Relazione per “Beesound Player”

Carlo Alberto Scola
Gianluca Cincinelli
Tiziano De Cristofaro

1 Febbraio 2016

Indice

Capitolo 1

Analisi

1.1 Requisiti

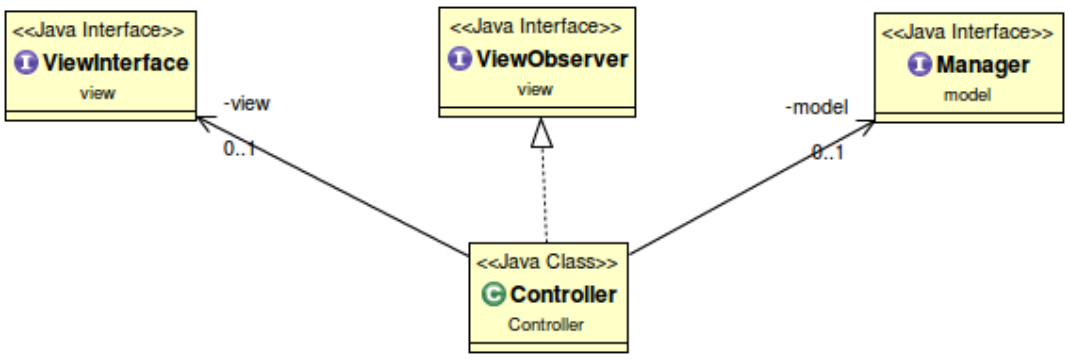
Il software è un lettore di file audio in formato mp3 che dovrà avere le seguenti caratteristiche:

- Gestione dei controlli di base per la riproduzione di un file audio: play, stop, pausa, next e previous track, riproduzione lineare o shuffle, controllo volume, barra di avanzamento.
- Gestione di una libreria di file musicali con possibilità di aggiungere o rimuovere brani e organizzarli in playlist.
- Visualizzazione delle informazioni principali relative a ognuna delle canzoni.
- Visualizzazione informazioni principali sullo stato della libreria: numero di brani, durata complessiva, elenco totale delle canzoni, degli album, degli artisti, delle playlist dell'utente, dei generi e dei brani più ascoltati.
- Barra di ricerca per trovare brani all'interno della libreria.
- Gestione di una coda di riproduzione con possibilità di aggiunta ed eliminazione.

1.2 Analisi e modello del dominio

Il software è organizzato in modo che i file mp3 vengano inseriti all'interno di una libreria creata nel file system. Una volta salvati i dati un gestore dell'archivio li tiene organizzati e li fornisce quando richiesto, in modo che siano disponibili all'utente nei diversi contesti di utilizzo. Il software si occupa quindi di recuperare i dati dal file system, di gestire la riproduzione audio e di visualizzare tutte le informazioni necessarie, fornendo un'interfaccia con l'utente.
La difficoltà principale sarà quella di dividere i compiti tra le parti funzionali in modo da non incidere sulla reattività del software, in particolare questo potrebbe accadere in relazione ai componenti visivi in continuo aggiornamento durante l'utilizzo.

Di seguito un diagramma UML che mostra le principali parti funzionali:

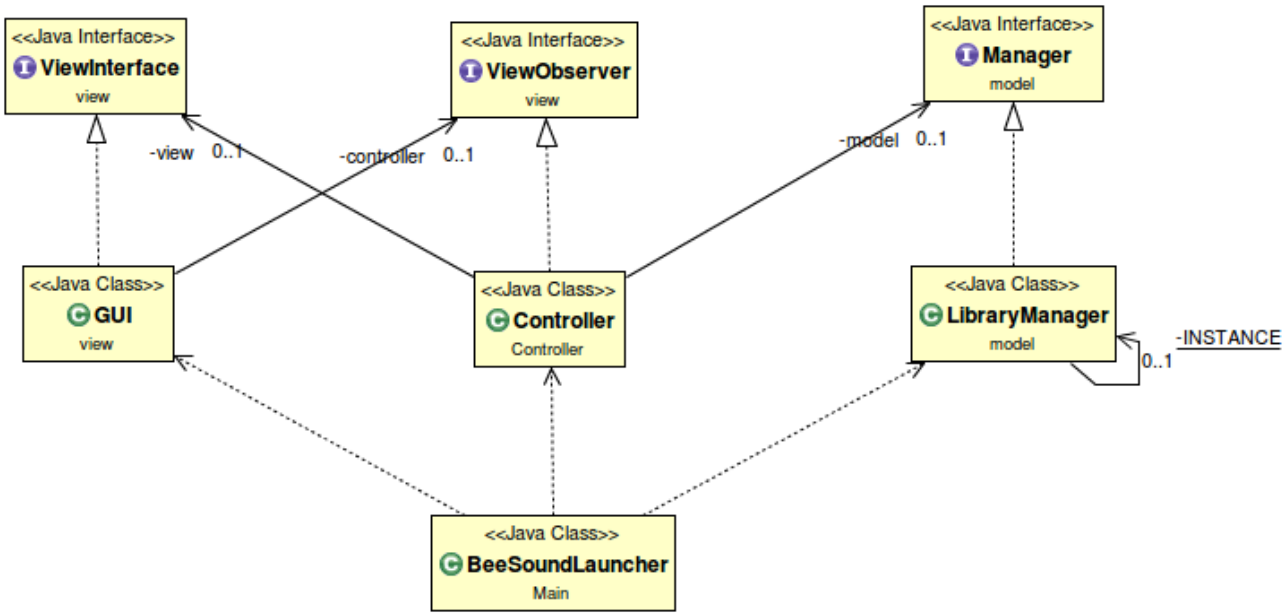


Capitolo 2

• Design

2.1 Architettura

Il software è stato organizzato seguendo il pattern MVC. In particolare è gestito in modo che il model metta a disposizione le proprie funzionalità senza mai chiamare quelle delle altre due parti. Model e view non sono mai a diretto contatto, le richieste della view al model quindi passano sempre attraverso il controller. Le parti sono messe in contatto tra di loro tramite apposite interfacce, che permettono quindi di gestire il progetto in modo modulare, è possibile sostituire in blocco ognuno dei componenti con altri, purché sia garantita l'adesione alle corrette interfacce.



UML architetturale

2.2 Design dettagliato

Tutte le funzionalità che il model fornisce al controller (e quindi indirettamente alla view) sono presenti all'interno dell'interfaccia Manager. Questa interfaccia rappresenta il contratto a cui aderisce il gestore della libreria musicale, e contiene metodi dedicati allo storage delle informazioni e al loro reperimento in vari formati, in modo da adeguarsi a richieste di diverso tipo. L'interfaccia ViewObserver stabilisce invece quali sono i metodi che il controller fornisce alla view, quelli che invece possono essere richiesti alla view da parte del controller sono contenuti nell'interfaccia ViewInterface.

• **Model** (Tiziano De Cristofaro):
Nel Model vengono gestite tutte le funzionalità di archiviazione dati e di reperimento degli stessi per fornirli (direttamente o meno) agli altri componenti del software. Ogni elemento della libreria musicale è rappresentato da una classe che può contenerne altre (ad

esempio un Album contiene delle Song). La totalità degli elementi presenti in libreria è raccolta nel gestore mediante delle liste, è possibile quindi ottenere un determinato elemento in più modi in base alle esigenze, poiché esso sarà salvato sia nelle liste del gestore che in quelle dei singoli elementi. L'interfaccia Manager (implementata dal gestore) permette un'eventuale sostituzione futura del gestore di libreria senza che altre parti del codice debbano essere ricondizionate.

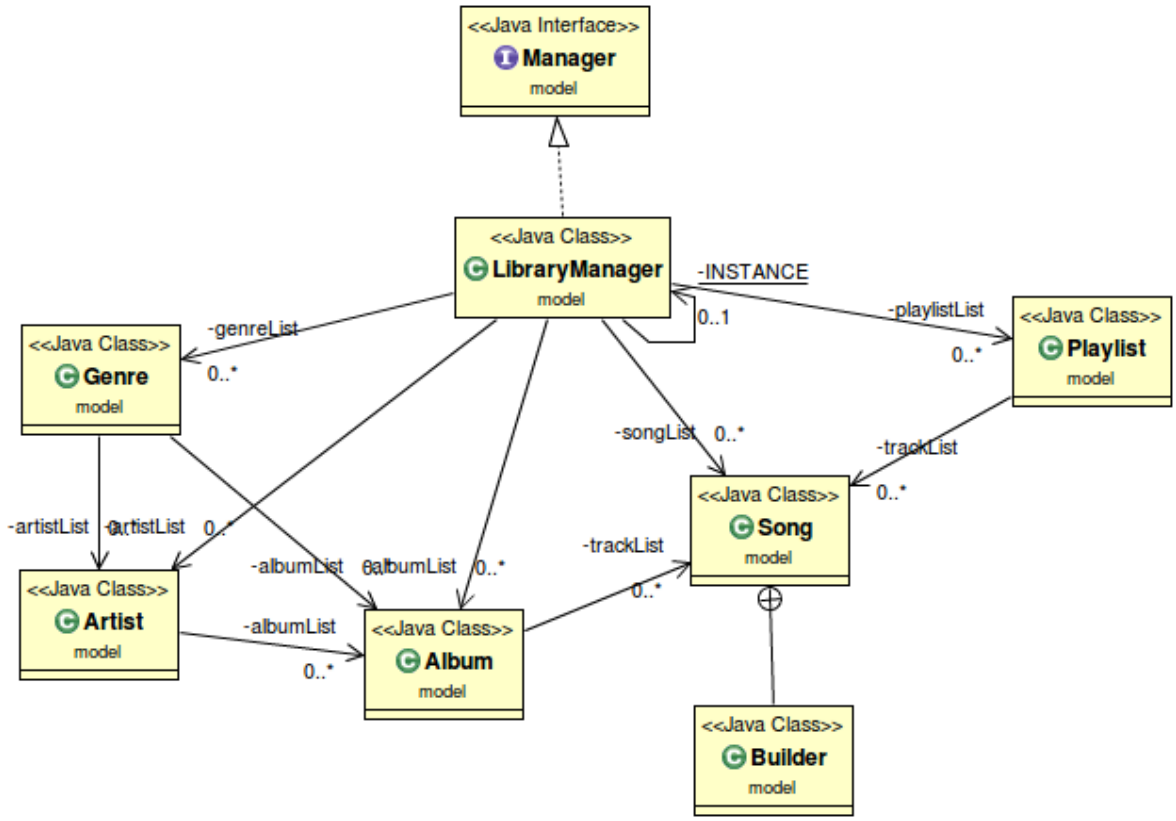
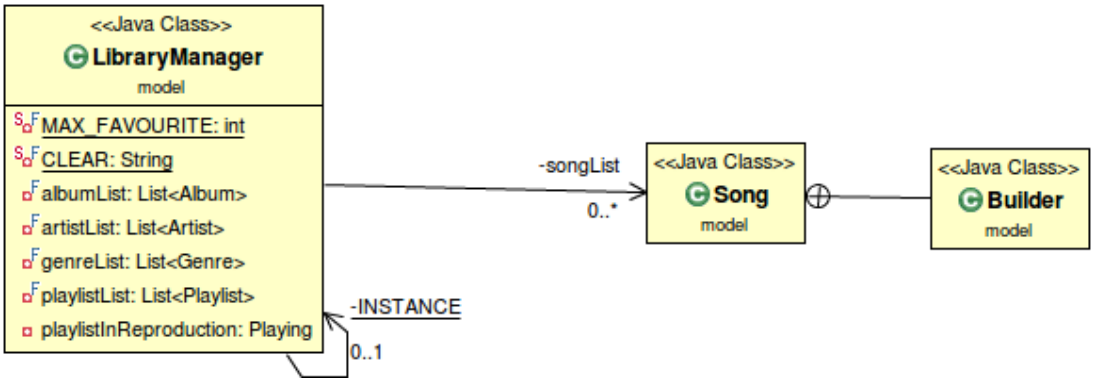


Diagramma UML relativo ai collegamenti tra le entità del model.

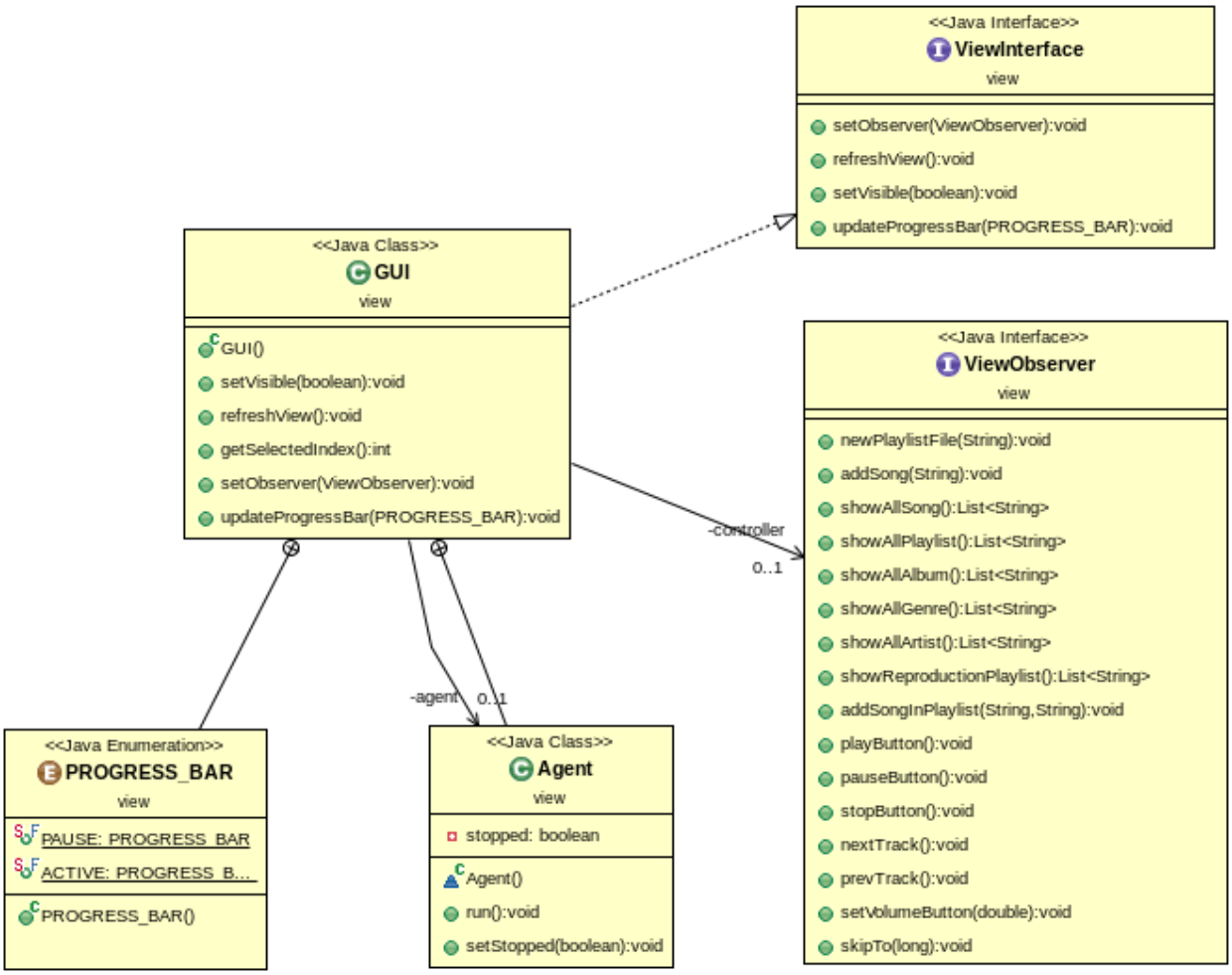
Nel model è stato impiegato un pattern Singleton per la creazione del gestore (LibraryManager) in quanto durante l'esecuzione del programma dovrà essere presente una sola istanza alla volta del gestore di libreria. Per la creazione delle Song è stato impiegato un pattern Builder, realizzato mediante una inner class, la classe Song infatti contiene numerosi campi e l'utilizzo del builder permette un miglior livello di chiarezza riguardo le informazioni inserite.



UML relativo al pattern Builder. E' incluso anche lo schema del Singleton.

• **View** (Gianluca Cincinelli):

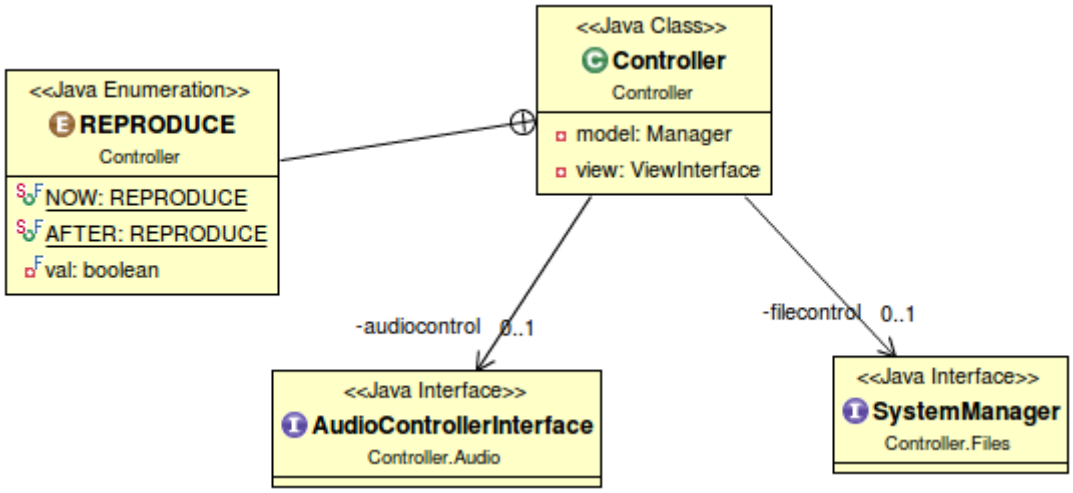
La GUI è stata sviluppata in modo da avere i soli compiti di aggiornare la visualizzazione di volta in volta richiesta dall'utente e di notificare il controller della messa in esecuzione dei brani. La reattività è stata l'obbiettivo principale. Anche se la classe è obiettivamente disorganizzata, è stata lasciata fuori per quanto possibile la gestione dei dati e la loro manipolazione, affidata per la quasi totalità al model. L'accesso ai dati è garantito dai metodi messi a disposizione della GUI dall'interfaccia ViewObserver, implementata dal controller. L'interfaccia ViewInterface è composta dei metodi che il controller ha facoltà di chiamare sulla GUI, in fase di inizializzazione e quando è necessario un aggiornamento. A causa di difficoltà nella fase iniziale di progettazione, non sono riuscito ad utilizzare pattern che potevano strutturare il codice in modo migliore per leggibilità e performance. Si noti che la funzionalità della barra di scorrimento della canzone(seekbar) è stata sviluppata dal collega di progetto Carlo Alberto Scola. Nel rispetto del pattern MVC è stato fatto si che Model e GUI non vengano mai a contatto diretto, ma sempre tramite il controller.



UML relativo alla view.

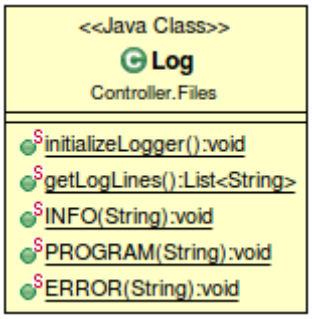
• **Controller** (Carlo Alberto Scola):

Il Controller ha appunto come compito quello di gestire ogni tipo di comunicazione fra model e view e in quanto ciò molti metodi implementati dalla parte Observer, altro non fanno che traslare al model le loro richieste. Parte molto importante nello svolgimento e nella progettazione è stata quella di mantenere il più possibile un incapsulamento forte. Ogni classe, all'interno non presenta nessun import di una classe del package diverso dal suo. Facendo ciò ci si assicura in maniera molto certa la futura interoperabilità e interscambiabilità dei componenti nel caso in cui in futuro si voglia completamente rimpiazzare una view con un altra, o un model. Questo poiché ogni classe fondamentale altro non è che un implementazione della sua interfaccia. (Grazie a questo è garantita anche una interscambiabilità dentro al controller stesso di classi singole)
All'interno il Controller suddivide alcune sue specifiche in altri due package.



Uno gestisce l'audio, e dunque qui ci si appoggia ad una libreria BasicPlayer che agevola mettendo a disposizione dei metodi per la lettura dei brani musicali, sebbene con qualche piccolo inghippo che spiegherò piu avanti. Oltre alla lettura ci si è muniti di un sistema per leggere i metadati (se presenti). Questo grazie ad una classe (presa dal suo produttore quindi non di mia fattura fondamentalmente) che utilizza una libreria per la lettura metadati Tritonus.
L'enum (puro motivo estetico e di immediata chiarezza) è per la riproduzione di un brano subito (al click) o successivamente (aggiunta in queue). Gestita come Strategia per la riproduzione dei brani è anche la enum ReproductionStrategy, la quale indica se i brani da riprodurre andranno riprodotti in maniera Shuffle o Linear.
Un'altra nota: Per garantire un corretto funzionamento nella GUI della seekbar ho aggiunto metodi get/set di un particolare campo che indica in tempo reale in quale istante della canzone ci si trovi, e lo manda alla seekbar in display, viceversa la seekbar manda quel istante al metodo per settare il salto temporale. Siccome si ha sempre uno “scrittore” e a volte si ha bisogno di “leggere” e “modificare” lo stesso dato contemporaneamente ho gestito tramite blocchi synchronized la mutua esclusione.
L'altro package si occupa della gestione fisica di file e cartelle. Più in particolare si riferisce ad ogni aspetto che ha a che fare con il sistema operativo e dunque con una gestione più fisica di quello che sta succedendo.

Si gestiscono eccezioni, errori, creazioni, eliminazioni e quant'altro.
Ogni operazione, errore, modifica è resa pubblica in un file di testo di logging grazie alla classe Log implementata. Da qualsiasi parte del programma è possibile loggare informazioni richiamando in maniera statica Log.”priority”(String)



Si è cercato di fare un uso il più possibile semplificato, chiaro e corretto del codice per aderire ad una “buona programmazione”, mediante pattern, streams & lambda.

Ps. Vorrei sottolineare un fatto alquanto buffo e stupido per il quale inizialmente non fui fortunato nel trovare la libreria giusta per la riproduzione audio, allora mi affidai a un implementazione “scarna” presa in rete e mi cimentai nell'implementarla io stesso.. Ci impiegai molte ore e alla fine il risultato ci fu, ma con molto codice dentro al progetto.. codice superfluo che poteva essere inglobato proprio in una libreria propria. Allora cercai meglio e trovai questa libreria BasicPlayer che mi riassunse il tutto con una facilità molto maggiore.. buttando in pratica tutto il mio precedente lavoro che comunque mi servì per esercitarmi.

La suddivisione in package è stata effettuata in modo da rispecchiare il più possibile la divisione logica dello schema MVC: ad ognuna delle parti corrisponde un package, tranne per quanto riguarda il controller che è invece distribuito su tre package per garantire una maggiore specificità delle sue sottosezioni, ovvero il controller che è a contatto con le altre parti, il controller audio che fornisce la gestione vera e propria dell'audio e la parte che si occupa dell'organizzazione a livello di file system.

Capitolo 3

• Sviluppo

3.1 Testing automatizzato
E' stata realizzata una classe per il testing automatizzato tramite la suite Junit. Sono state testate le funzionalità di aggiunta brani alla libreria (tramite lunghezza della lista e titoli inseriti) , aggiunta album (derivante dall'aggiunta dei brani), aggiunta di brani in lista di riproduzione, aggiunta di playlist e di brani nelle playlist e funzionamento della barra di ricerca.

3.2 Metodologia di lavoro
La suddivisione del lavoro rispecchia essenzialmente quella proposta nella sezione del design di dettaglio: Carlo Alberto Scola (controller), Gianluca Cincinelli (view), Tiziano De Cristofaro (model). Questa divisione è stata rispettata fedelmente, anche se sulla view da un certo punto in avanti ci sono stati degli interventi da parte dei due membri non preposti ad essa. Degli interventi minimi sono stati effettuati sul controller da parte del gestore della view, in modo da ottenere un accesso più veloce alle chiamate sul model. Ad ogni modo la suddivisione iniziale del lavoro ha permesso di dedicarci in maniera separata alle tre sottoparti del pattern MVC seguendo quindi tre linee di sviluppo diverse, il punto di contatto tra le tre parti è rappresentato dalle interfacce, che contengono i metodi che ognuna delle parti può chiamare sulle altre. Al momento dei primi collegamenti tra model e controller i metodi non previsti in fase di design che si rendevano man mano necessari venivano aggiunti all'interfaccia Manager e quindi implementati.
Il DVCS è stato utilizzato inizialmente lavorando su tre branch differenti, in quanto la prima fase è stata quella della scrittura individuale della propria parte, in seguito è stato effettuato un primo merge tra i due branch relativi a model e controller, in modo che il controller potesse chiamare concretamente i metodi del model. Successivamente a questa linea di sviluppo è stata aggiunta quella relativa alla view, in modo da avere visibilità su tutti i componenti e un maggior dialogo tra le parti. Nelle ultime fasi c'è stata una divisione in cui da una parte è stato curato lo sviluppo della barra di avanzamento (Carlo Alberto Scola) e dall'altra si sono aggiunte funzionalità alla view aggiungendole al model e inserendo le chiamate nel controller (Gianluca Cincinelli e Tiziano De Cristofaro).

3.3 Note di sviluppo
Un punto di complessità più elevata durante lo sviluppo del software è stato sicuramente quello relativo alla gestione della barra di avanzamento, in quanto la sua gestione è affidata ad un thread separato e la scrittura del codice è stata effettuata in modo che la GUI non risultasse poco reattiva a causa del continuo aggiornamento della barra (aggiungere eventuali dettagli).
Per la riproduzione dei file mp3 e il recupero dei metadati da file system abbiamo utilizzato la libreria Tritonus e BasicPlayer(controllare compatibilità con i sistemi operativi).
L' implementazione del metodo createSelectableList all'interno della GUI è stata realizzata facendo riferimento a del codice mostrato su StackOverflow.

Capitolo 4

• Commenti finali

4.1 Autovalutazione e lavori futuri

Il lavoro in generale gode di una buona impostazione per quanto riguarda l'utilizzo di MVC, questo ha portato a una strutturazione in cui è garantita la modularità dei componenti e quindi la loro sostituibilità. Al momento non risultano bug significativi e l'interfaccia del software risulta sostanzialmente reattiva durante l'utilizzo. Dei punti di debolezza potrebbero essere l'organizzazione strutturale della view e la presenza non molto numerosa di pattern individuati. Di seguito l'autovalutazione di ognuno dei membri del team:

• Model (Tiziano De Cristofaro):

Il model è stato realizzato con un buon incapsulamento, cercando di fornire agli altri componenti solo le funzionalità realmente necessarie. In generale risulta ordinato, in quanto si è cercata una corrispondenza tra elementi concettuali e classi utilizzate, per cui ogni classe ha una logica ben determinata. Probabilmente l'utilizzo di interfacce aggiuntive permetterebbe una migliore estensibilità del software.
C'è stata collaborazione con entrambi gli altri membri del gruppo, è stato necessario infatti fornire funzionalità al controller quando richiesto, e la necessità della view di relazionarsi indirettamente con il model mi ha portato a partecipare allo sviuppo di alcune funzionalità nella GUI.

• View (Gianluca Cincinelli):

La View risulta non organizzata secondo i pattern come auspicato, alcuni dei quali individuati come possibile soluzione troppo tardi per poter rifattorizzare adeguatamente il codice. Sicuramente diventando così una classe con molto codice ripetuto e di difficile lettura e gestione, quindi difficilmente modificabile in futuro o estensibile in modo chiaro. Per la buona separazione dei ruoli però una nuova GUI non dovrebbe comportare cambiamenti problematici in model e controller.

• Controller (Carlo Alberto Scola):

Il controller è stato realizzato pezzo per pezzo, testando e implementando ogni nuova funzione in maniera ordinata e essenzialmente chiara e concisa, senza ripetizioni. Si è tentato il più possibile di scrivere del “bel codice” capibile e fruibile da chiunque. Mi sono divertito a cercare soluzioni più eleganti e devo ammettere che mi è servito molto per migliorarmi sotto alcuni campi. Ho trovato molto interessante questo ruolo assegnatomi, insieme al gestire il thread per la barra di avanzamento, la quale ha dato non pochi problemi poiché non garantiva sempre un suono continuo, uniforme e pulito. Per questo la soluzione si è trovata tentando ogni possibile via, andando a capire a più basso livello il problema. (che poi si rivelò una non corretta sincronizzazione dell istante temporale

in cui la barra dovesse far partire la canzone, e dunque uno sfasamento di byte che mandato in riproduzione produceva “gracchio”).
Concludendo, il lavoro mi è servito molto per capire a più alto livello le linee guida della “buona progettazione” e spero di migliorarmi sempre più in futuro, se non con questo linguaggio di programmazione in altri, ma sempre adottando il giusto “modus operandi” acquisito durante il corso.

Appendice A

• Guida utente

Al primo avvio dell'applicazione viene creata nella home dell'utente una cartella che conterrà la libreria.
E' possibile dal menù “file” gestire la propria libreria musicale (aggiungere canzoni, creare playlist ecc…).
Sulla sinistra sono presenti i bottoni che visualizzano le varie liste di elementi della libreria, questi sono mostrati nello spazio centrale. E' possibile mandare un brano in esecuzione effettuando un doppio click o selezionandolo e premendo play. Con il tasto destro è possibile aprire un menù che fornisce opzioni relative agli elementi visualizzati.
Nella parte bassa dell'interfaccia sono presenti i comandi di base per la riproduzione.