere dovutamente separati la parte di controller e di view e isolare totalmente la complessità della libreria in modo da permettere al controller di trattarla alla stregua di un'interfaccia testuale per quei, seppur pochi, punti in cui si interfaccia direttamente verso di lei.

Per rendere possibile ciò però è stato necessario spostare in un apparato dedicato la logica di caricamento e di runtime tipica di JavaFX, le cui applicazioni sono concepite per essere una vera e propria estensione del controller, se non uno vero e prorpio, più che mere interfacce grafiche.

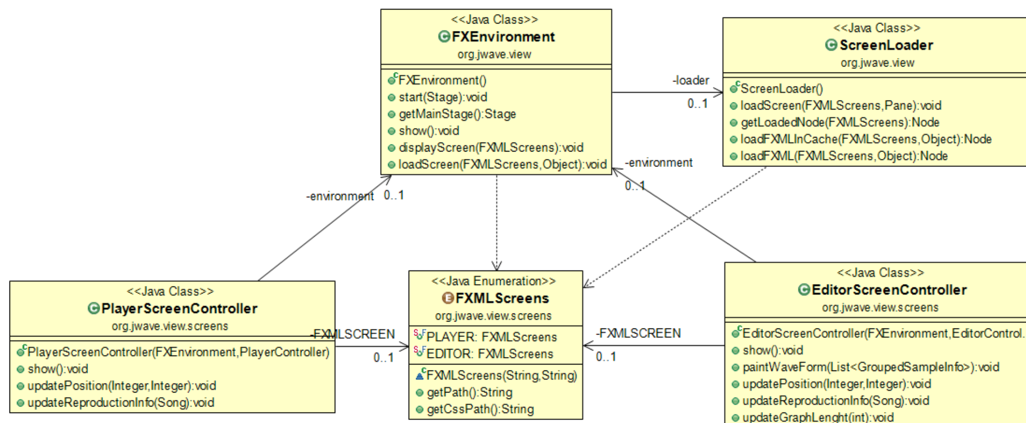

Figura 2.10: È stato creato un vero e proprio ambiente in cui la routine JavaFX possa eseguire all'oscuro del controller, il quale si interfaccia solo con le classi che controllano direttamente ciò che viene renderizzato dall'ambiente. La classe FXEnvironment è un facade che realizza la vera e propia astrazione della libreria JavaFX rispetto a un utilizzatore. Essa si occupa di renderizzare tutti i componenti richiesti (una volta caricati in memoria) e collegarne il funzionamento con le classi con cui avrà a che fare il controller. Per fare ciò in maniera più strutturata si appoggia ad un altro componente che isola e definisce le policy di gestione dei componenti grafici e la strategia di caricamento. Tale entità, ScreenLoader, si occupa di recuperare dal file system, interpretare, istanziare e memorizzare i files .fxml tipici di FX, contenenti le informazioni. Questo strategy definisce quindi una sua politica interna di gestione degli .fxml i quali, una volta recuperati vengono istanziati e uniti al codice della classe del loro rispettivo controller, per poi essere raccolti in una cache che permette transizioni immediate o anche l'utilizzo in conemporanea di diverse schermate

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Testing editor

Per verificare il corretto funzionamento del componente editor sono stati utilizzati sia test automatizzati che test manuali. I test automatizzati sono stati scritti utilizzando il framework JUnit per testare le azioni di copia/incolla e del taglio, contenuti all'interno del modello. Queste azioni hanno un comportamento ben definito, matematico e facilmente esprimibile per iscritto, che non tiene conto del contenuto di una traccia, ma soltanto della sua durata. Questo permette di scrivere con grande facilità test per diverse situazioni di utilizzo. Inoltre la natura molto precisa di queste azioni (non si possono perdere millisecondi di traccia senza motivo) e la presenza di alcuni edge-case da trattare in modo particolare, fanno sì che i test automatizzati permettano di rilevare molto facilmente eventuali regressioni nelle funzionalità che si possono controllare ad ogni nuova aggiunta di codice in modo molto veloce ed efficace con un approccio red-green-refactor.

Per quanto riguarda invece la corretta generazione di waveform, è stato usato un approcio di testing manuale, poiché i valori generati variano da canzone a canzone e dei valori con cui confrontarsi si possono ricavare soltanto da altri programmi simili (i.e. Audacity) che comunque non offrono un modo utile per esportare i dati necessari per effettuare dei test automatizzati. La tecnica utilizzata in questo caso è quindi stata utilizzare l'output dei metodi per la generazione di waveform per creare dei grafici tempo-ampiezza d'onda che poi venivano confrontati con quelli generati per le medesime canzoni da altri programmi. Essendo un componente che serve all'utente come guida visiva, fedele ma non di una elevatissima precisione, ho ritenuto che questo livello di testing sia stato sufficiente.

Testing player

Anche per testare il player si è fatto uso di test automatizzati appoggiandosi al framework JUnit. Le funzionalità testate riguardano interrogazioni sul corretto stato del player a fronte di caricamenti di brani e del susseguirsi di cambi di stato repentini che simulano quello che l'utente può eseguire attraverso i click dei bottoni dell'interfaccia grafica. Per quanto riguarda il playlist manager sono state testate le operazioni più comuni quali la creazione/rimozione di playlist e l'aggiunta di brani a queste ultime, nonchè si è verificato il corretto settaggio di una playlist come coda di riproduzione corrente.

Test manuale

Nel corso dello sviluppo si è reso necessario testare il corretto funzionamento dell'interfaccia grafica manualmente, dal momento che questa necessita la visione dell'utente per verificare la corretta rappresentazione dei componenti e non è possibile automatizzare il processo. Inoltre si presentavano problemi di rappresentazione e comportamento system dependent che sono stati poi risolti.

3.2 Metodologia di lavoro

La divisione del lavoro è stata la seguente:

- Dario Cantarelli: Progettazione e sviluppo delle funzionalità del player, di cui il componente che riproduce i file audio e il componente che organizza le varie tracce in playlist. Progettazione della utility class del controller che si occupa di salvare le playlist nel file system e realizzazione dell'agente addetto alla riproduzione automatica dei brani.

- Alessandro Martignano: la progettazione e lo sviluppo di tutta l'interfaccia grafica (GUI) e del controller che interagisce con il player e l'editor.

- Aleksejs Vlasovs: Progettazione e lo sviluppo delle funzionalità del editor: modifica di tracce, riproduzione di tracce modificate e esportazione di tracce modificate.

- Parti in comune: progettazione dell'architettura dell'applicazione. Convergenza di alcune funzionalità di competenza personale nella classe

ClockAgent, il cui sviluppo è stato supportato da tutti i membri del gruppo.

L'uso di un DVCS (tramite mercurial e un repository hostato su Bitbucket) ha permesso ad ognuno di noi di sviluppare in maniera agile ed autonoma i propri componenti del software, verificando facilemente quali modifiche sono state effettuate nella base di codice e confrontandoci tra di noi quando necessario tramite la repository condivisa. I lavori di progettazione sono stati svolti sia attraverso incontri di persona che tramite chat, ed è stata utilizzata una chat di gruppo durante tutto il periodo di sviluppo per confrontarsi e discutere delle varie problematiche che si incontravano.

Riteniamo questa divisione dei lavori congeniale al progetto. La forte separazione logica delle tre aree descritte ha permesso a ciascun membro del gruppo di concentrarsi sulla propria parte, e una volta definite le interfacce abbiamo potuto lavorare in quasi totale autonomia. Talvolta è stato necessario modificare alcune parti in relazione a delle nuove esigenze non identificate in fase di analisi, dovendo ricorrere così ad un approccio di progettazione a spirale.

3.3 Note di sviluppo

Minim

Per riprodurre tracce musicali (in formato WAV e MP3) è stata usata una libreria Java per file audio chiamata Minim [1], che permette di riprodurre e analizzare file audio. In particolare la libreria Minim è stata utilizzata per riprodurre le tracce audio caricate dall'utente, per generare la waveform delle tracce caricate nell'editor, per riprodurre le tracce modificate dall'utente tramite l'editor e infine per esportare le tracce modificate salvandole sul computer dell'utente. I Javadocs di Minim sono stati molto utili ai fini dell'implementazione e sono stati usati ampiamente.

mp3agic

Per elaborare i metadati di un file audio si è deciso di affidarsi alla libreria [6], che permette di leggere e scrivere i metadati dei file in formato mp3.

Algoritmo Export/Generate Waveform canzoni modificate

Per analizzare un file audio mentre il file non è in riproduzione, bisogna iterare i dati che rappresentano i sample che compongono i canali audio della traccia in modo programmatico. La logica per farlo non è banale, però la repository ufficiale di Minim su Github fornisce degli esempi di come fare [2]. Questo algoritmo viene utilizzato dall'implementazione dell'editor per produrre una waveform che rappresenta la traccia caricata. Viene successivamente usato in combinazione ad un altro esempio presente nella repository ufficiale che mostra come salvare un file caricato in memoria in un file in memoria permanente [3].

JavaFX

Per apprendere il corretto funzionamento e utilizzo della libreria è stato praticato uno studio intensivo dei tutorial messi a disposizione da [5] e anche dei corrispondenti di [4] per quanto riguarda componenti più recenti e customizzazioni particolari.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

team

Nonostante la complessità di sviluppo di alcune funzionalità e la necessità di adottare alcune scelte non proprio ottimali dettate dal tempo a disposizione, ci riteniamo complessivamente soddisfatti del lavoro svolto. Il software prodotto si presenta secondo le nostre aspettative, pronto ad essere modificato in futuro per aggiungere alcune funzionalità previste nel model a livello grafico e per ampliare quest'ultimo. Alcuni esempi di feature che potranno essere aggiunte sono la possibilità di settare i metadati dei brani a livello grafico e la possibilità di visualizzare più campi di questi ultimi, come le album art, immagini presenti sulle copertine degli album di cui le canzoni fanno parte.

Dario Cantarelli

Per quanto riguarda il mio lavoro di sviluppo della parte player devo ammettere che nonostante il consistente lasso di tempo dedicatogli la parte di progettazione e analisi dei requisiti avrebbe dovuto essere ancora più approfondita. La prima versione del modello non corrispondeva alle aspettative di chi si occupava della view. Ci si è trovati dunque a realizzare nuovamente un'analisi dei requisiti per fare in modo che non ci fossero ambiguità nella definizione delle entità di modello, per poi procedere in maniera più snella. Nel complesso però considerando che è il mio primo lavoro svolto in team, mi ritengo complesivamente soddisfatto: penso che il modello che ho realizzato all'interno del monte ore a disposizione possa essere agevolmente ampliato e modificato in futuro.

Alex Vlasov

Per quanto riguarda le funzionalità principali dell'editore, in particolare il modello della canzone modificabile, ritengo che la soluzione progettata e implentata sia elegante e funzionale. Un errore indubbiamente commesso però è quello di non considerare la performance in fase di progettazione. Al momento tutte le funzionalità legate all'editor funzionano a "tappeto" e vengono ricacolati molti valori inutilmente. Una soluzione duale di caching e algoritmi più intelligenti avrebbe permesso di avere più fluidità nell'utilizzo dell'editor. Questa è sicuramente la feature che vorrei aggiungere attraverso futuri lavori, oltre alla possibilità di apportare modifiche comportamentali a pari livello di quelle strutturali.

Alessandro Martignano

Lo studio sul particolare funzionamento dell'ambiente JavaFX mi ha permesso non solo di apprendere cose molto utili in ambito progettuale, ma anche di creare strumenti che sicuramente un domani potrò riutilizzare per altri scopi data, a mio avviso, la loro buona qualità di progettazione e facile estensibilità. Per quanto riguarda invece il grado di estensibilità degli altri componenti quali le grafiche vere e proprie e il controller purtroppo non c'è stato il tempo di organizzare qualcosa di più strutturato e estendibile poichè mi rendo conto che lo studio e il lavoro sul back-end grafico hanno portato via gran parte del mio monte ore. Complessivamente però mi ritengo soddisfatto di essere entrato in possesso di strumenti a me precedentemente ignoti e aver appreso il significato del lavoro in team.

Appendice A

Guida utente

A.1 Player mode tutorial

Figura A.1: Schermata principale del player.

La schermata principale del player mostra sulla sinistra una vista di tutte le playlist disponibili, sulla destra i brani contenuti nella playlist selezionata con le eventuali informazioni contenute nei metadati se presenti. In basso nella barra degli strumenti a partire da sinistra è disponibile il menù di selezione della play mode e i più comuni tasti di riproduzione, compreso uno slider per regolare il volume e la barra di scorrimento del brano.

- Per aggiungere un brano a "Tutti i brani": File - Open - Selezionare il brano utilizzando l'esplora risorse che viene aperto - Premere Ok.
- Per creare una Playlist: Cliccare su "Nuova playlist" - Immettere un nome per la nuova playlist - OK
- Per aggiungere un brano già presente in una playlist: tasto dx sul brano desiderato - Aggiungi a Playlist - Selezionare dal menù a tendina la playlist desiderata - Premere OK.
- I brani "Mistery" e "Snow Time" contenuti nella directory songs sono prodotti da Dario Cantarelli. Il brano "Dream of Love" è stato prodotto da Dario Cantarelli e Marco Dall'Ara.

A.2 Editor mode tutorial

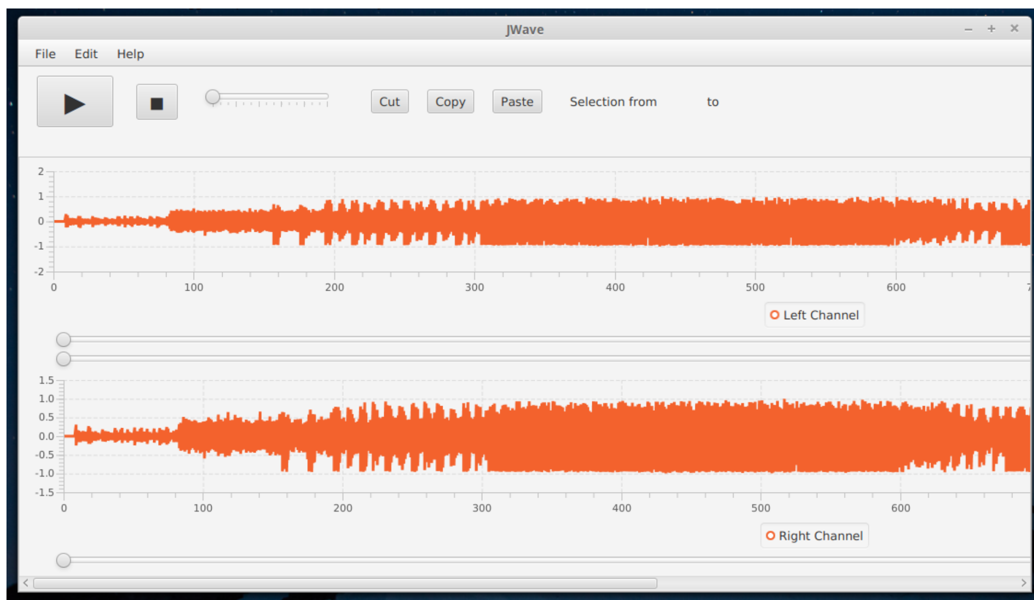

Figura A.2: Schermata principale dell'editor mode.

La modalità editor permette di caricare un brano e di visualizzare il grafico tempo/ampiezza dei suoi due canali, sinistro e destro. Lo slider in fondo ha la stessa funzione di quello presente nella player mode. I due slider al centro della schermata sono i cursori, che permettono di selezionare una porzione di brano per svolgere una delle operazioni rappresentate dai bottoni in alto:

- Cut: permette di tagliare la porzione di brano selezionata.
- Copy: permette di copiare la porzione di brano selezionata.
- Paste: permette di incollare la porzione di brano selezionata.

Bibliografia

[1] D. Di Fede. Minim library.

[2] D. Di Fede. Minim offline analysis.

[3] D. Di Fede. Minim save.

[4] M. Jakob. Codemakery fx tutorials.

[5] Oracle. Oracle fx examples.

[6] M. Patricios. mp3agic.