saudio-sim Report

Alessandro Sciarrillo, Niccolò Mussoni, Alex Presepi, Simone Lugaresi

25 aprile 2022

Indice

Capitolo 1

Analisi

Il software ha l'obiettivo di permettere all'utente di sperimentare un ascolto realistico in un ambiente dinamico. Per ascolto realistico si intende l'esperienza uditiva che si può sperimentare tramite un ascolto con cuffia che trasmette all'utente un effetto di spazialità. Invece con ambiente dinamico ci si riferisce ad un sistema di cui si possono spostare i componenti.

1.1 Requisiti

Requisiti funzionali

- Permettere all'utente di ascoltare tramite le proprie cuffie una riproduzione realistica in relazione alla posizione all'interno di un ambiente personalizzabile.
- Garantire all'utente la possibilità di spostare l'ascoltatore e le sorgenti sonore. Un ascoltatore ha una posizione all'interno di un ambiente ed è colui nel quale l'utente si impersona. Le sorgenti sonore rappresentano i dispositivi da cui il suono viene riprodotto.
- Fornire un equalizzatore per poter applicare vari effetti alle sorgenti sonore.
- Cambiare il range di frequenza di un dato speaker.
- L'ascoltatore dovrà poter avere un orientamento di 360 gradi modificabile.

Requisiti non funzionali

- Sarà possibile importare file audio e dovranno essere supportate anche tracce stereo (2 canali) oltre a quelle mono (1 canale).
- Possibilità di aggiungere funzionalità visive e uditive aggiuntive per l'ascoltatore.

1.2 Analisi e modello del dominio

Saudio-sim dovrà essere in grado di scegliere una traccia audio che potrà essere modificata da un equalizzatore, posizionare le sorgenti da cui verrà riprodotta e il punto da cui verranno ascoltate. Il Listener e le Sources faranno parte di un Environment, che ne definirà lo spazio nel quale potranno essere posizionate. Inoltre le Sources dovranno riprodurre le tracce associate a dei Buffer, con uno specifico range di frequenza, apportato tramite filtri. Uno dei principali problemi è quello di simulare in modo realistico l'ascolto in relazione a:

- posizione del Listener
- dimensione dello Spazio
- posizione e tipo di Sources.

Tra le maggiori difficoltà c'è quella di applicare filtri ed effetti alle tracce.

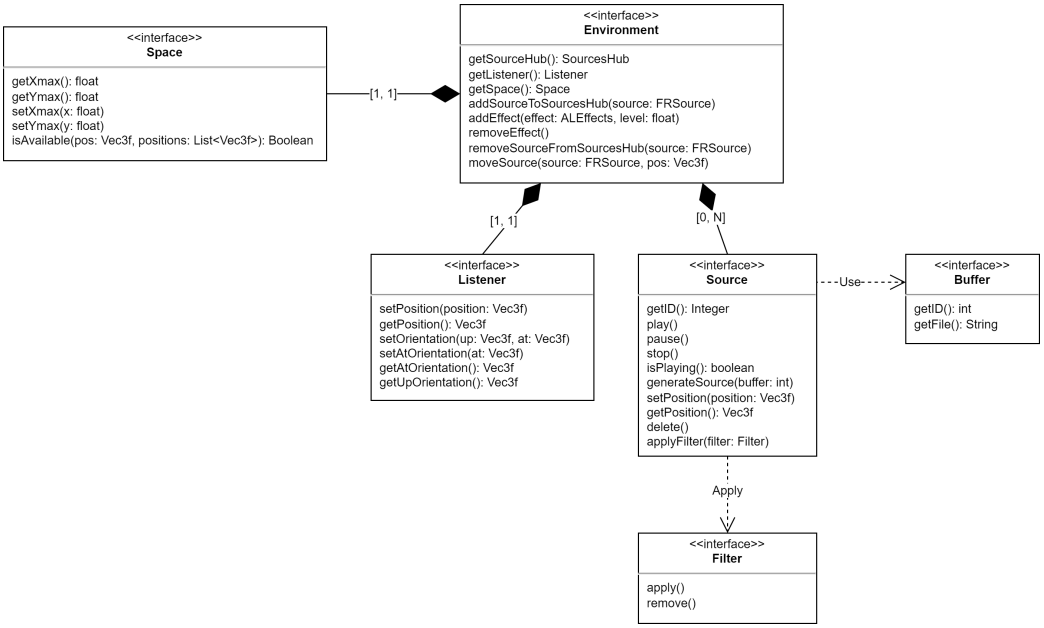

Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

MVC

Nella strutturazione del progetto abbiamo cercato di seguire una linea standard con l'implementazione di MVC, cercando di rendere ogni parte di quest'ultimo il più modulare possibile dividendo in comparti ben definiti ogni sezione. Il model ha una struttura classica, invece abbiamo dovuto aggiungere componenti che avessero la funzione di coordinare controller e view.

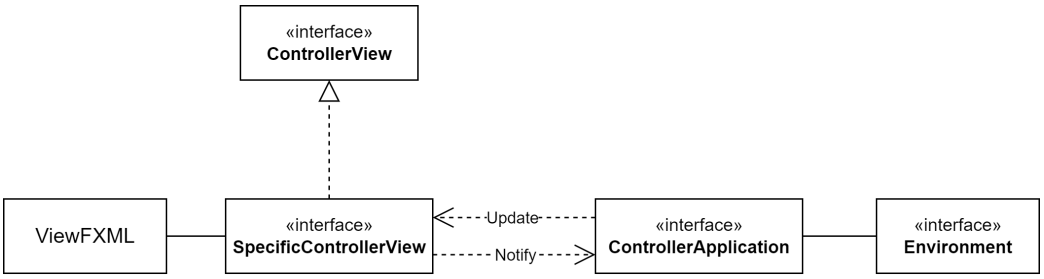

Figura 2.1: Schema UML dell'MVC

Model

L'interfaccia principale del model è offerta dall'Environment, che coordina le entità principali e ne contiene le istanze, ma ogni sua sottoparte ha comunque punti di ingresso per essere utilizzata dal proprio controller specifico, in modo da diversificare gli entry point e rendere il tutto più modulare.

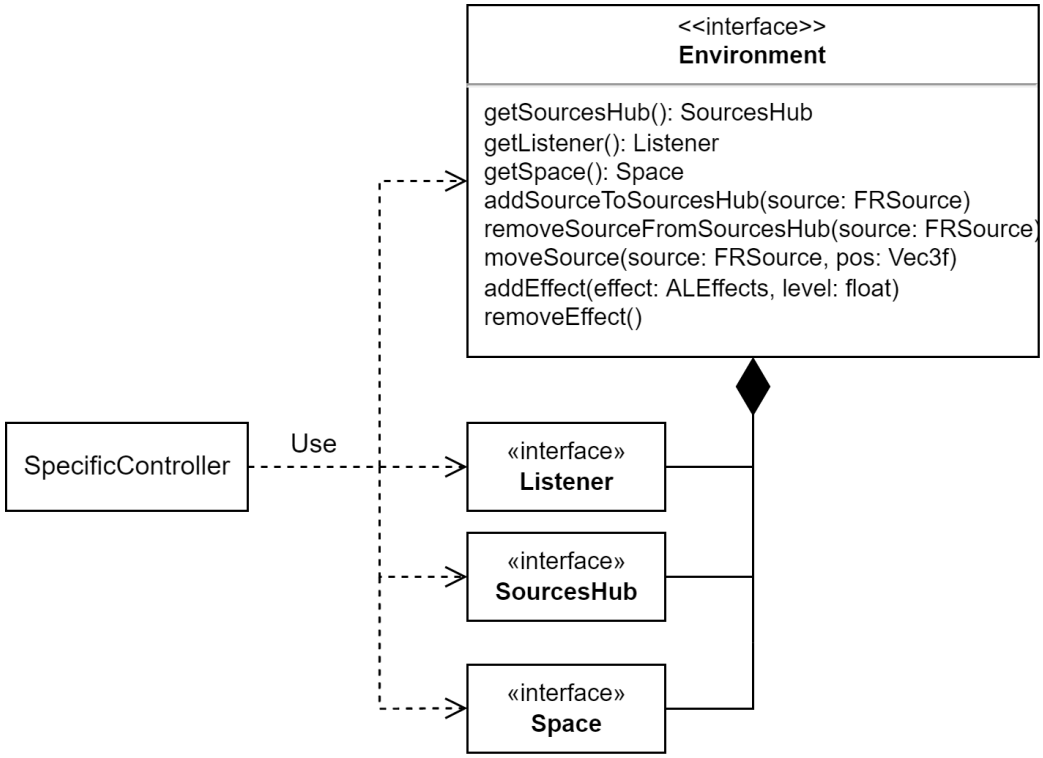

Figura 2.2: Schema UML del model

Controller

Abbiamo deciso di creare un MainController che gestisce tutti gli SpecificController in modo da rendere l'interazione tra essi più semplice. Esso mette a disposizione a ogni ControllerApplication la possibilità di accedere agli altri.
La scelta di creare i vari SpecificController è dettata dal fatto che ognuno ha bisogno di gestire in modo specifico una parte di model e la creazione di un numero più ridotto di controller sarebbe stata troppo caotica e avrebbe limitato l'estendibilità e la manutenibilità del codice.

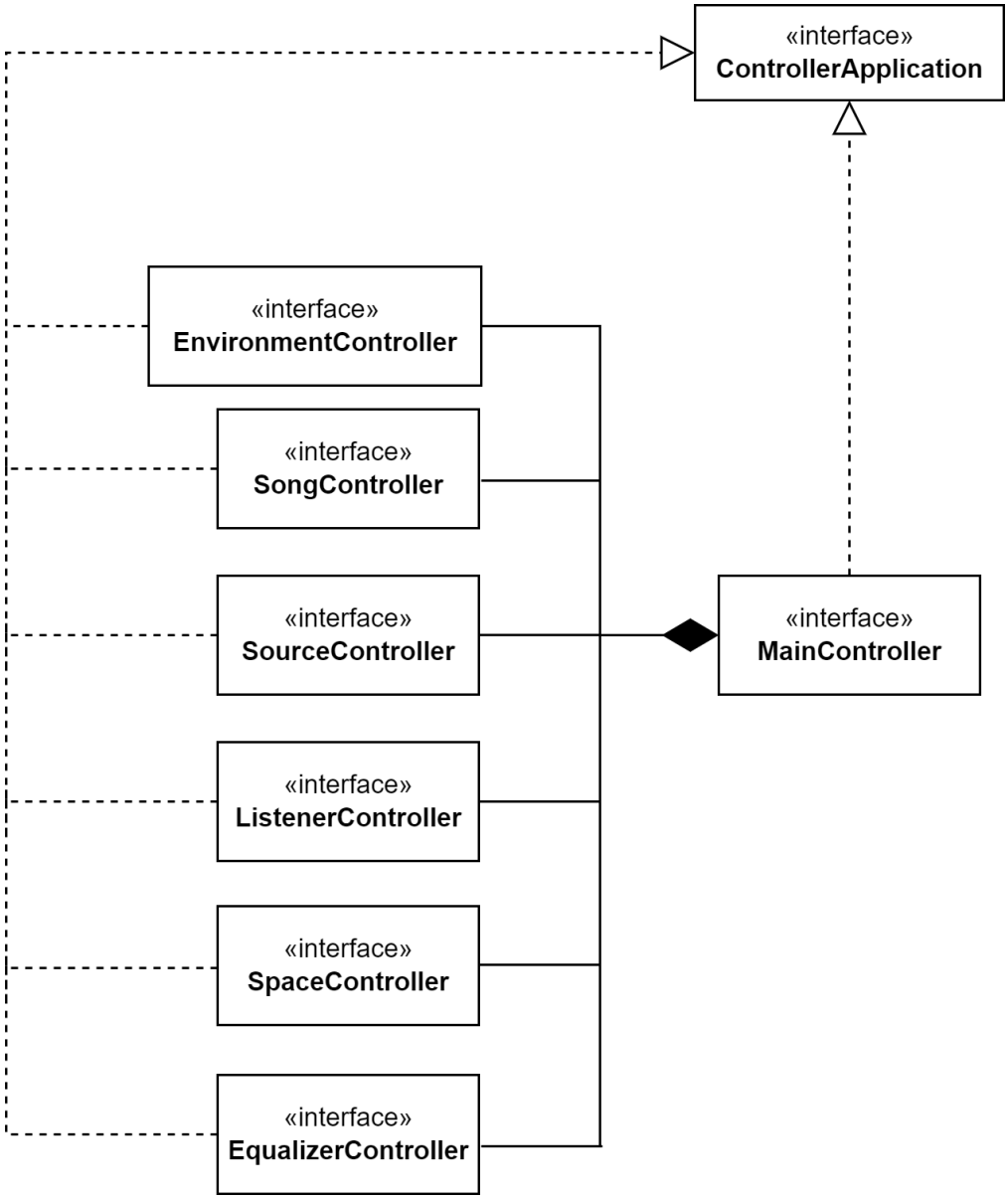

Figura 2.3: Schema UML dei controller

View

Questa parte è risultata la più critica da progettare, data la poca esperienza con la tecnologia grafica utilizzata visto che abbiamo voluto mantenere una salda divisione tra le parti anche in questa fase di sviluppo.
L'obiettivo era di traslare la modularità del model e del controller anche nella view e questo ci ha portato a definire diverse view per ogni entità principa-

le. Abbiamo introdotto dei controller all'interno della parte di View relativa all'MVC, in modo da dividere la mera parte visiva da quella di controllo, permettendo ai ControllerView di gestire la specifica tecnologia grafica e creando delle interfacce view che non facessero trasparire la tecnologia utilizzata.

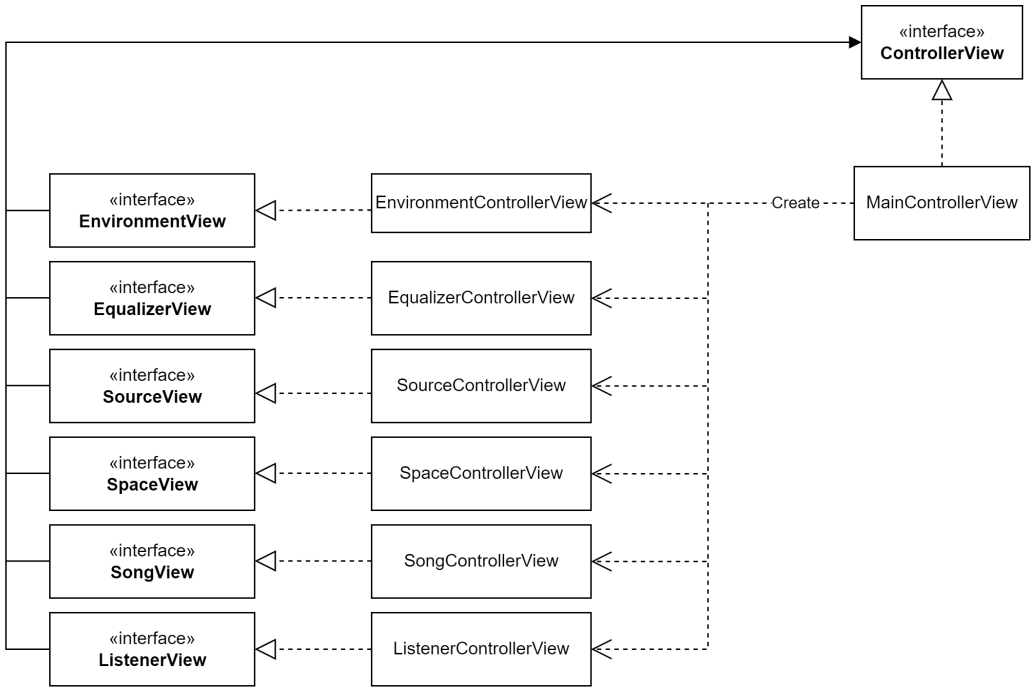

Figura 2.4: Schema UML della view

2.2 Design dettagliato

2.2.1 Niccolò Mussoni

Il mio dominio di lavoro è centrato su due argomenti:

- I Buffer, che sulla base della traccia importata generano un ID univoco associato ad esso, attraverso il quale le sorgenti possono riprodurre la traccia.
- Le estensioni, che riguardano la gestione di filtri ed effetti per le sorgenti.

Buffer

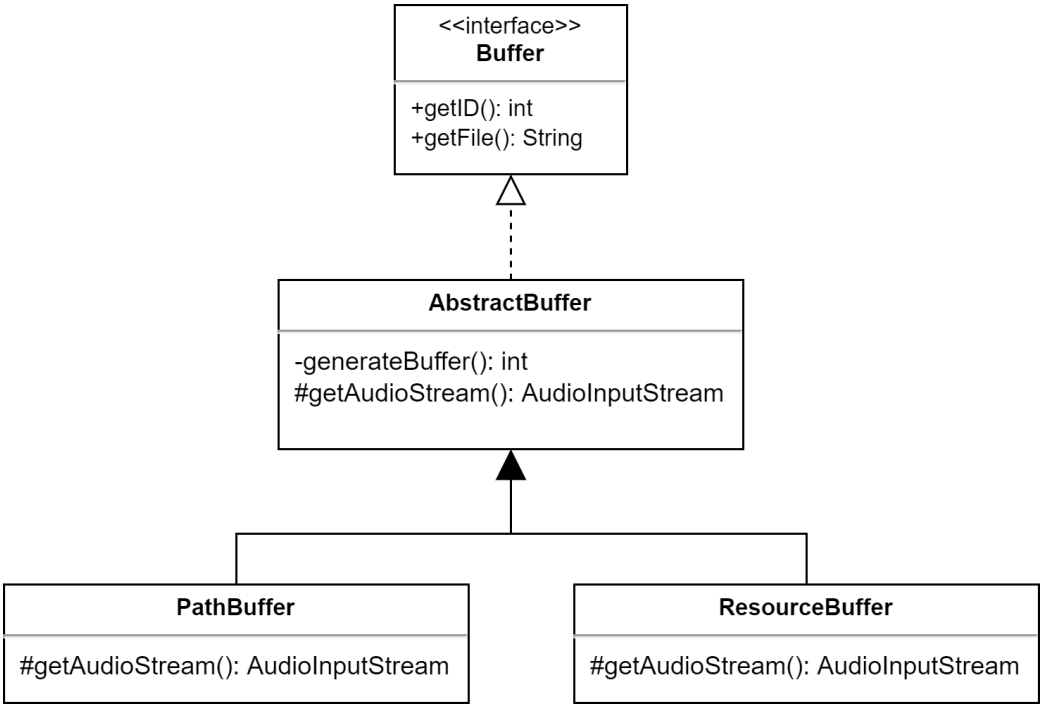

Figura 2.5: Rappresentazione UML dell'applicazione del pattern Template Method per i diversi tipi di Buffer

Il primo problema da gestire è stato il fatto che il Buffer potesse essere generato sia da un file nel file system, sia da un file incluso nelle risorse del classpath. Visto che i Buffer differiscono solo per come viene importato inizialmente lo Stream, per risolvere questo problema ho sfruttato il pattern *Template Method*, in modo da poter essere usato anche per altri tipi di Buffer non presenti nel progetto. Nel mio caso, il metodo template è generateBuffer, mentre il metodo primitivo è getAudioStream.

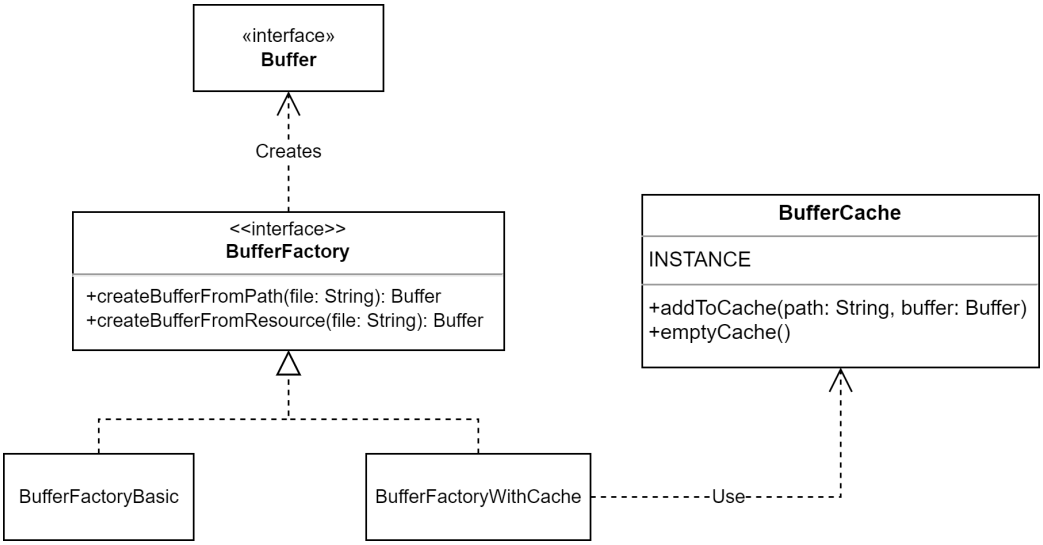

Figura 2.6: Rappresentazione UML del pattern Factory Method e del pattern Singleton per il caching

Il secondo problema da gestire era evitare Buffer duplicati e per risolvere ciò mi sono affidato a un istanza Singleton che potesse fare da cache in maniera thread-safe, grazie alla versione attraverso enum di J. Bloch. Questo mi ha risparmiato di aprire un nuovo InputStream e analizzarlo per la creazione del Buffer nel caso la path del file che volesse essere importato fosse già salvata in memoria.
Per la creazione dei diversi tipi di Buffer mi sono affidato a una factory, applicando il template *Factory Method*: l'interfaccia di partenza è BufferFactory con all'interno i diversi tipi di Buffer che è possibile creare: in particolare ho implementato BufferFactoryWithCache che, prima di creare un nuovo Buffer, controlla che non sia già presente nella cache e successivamente procede a crearne uno nuovo e BufferFactoryBasic che ritorna direttamente una nuova istanza di Buffer, ma grazie al template se ne possono creare ulteriori tipi.

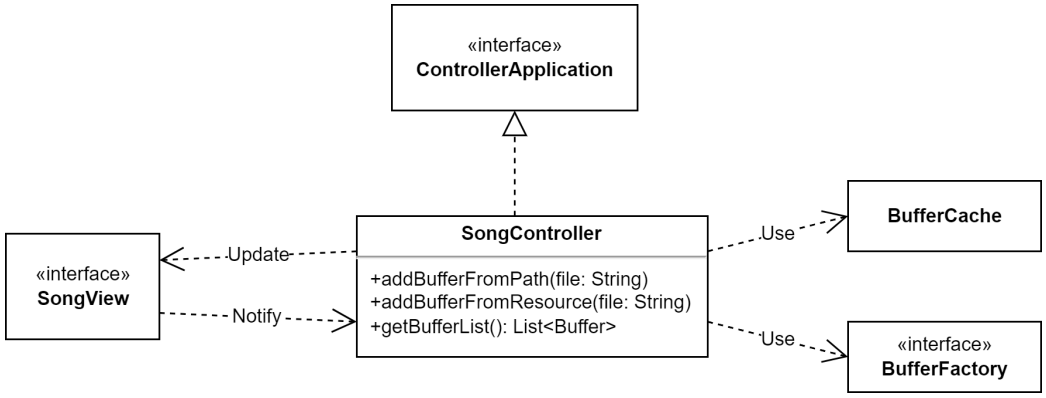

Figura 2.7: Rappresentazione UML della Factory per Buffer e dell'applicazione del pattern Singleton per il caching

Quando dalla view viene richiesta l'importazione di un file, esso viene trattato come PathBuffer, in quanto viene selezionato dall'utente. Al momento dell'inizializzazione invece tutti i file audio presenti come risorse del progetto vengono trattati come ResourceBuffer. Questo mi ha aiutato a poter differenziare anche i metodi nel controller senza ricorrere a variabili per identificare che tipo di file fosse. Il controller, dopo aver ricevuto la richiesta di creare un nuovo buffer, chiama la factory, che procede a creare il Buffer, che se non è presente viene aggiunto anche nella cache. Inoltre la cache è risultata utile anche per poter aggiornare la view successivamente al caricamento di un file, in quanto il controller ricava il solo nome dei file presenti nella cache e notifica alla view di aggiornarsi sulla base della lista dei file.

Extension

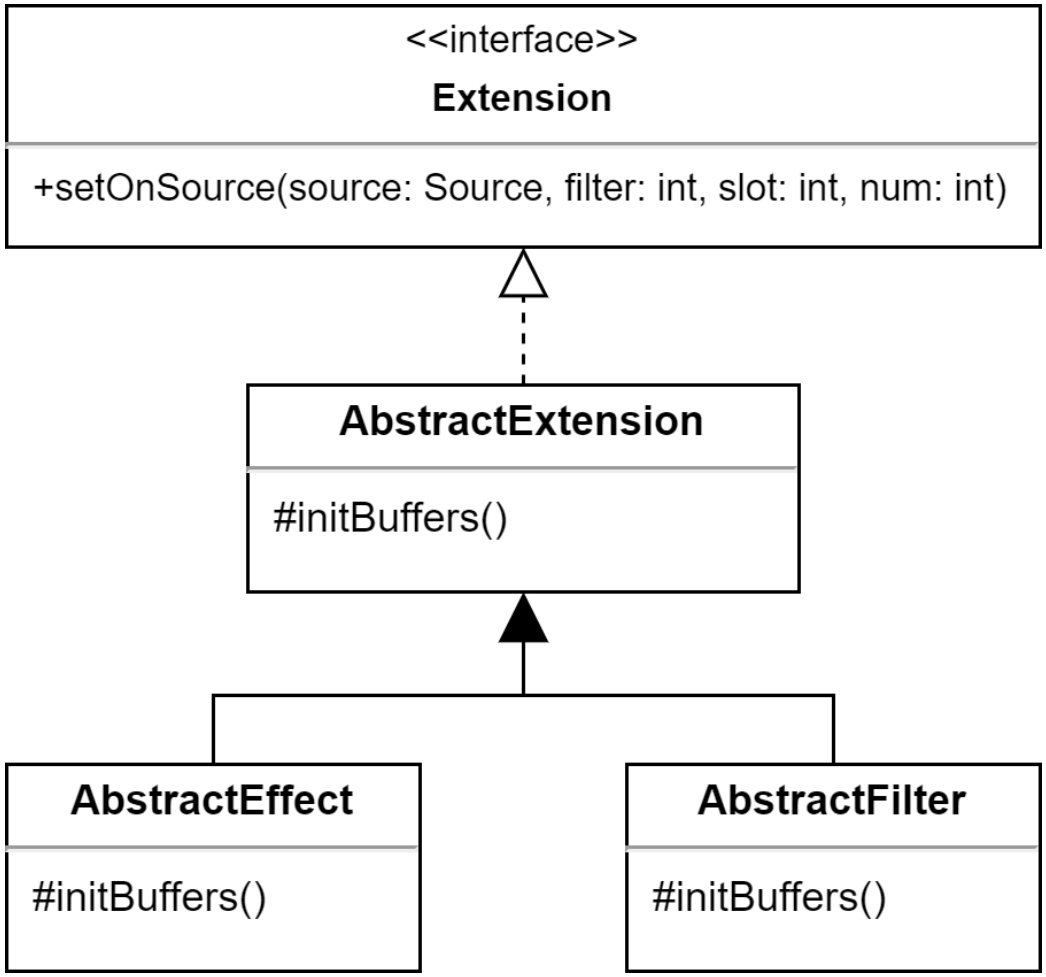

Figura 2.8: Rappresentazione UML della parte comune delle estensioni

Sia filtri che effetti condividono la parte in cui vengono applicati alla sorgente, ma non i buffer necessari alla creazione di essi, quindi ho deciso di creare un'interfaccia comune per la parte di applicazione, e successivamente una sua implementazione astratta, che implementa l'applicazione delle estensioni, ma non l'inizializzazione, la quale viene lasciata alle due classi che ereditano. Invece la creazione/rimozione del filtro o effetto attraverso la libreria era differente, quindi ho sfruttato due interfacce separate, mostrate nelle due figure successive.

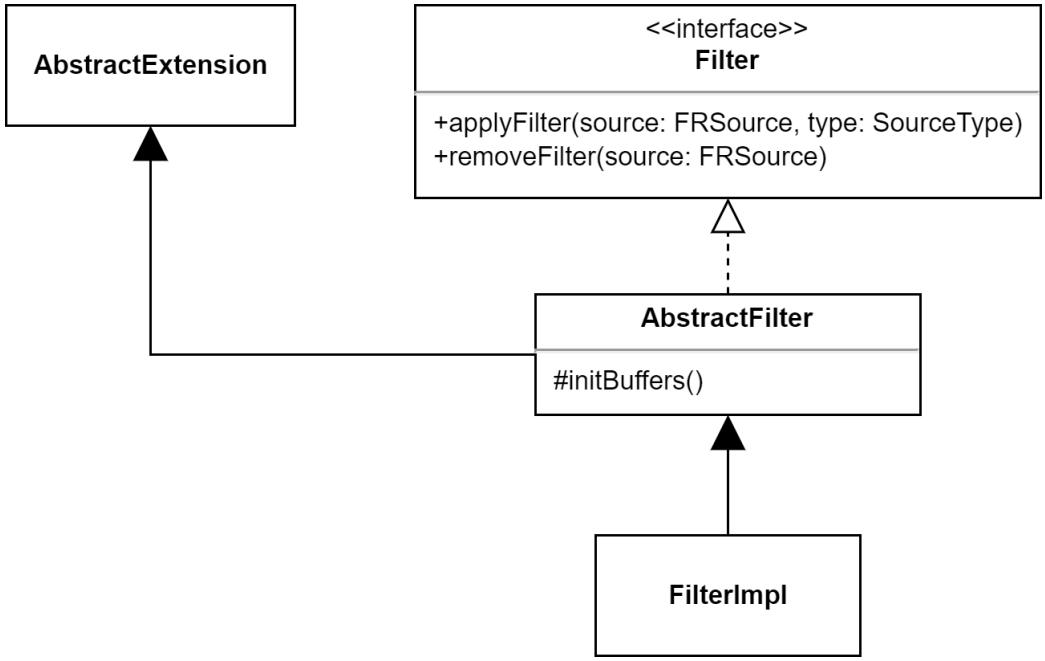

Figura 2.9: Rappresentazione UML della gestione dei filtri

I filtri hanno un valore impostato a priori per i tre tipi di filtro (passa-basso, passa-banda, passa-alto), quindi mi è bastato solo implementare la creazione, sulla base del filtro richiesto, e la rimozione.

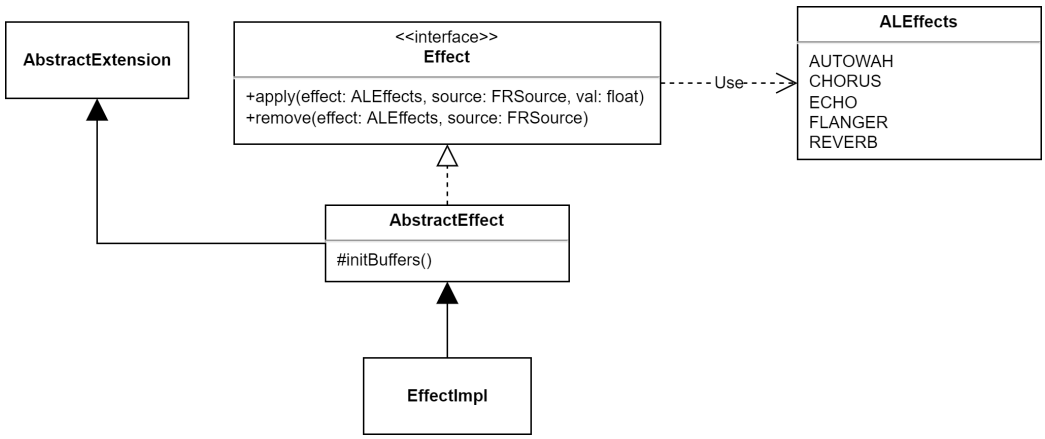

Figura 2.10: Rappresentazione UML della gestione degli effetti attraverso enum

Per la gestione degli effetti e dei suoi attributi mi sono affidato ad un enum con costruttore, che mi permettesse in maniera veloce di cambiare i parametri

di un dato effetto, come valore massimo e minimo, e di aggiungerli o rimuoverli. A differenza dei filtri, il valore degli effetti dipende dall'interazione dell'utente con la view.

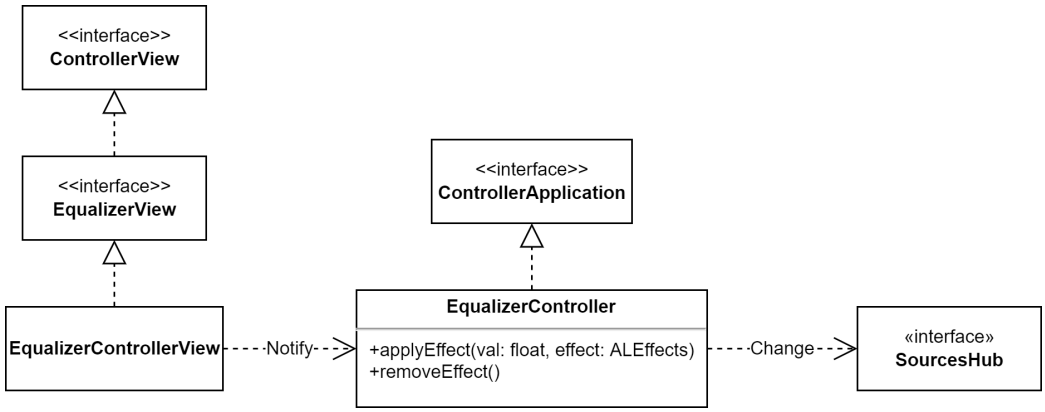

Figura 2.11: Rappresentazione UML dello schema MVC relativo all'utilizzo degli effetti

L'interazione dell'utente con l'equalizzatore, composto da slider, mi ha permesso di ricavare il valore dell'effetto il quale viene passato al controller che notificherà a tutte le sorgenti di applicare il determinato effetto con il determinato valore. In questa parte di MVC il controller non notifica nulla alla view, in quanto si tratta di un equalizzatore che non ha bisogno di aggiornare l'interfaccia in base alle azioni dell'utente.

2.2.2 Alessandro Sciarrillo

Il dominio su cui ho lavorato è quello delle sorgenti che comprende la loro implementazione sia come singole ma anche come gruppo.

Inizialmente mi sono occupato della strutturazione delle sorgenti che modellano i diversi tipi di speakers nelle sezioni a loro comuni e non. Per agevolare la riusabilità ho optato per una classe più primitiva "Source" che riguardasse l'utilizzo di base comune a ogni speaker, lasciando a "FRSource" (Frequency Range Source) solamente le funzioni relative al range di frequenze:

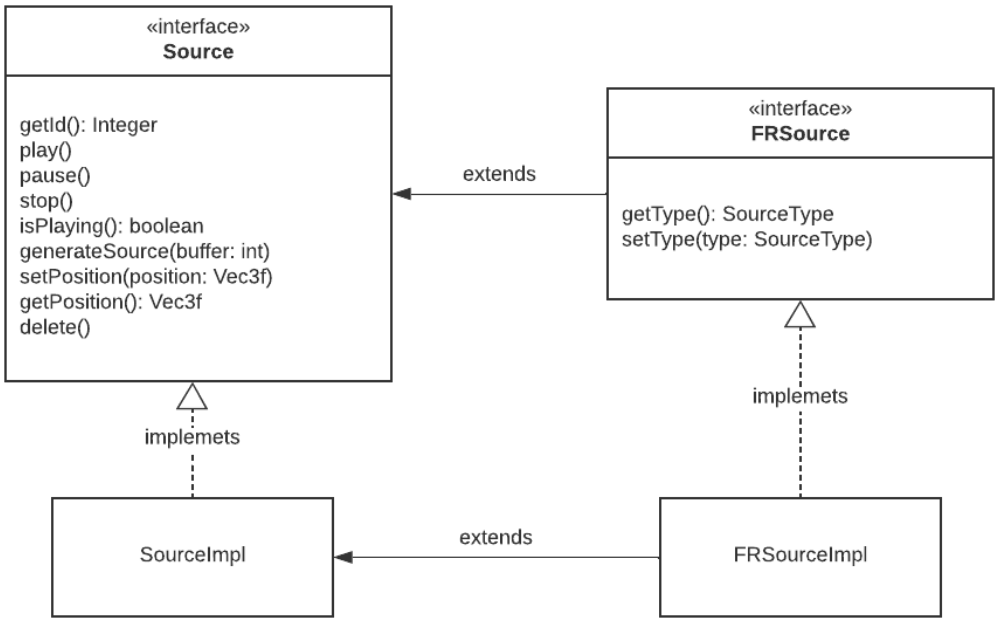

Figura 2.12: Riuso del codice da parte di FRSource.

Così facendo il codice scritto per la versione base "Source" è stato riusato come fondamenta della versione più avanzata "FRSource" lasciando però libertà in implementazioni future di creare altre forme avanzate di Source che non necessariamente debbano gestire range di frequenze.

Uno dei principali aspetti del mio lavoro riguarda l'implementazione di speakers con range di frequenza differenti, qui ho incontrato uno dei problemi strutturali maggiori, le qualità che volevo imprimere nella soluzione erano:

- Le sorgenti con range di frequenza eterogenei dovevano essere riconducibili a una classe padre comune a tutte.
- La quantità di codice necessaria ad implementare i vari tipi di range doveva essere estremamente ridotta.

Quindi la prima soluzione a cui ho pensato aveva questa forma:

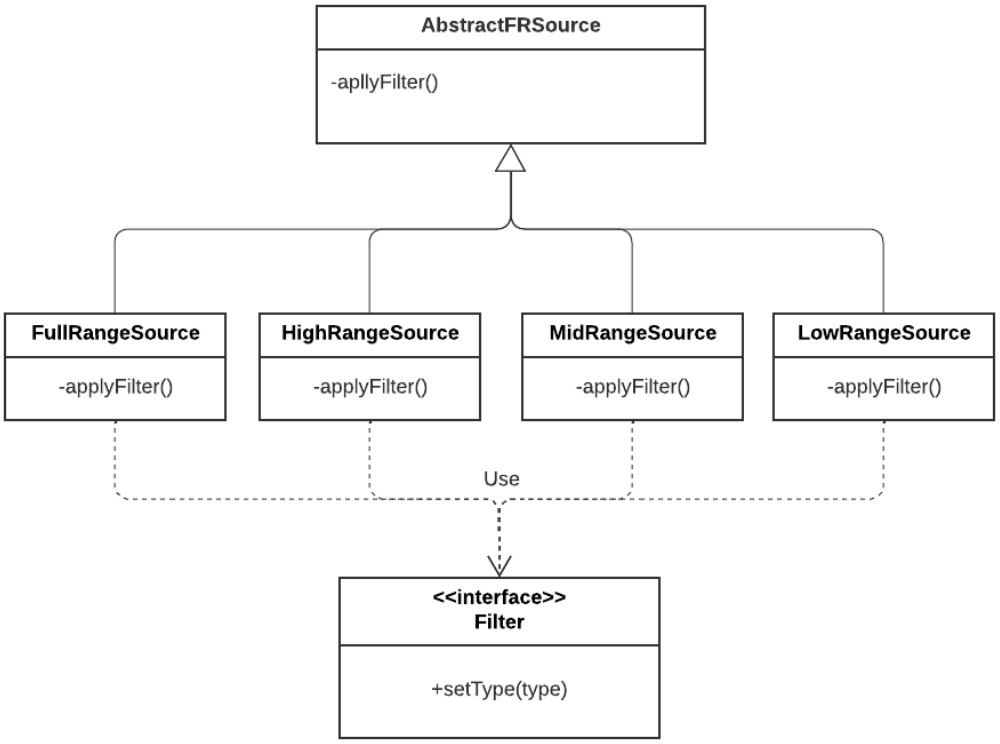

Figura 2.13: Prima "bozza" rappresentazione schematica FRSource.

Inizialmente sembrava scontata la necissità di un Template method per il metodo applyFilter, ma dopo una analisi più dettagliata del tipo di operazioni e lo scopo fondamentale di questa entità, cioè il poter cambiare in modo molto dinamico il range di frequenza, sono arrivato alla conclusione che un implementazione del template method in questo modo avrebbe appesantito il codice dell'utilizzatore, in quanto il cambiamento di range di frequenza per una istanza (ad esempio da High a Low) avrebbe comportato la creazione di una nuova istanza della classe relativa alla nuova frequenza, che sarebbe andata a sostituire quella precedente. In sostanza un cambiamento di frequenza avrebbe comportato il "passaggio" della source da una classe ad un altra quando la differenza tra quest'ultime è unicamente l'implementazione del metodo con cui impostano il proprio range di frequenza al buffer a loro associato. Così ho optato per la costruzione di una classe FRSource che avesse un proprio tipo di range di frequenza SourceType:

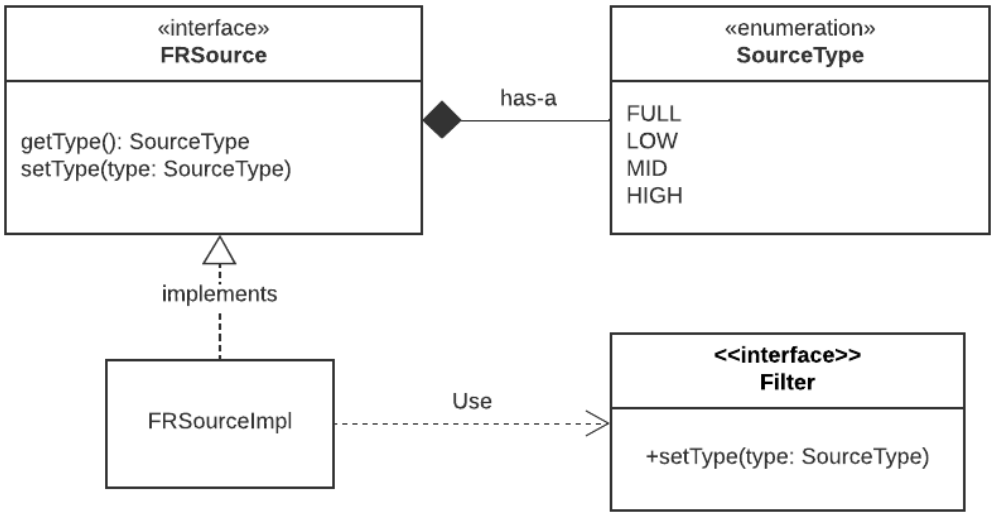

Figura 2.14: Rappresentazione struttura FRSource con ralazione "has-a" relativa a SourceType e l'implementazione FRSourceImpl che usa Filter.

In conclusione con questa struttura cambiare range di frequenza a uno speaker comporta unicamente una chiamata al metodo setType. Con altre strutture valutate invece, sorgeva sempre la necessità di compiere operazioni aggiuntive da parte del chiamante che voleva effettuare il cambio di range.

In seguito il primo passo nella gestione di un insieme di speaker è stato quello di ideare una struttura che permettesse di gestire le sorgenti come singole ma anche come gruppo e da qui è nato il concetto di SourcesHub:

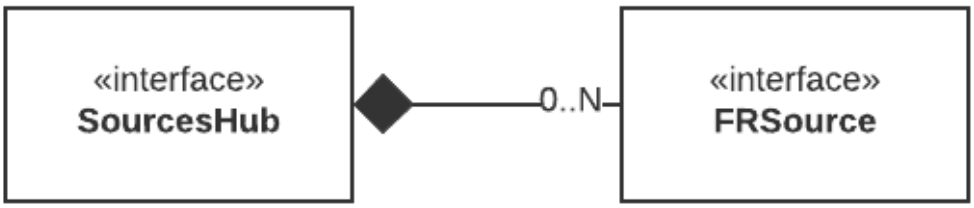

Figura 2.15: Rappresentazione schematica composizione di SourcesHub.

SourcesHub è, come spiega il nome, un Hub che permette di avere un insieme di Sources al suo interno e la cui funzione è indubbiamente utile per un progetto di questo tipo che spesso necessita di eseguire operazioni o procedure uguali su più Source che fanno parte dello stesso contesto.

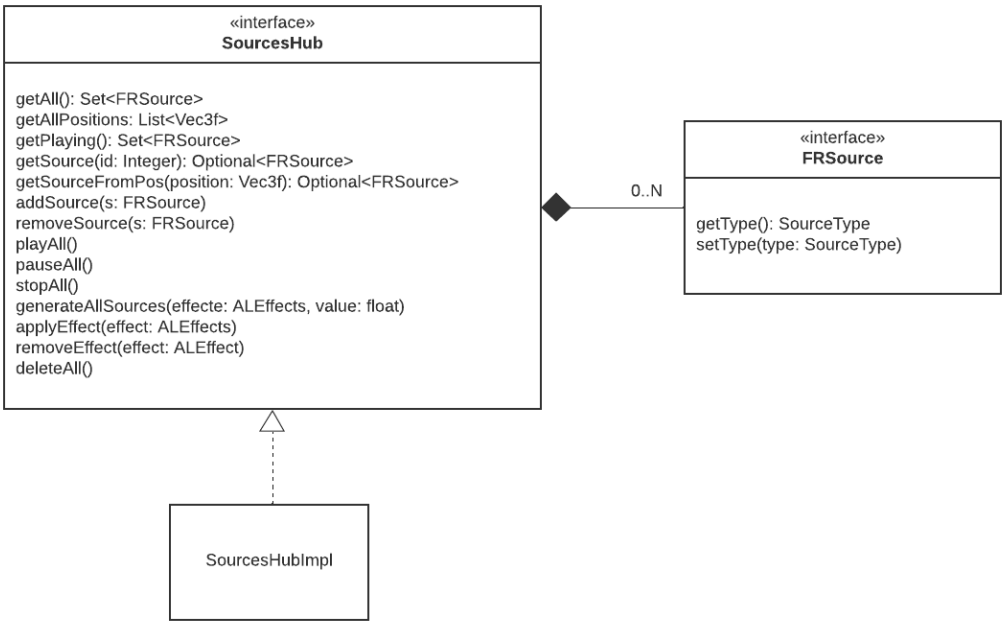

Figura 2.16: Rappresentazione schematica composizione di SourcesHub.

Successivamente per la creazione di Source/FRSource e SourcesHub ho seguito una strategia uguale per entrambe, ho creato un interfaccia con vari metodi che differiscono per parametri e il modo con cui creano le nuove istanze delle loro relative classi. Le ho poi implementate in un unico modo, lasciando comunque la possibilità di inserire nuove implementazioni che rispettino la stessa interfaccia, ma facendolo in modo diverso sfruttando così il Factory Method.

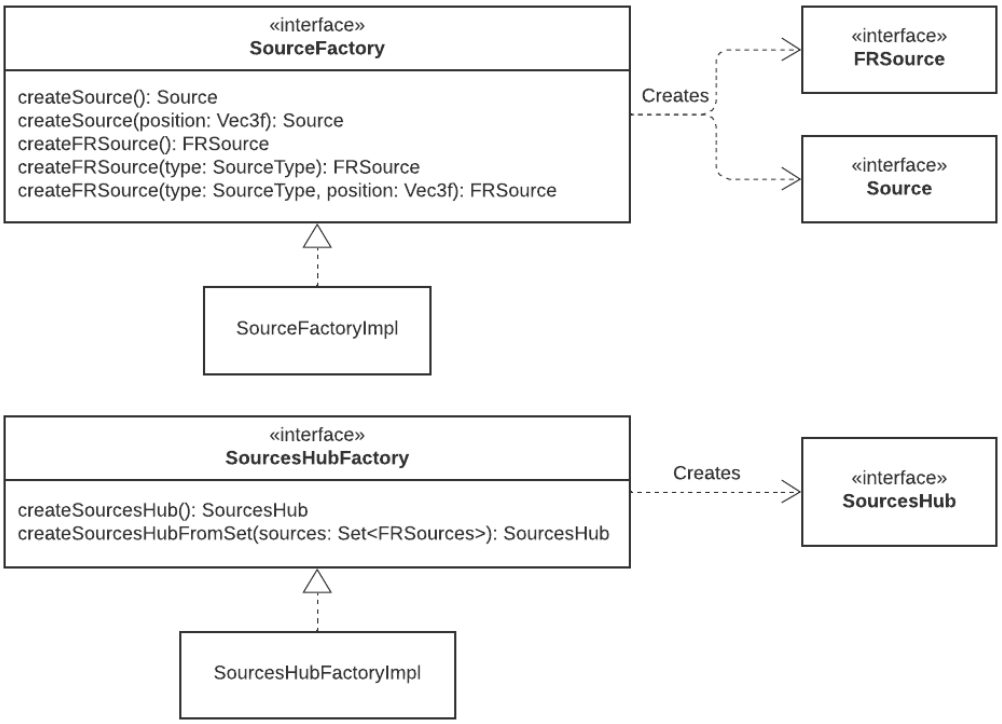

Figura 2.17: Rappresentazione SourceFactory e SourcesHubFactory.

La gestione dell'MVC è alquanto semplice, il SourceController aggiorna SourceControllerView interfacciandosi con SourceView, tra i due è presente una comunicazione bilaterale, poichè la View deve poter richiedere operazioni in risposta alle interazione con l'utente (aggiunta, rimozione e modifica di uno speaker) e allo stesso tempo il Controller deve poter aggiornare la View quando EnvironmentController lo notifica che lo speaker selezionato è cambiato e le componenti della view devono essere aggiornate per essere mantenute consonsistenti alle informazioni dello speaker correntemente selezionato. Il controller inoltre effettua i cambiamenti sulle sorgenti che sono contenute in SourcesHub all'interno di Environment interfacciandosi a lato pratico con EnvironmentController.

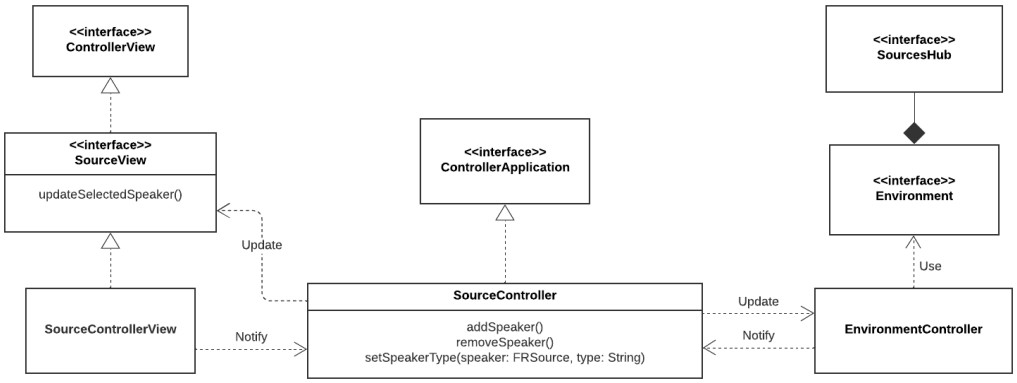

Figura 2.18: Rappresentazione schematica MVC.

2.2.3 Alex Presepi

La progettazione e lo sviluppo da me affrontato riguarda principalmente due argomenti: Listener e relativi plugin che ne estendono le funzionalità visive e/o sonore.

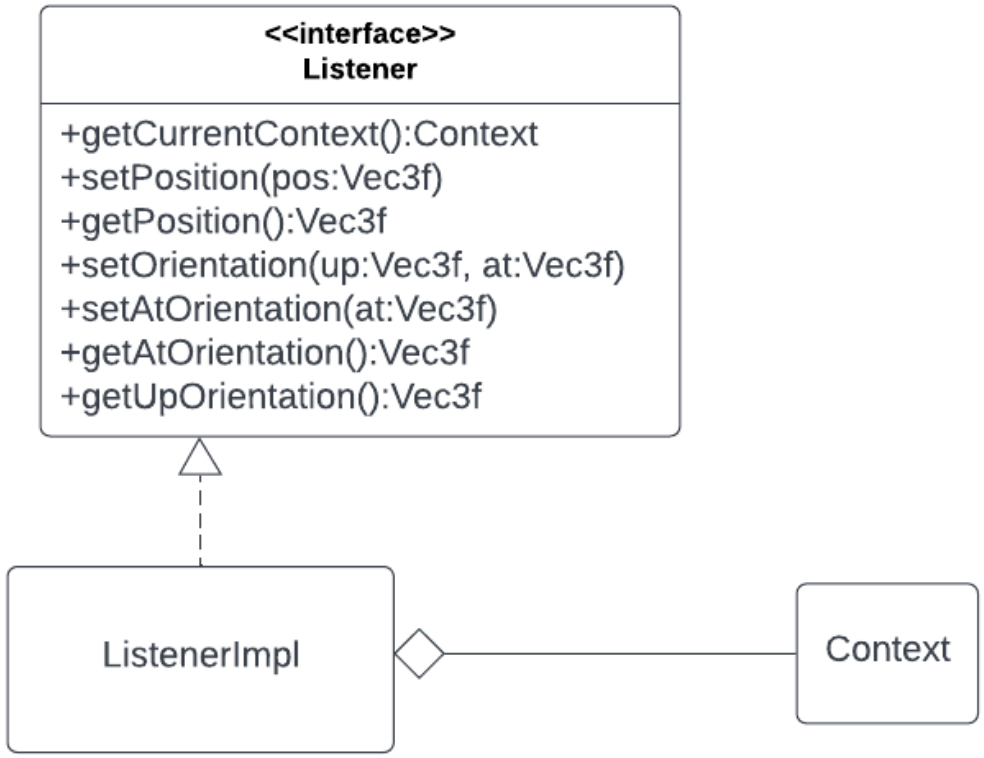

Figura 2.19: Rappresentazione UML dell' interfaccia Listener e relativa implementazione.

Il Listener deve essere univoco in ogni contesto di eseguzione perciò per risolvere questo primo problema ho optato per creare un Factory Method che permettesse di gestire l'univocità per ogni contesto e mettesse a disposizione diverse modalità di creazione. Cosi facendo, si mette a disposizione una struttura, per future implementazioni di diverse politiche di gestione dell'univocità. Quando si richiama il metodo di creazione, passandogli il contesto di esecuzione, esso controlla se per quel contesto esiste già un listener e ritorna o l'istanza presente o ne crea una nuova.

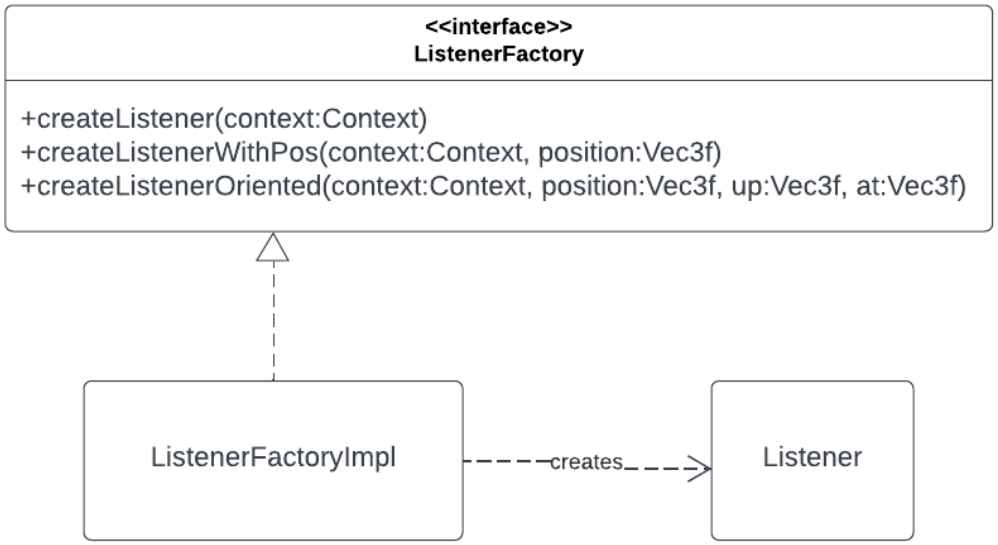

Figura 2.20: Rappresentazione della struttura Factory Method.

L' utilizzo del pattern MVC, della parte relativa al Listener, segue il filo generale descritto nella sezione precedente. In particolare il controller viene notificato dal EnvironmentController ogni volta che si verifica uno spostamento del Listener. Inoltre il controller è il punto di accesso per verificare quali Plugin sono presenti all'interno del ClassPath, e su richiesta notifica il ListenerControllerView. Il ListenerController gestisce anche la parte di management dei Plugin, attraverso la view si richiede una modifica e il controller notifica il PluginManager.

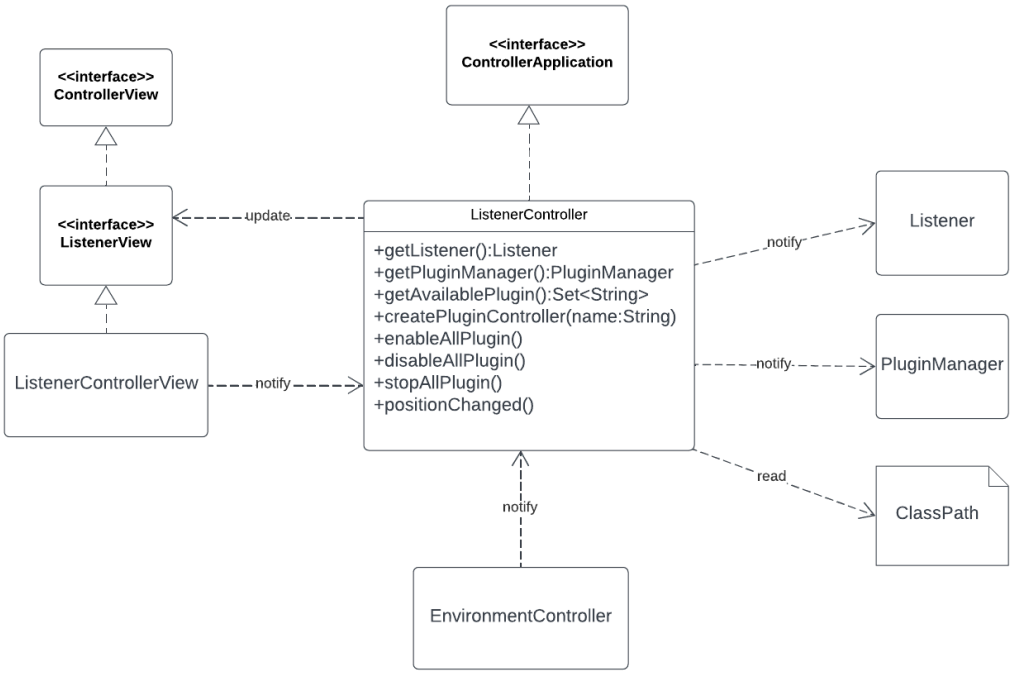

Figura 2.21: Design MVC per il Listener.

Nello sviluppo dei Plugin volevo ottenere una struttura il più modulare e estendibile possibile.
Per prima cosa ho creato un interfaccia e una classe astratta che contenessero le operazioni di base. Inoltre ho usato un Tamplate Method interno alla classe astratta (protected) per gestire le diverse fasi di salvataggio e ricaricamento dei settings, ogni volta che viene abilitato o disabilito il Plugin.

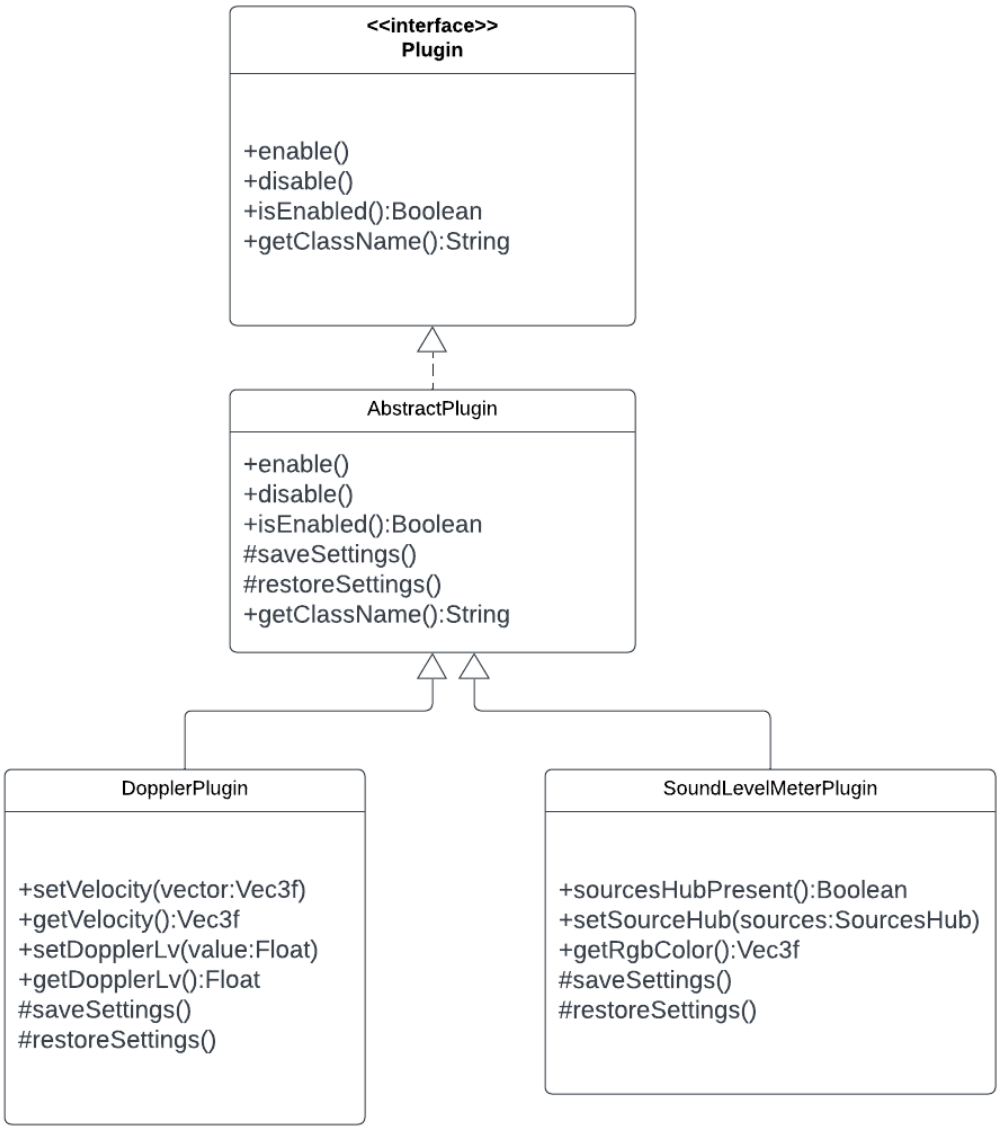

Figura 2.22: Utilizzo di Tamplate Method in Abstract Plugin.

Per la gestione dei Plugin ho scelto di creare un classe che facesse da manager in modo da poter effettuare in modo semplice ed organizzato operazioni comuni su tutti i componenti gestiti. In ottica futura risulterà utille per poter gestire più listener, dove ognuno avrà un proprio manager associato, o implementare e gestire più operazioni comuni tra Plugin.

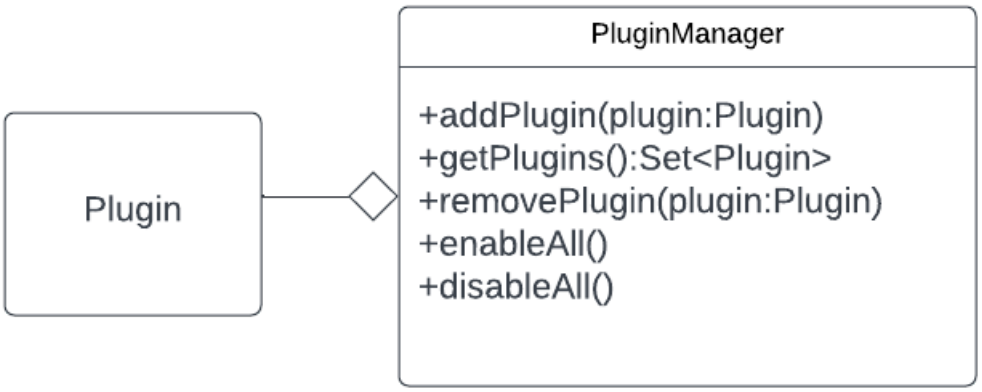

Figura 2.23: Schema UML della classe PluginManager.

La parte piu complessa nei Plugin è stata:

- la gestione dell'interfaccia utente.
- il caricamento di tutte le classi e risorse necessarie.

Per la comunicazione con la view ho creato un sotto pattern MVC. Cosi facendo lo sviluppo e l'integrazione dei Plugin non influisce in alcun modo con la struttura MVC dell' applicazione. Ogni plugin ha quindi la propria parte di model, di controller, e di view (controller e FXML), come nel MVC generale. Inoltre è presente un collegamento tra il controllerView del Plugin e il ListenerControllerView per aggiornare la view del applicazione, scelta dovuta dalla tecnologia grafica in uso. Infine ho reso possibile accedere ai ControllerApplication (MVC generale), per dare la possibilità di ottenere informazioni sullo stato dell' applicazione, mediante la composizione con il MainController.

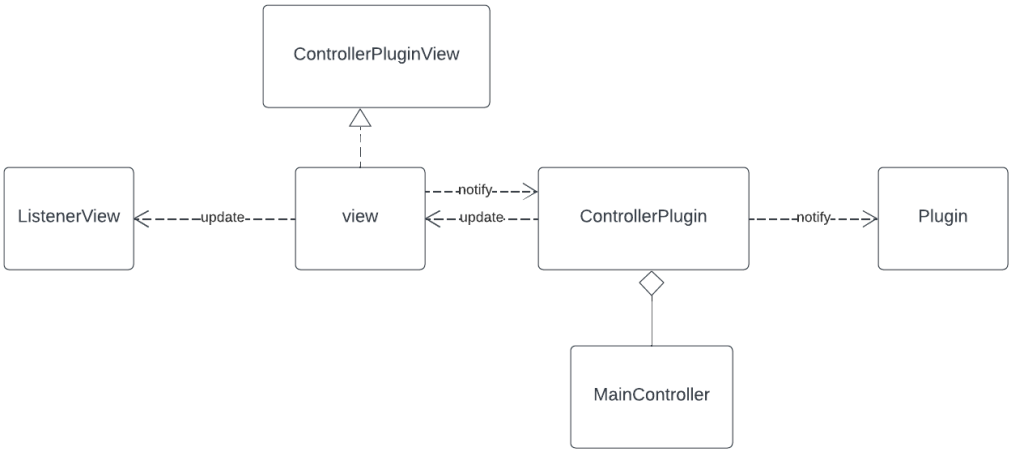

Figura 2.24: Utilizzo pattern MVC nei Plugin.

Per il secondo problema mi sono avvalso dell'utilizzo delle Reflection. Questa decisione è nata dal trovare una soluzione alla creazione di istanze di classi specifiche, che in fase di compilazione non conosco. L' utilizzo di esse aumenta l'estendibilità in quanto l'integrazione di plugin futuri è resa possibile senza intaccare il codice dell' applicazione. Il ListenerController crea via Reflection il ControllerPlugin desiderato e sarà esso a istanziare il model e la view relative a quel determinato Plugin. Grazie a quesa struttura, a cui sono arrivato dopo diverse analisi di design, si riesce a mantenere solida e indipendente la creazione di un Plugin, e tutte le risorse necessarie, dal resto dell' applicazione. Inoltre pone le basi per una possibile integrazione di Plugin anche durante l'esecuzione dell' applicazione (Hot-Swap), utile in questo tipo si software.

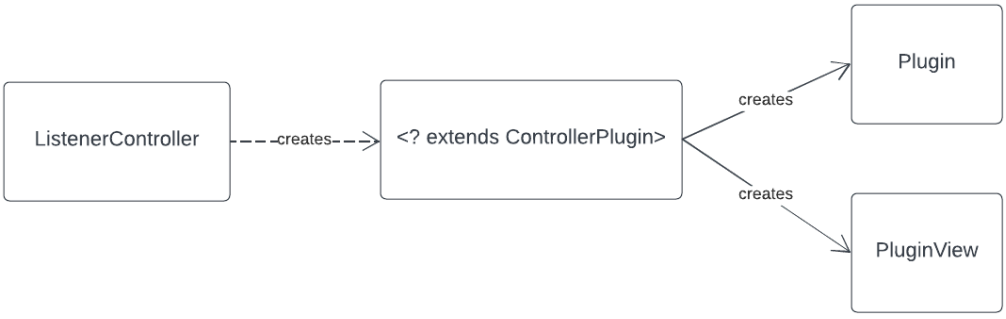

Figura 2.25: Schema creazione Plugin.

Infine per la realizzazione dei due Plugin, che sono riuscito ad implemen-

tare, mi sono tornati utili i Thread, per fare calcoli real-time continuativi. Ho pensato che l'utilizzo dei thread nei plugin fosse a discrezione di ogni plugin e fosse un caso che i due implementati utilizzassero entrambi dei thread. L' implementazione deve essere comunque indipendente da tutto, quindi ho deciso di creare una classe interna privata ad ogni ControllerPlugin che estendesse la classe Thread. In questo modo sono riuscito a gestire facilmente l'interrupt di terminazione del thread.

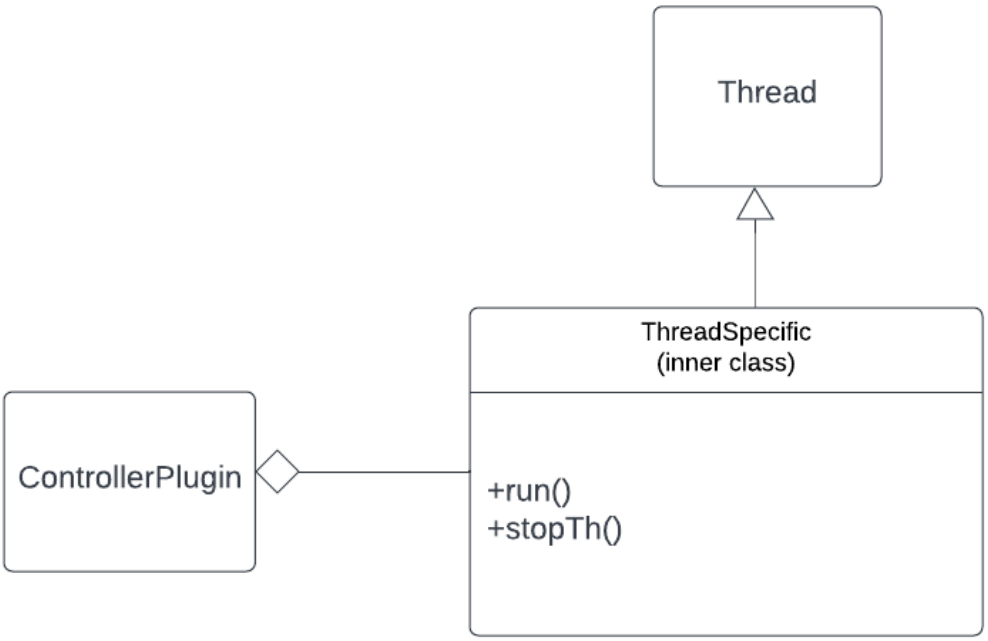

Figura 2.26: Schema gestione Thread.

2.2.4 Simone Lugaresi

Gli argomenti principali da me elaborati sono 3:

- Space, che serve a delimitare uno spazio 2D di lavoro nel quale si potranno muovere Source e Listener.
- Environment, che è il "contenitore" di tutti gli elementi, sourceHub, Listener, space.
- AudioManager, inizializza il contesto audio della libreria che abbiamo utilizzato.

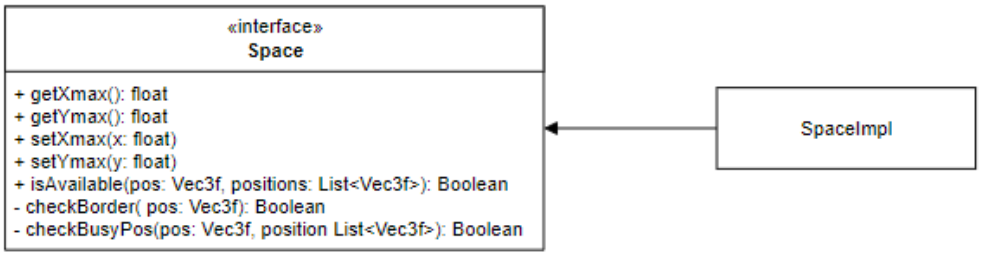

Figura 2.27: Rappresentazione UML dell'nterfaccia Space e la sua implementazione

Space si pone l'obbiettivo di risolvere il problema di delimitare uno spazio in cui poter utilizzare e spostare gli elementi come possono essere le casse (source) o l'ascoltatore(listener).
In questa implementazione viene utilizzato principalmente per fissare le dimensioni massime degli ambienti, dato che la maggior parte dei controlli sulla corretta posizione degli oggetti viene gestita dalla View, rimane però la possibilita di controllare le posizioni degli spostamenti nell'ipotesi di un'interfaccia da console.

Figura 2.28: Rappresentazione UML della Factory per Space

Per agevolare la creazione di Space, sia da utilizzare nei preset degli Environment sia per i Test, mi sono avvalso del template Factory Method, partendo dall'interfaccia SpaceFactory che da la possibilità di scelta su due tipi di creazione di space, uno con dimensioni di default e uno con paramentri, dando però la possibilità in un futuro ad un'ulteriore implementazione della factory.

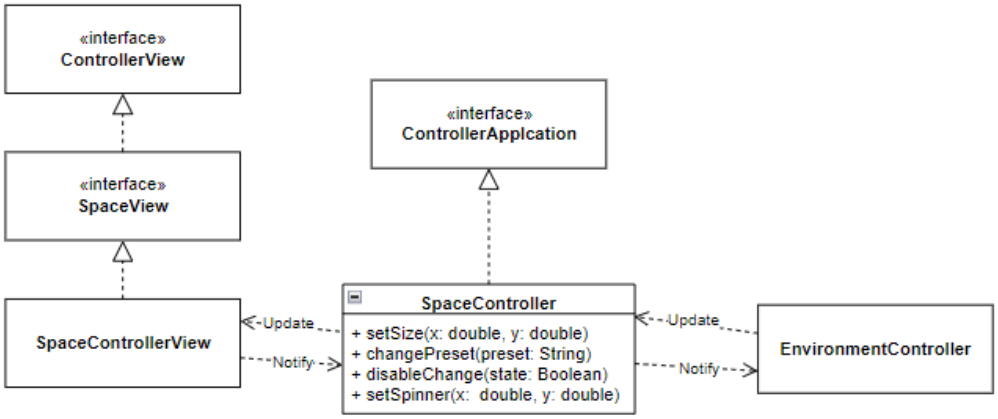

Figura 2.29: Rappresentazione UML della gestione del MVC con space

La parte di MVC che riguarda Space è molto semplice, SpaceController comunica dinamicamente con SpaceView che notifica quest'ultimo qualora l'utente cambiasse dei settaggi, in tal caso si rende necessaria anche una comunicazione con Environment per apportare tali modifiche, questa comunicazione per un corretto funzionamento necessita di una sorta di sistema half-duplex in modo da poter aggiornare anche i dati di SpaceView inviati da Environment, un esempio può essere la selezione di un determinato preset.

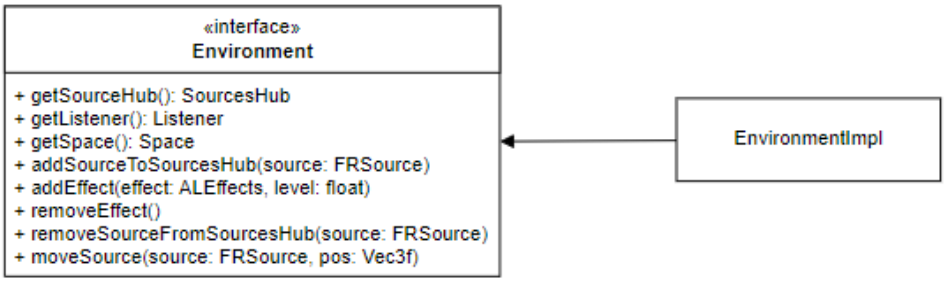

Figura 2.30: Rappresentazione UML della parte di Environment

Environment ha come scopo migliorare l'approccio agli elementi del programma quali listener, source, effects, raggruppandoli tutti insieme in un solo "ambiente", in modo da poterlo rendere unico e riutilizzabile, anche in un ottica di un futuro aggiornamento nel quale si possono gestire più Environemnt e lavorare in contemporanea su più di uno. Per una creazione più semplice e user friendly possibile mi sono servito anche qui di Factory Method: nell'interfaccia principale, EnvironmentFactory, do la possibilità di scelta tra un

Environment vuoto, creato tramite i parametri Sources-Listener-Space, oppure tramite un file JSON correttamente impostato, ma lasciando cosi libertà di implementazione di nuove factory, magari con diversi tipi di struttura del JSON, etc.

Figura 2.31: Rappresentazione UML della factory di Environment

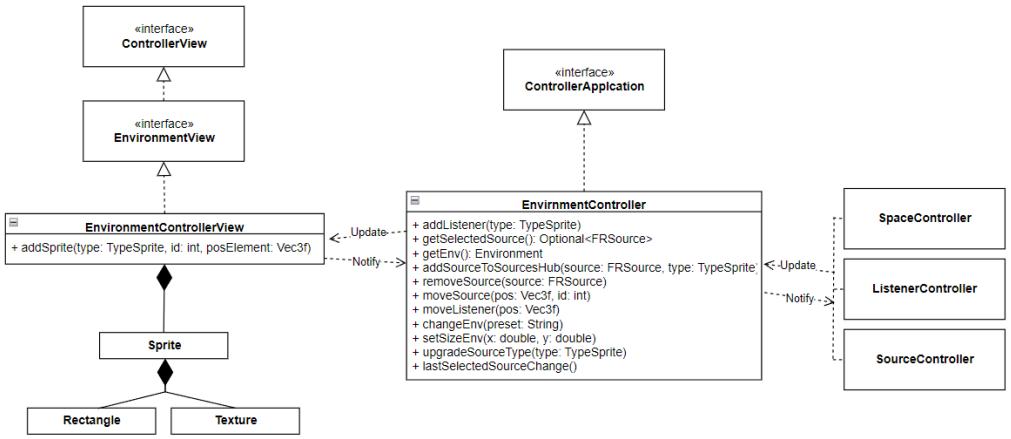

Figura 2.32: Rappresentazione UML della gestione del MVC con Environment

La parte di MVC per Environment è stata gestita similmente alle altre: EnvironmentController viene notificato dalla sua View di eventuali cambiamenti, che possono essere di un solo genere ossia lo spostamento di un elemento nell'ambiente, sarà poi EnvironmentController a comunicare con i controller degli altri elementi interessati, come Source e Listener, che però rimarrà aperto a ricevere update da quest'ultimi, o da altri controller come SpaceController, e se necessario applicandoli alla view.

Figura 2.33: Rappresentazione UML dello schema semplice di AudioManager

Si tratta dell'implementazione di tre semplici classi, di cui AudioManager è statica, che pero sono necessarie per una corretta inizializzazione della libreria e, di conseguenza, delle nostre classi. Hanno lo scopo di creare il contesto di esecuzione per il listener e le source, quindi per la nostra implementazione al supporto di un singolo Environment per volta abbiamo optato per una soluzione statica.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

La suite utilizzata per i test automatizzati è JUnit. Durante il processo di implementazione siamo stati inizalmente accompagnati dai test effettuati tramite le actions di GitHub, che ne verificavano l'assenza di errori sui principali sistemi operativi (Windows, MacOSX, Ubuntu), ma che abbiamo dovuto abbandonare a causa della mancanza di una scheda audio nei sistemi su cui venivano eseguite che causava errori dato che vari test vengono inizializzati attraverso essa.
In particolare, si è deciso di sottoporre a test automatizzato i seguenti componenti:

- Buffer: test di caricamento sia da file system che da classpath, oltre al test del corretto funzionamento del caching.
- Filter: test di applicazione dei tre tipi di filtri su delle sorgenti.
- Space: test di una corretta esecuzione dei metodi, e dei loro risultati con determinati input.
- Environment: test di corretta counicazione tra tutti gli elementi presenti dentro Environment, e i suoi metodi.
- Source: test sulla consistenza degli stati di riproduzione delle sources (play-pause-stop), il loro spostamento e la corretta applicazione di filtri.
- SourcesHub: test sui metodi con cui ci si interfaccia a SourcesHub per compiere operazioni sulle FRSource come aggiunta, rimozione, riproduzione e spostamento sia agendo su esse come singole, che come gruppo.

- Listener: test di corretto settaggio di posizione e orientamento, e test sulla gestione dell'univocità del listener per singolo contesto.
- Plugin: test sulle operazioni di gestione dei plugin nel manager, e singoli test di corretto funzionamento dei plugin implementati.

Altre funzionalità richiedevano un test uditivo per verificarne la correttezza, che ci era difficile verificare in modo automatizzato.

3.2 Metodologia di lavoro

Una ingente parte di ricerca è stata svolta prima della creazione del progetto e dell'analisi, poichè avevamo la necessità di verificare la fattibilità del progetto che avevamo ideato e che ha comportato la valutazione di varie librerie con tutto quello che ne consegue, come test di prototipi e studio delle documentazioni. La scelta finale è ricaduta su OpenAL, cercando successivamente librerie Java che facessero da API ad essa, di cui abbiamo valutato JOAL e LWJGL, selezionando quest'ultima dopo un'analisi dei pro e contro di entrambe da cui abbiamo riscontrato incompatibilità critiche in JOAL.
Visto che nessuno dei componenti aveva esperienza pregressa nell'utilizzo di GitHub in progetti di questa entità, abbiamo optato per una gestione di base attraverso una semplice strategia "push e pull", lavorando spesso sullo stesso branch, salvo poche eccezioni, per evitare complicazioni e concentrarci sulla pura implementazione.
Il lavoro in comune è stato effettuato per definire le interfacce che servivano come punto di accesso tra le diverse parti, nei componenti "manager" (come MainController e MainControllerView) e nei vari package utility.

Niccolò Mussoni

Buffer, BufferCache, Extensions (Effect + Filter), view per il player e per l'importazione di un file WAV, view per l'applicazione degli effetti su tutte le sorgenti attraverso un semplice equalizzatore, con relativi SongController e EqualizerController per la comunicazione tra view e model.

Alessandro Sciarrillo

Source, FRSource, SourcesHub, view per la creazione, eliminazione, modifica ed esposizione dati degli speakers.

Alex Presepi

Listener, Plugin, PluginManager, view per la visualizzazione e la gestione dei parametri del Listener, e view relativa alla gestione dei plugin creazione, abilitazione, disabilitazione e chiusura di essi.

Simone Lugaresi

Environment, Space, view per la gestione degli elementi all'interno della canvas, view per la gestione degli spazio simulati.

3.3 Note di sviluppo

Niccolò Mussoni

- Lambda expressions
- Stream
- Optional
- JavaFX, per la GUI.
- Gradle Build Tool.
- OpenAL, libreria audio su cui si basa l'intero progetto.
- LWJGL, libreria che mette a disposizione delle API per la libreria OpenAL.
- Spring Framework, per accedere a più risorse all'interno di una cartella del classpath.

Per la conversione da WAV stereo a WAV mono mi sono affidato a del codice online, consultabile *qui*, non avendo le competenze necessarie per la manipolazione dei file WAV, che ho poi adattato alla singola conversione di cui necessitavo.

Alessandro Sciarrillo

- Lambda expressions
- Stream

- Optional
- JavaFX (GUI)
- Gradle Build Tool.
- OpenAL, libreria audio su cui si basa l'intero progetto.
- LWJGL, libreria che mette a disposizione delle API per la libreria OpenAL.

Alex Presepi

- Lambda expressions
- Stream
- Optional
- JavaFX (GUI)
- Gradle Build Tool.
- Reflection
- Thread, semplice utilizzo per calcoli real-time.
- OpenAL, libreria audio su cui si basa l'intero progetto.
- LWJGL, libreria che mette a disposizione delle API per la libreria OpenAL.
- ClassGraph, libreria per visualizzare le classi caricate nel ClassPath, utilizzata per cercare i plugin presenti.

Simone Lugaresi

- Lambda expressions
- Stream
- Optional
- JavaFX
- Gradle Build Tool.

- OpenAL, libreria audio su cui si basa l'intero progetto.
- LWJGL, libreria che mette a disposizione delle API per la libreria OpenAL.

Per la gestione degli sprite all'interno della canvas mi sono basato sull'implementazione di codice visto online che poi ho modificato e riadattato alle mie esigenze: `https://github.com/stemkoski/BAGEL-Java-2022`(che però ora è in una versione più aggiornata rispetto alla versione dalla quale avevo preso spunto io, di conseguenza alcune classi potrebbero non esserci piu o essere state cambiate.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Niccolò Mussoni

Essendo il primo progetto al quale lavoravo in team e considerando le dimensioni del progetto mi ritengo abbastanza soddisfatto del lavoro svolto. Certi aspetti dello sviluppo potevano essere svolti in modo migliore, ma la continua comunicazione tra i componenti ci ha portato a risolvere molti dei problemi sorti. Nella progettazione non penso di aver fatto un brutto lavoro: credo che la parte di Extension potesse essere progettata in maniera migliore, soprattutto per l'utilizzo di un enum, mentre in quella relativa al Buffer ritengo sia stata effettuata una buona analisi e progettazione.
Nell'immediato non credo di essere interessato a portare avanti il progetto, vista la mia volontà di volermi focalizzare su nuovi progetti, ma in futuro non escludo di poterci rimettere mano per includere nuove parti della mia sezione, ampliare le possibilità di utilizzo e migliorarne il modo in cui è stato progettato.

Alessandro Sciarrillo

Considerando il monteore a disposizione e il prodotto finale valuto la mia prestazione in modo positivo. Sicuramente c'è molto margine di miglioramento nell'utilizzo di MVC, che abbiamo cercato di implementare al meglio, ma non è di certo nella sua forma ideale, visto che è il primo vero progetto in cui lo applichiamo e non è qualcosa che si impara a utilizzare al meglio con una esperienza così ridotta. Un grande punto di forza della mia parte penso che sia la sua estendibilità per quanto riguarda le Source ma anche il corretto utilizzo della libreria LWJGL che a inizio progetto era del tutto nuova. D'al-

tra parte riconosco una carenza nell'uso dei generici che avrebbero permesso un riutilizzo più flessibile del SourcesHub e che sicuramente avrei integrato se avessi avuto più tempo. Senza dubbio questo è un progetto che non si fermerà qui per quanto mi riguarda, soprattutto perchè l'idea iniziale è nata da me e il risultato attuale è molto vicino a quella che era la mia visione, che vorrei quindi portare avanti, sia per rivedere e migliorare l'architettura generale ma anche per aggiungere nuove funzionalità.

Alex Presepi

Valutando il tempo messo a disposizione per sviluppare il progetto mi sento soddisfatto e appagato. Guardando l'applicazione in sé, mi sarebbe piaciuto arrivare ad un livello di dettaglio del suono, simulato, più preciso e realistico. Per questo mi piacerebbe continuare lo sviluppo per rendere l'applicazione davvero utile. Preciso che per me è il primo progetto di questo livello, sviluppato in gruppo, e nell'utilizzo del DVCS riconosco una non buona gestione, ho però imparato molto grazie ad esso. Penso che la struttura dei Plugin da me realizzata goda e soddisfi vari principi, anche se avrei potuto creare delle basi più astratte per implementare plugin non solo relativi al Listener. Ultima nota che vorrei precisare: nel post del progetto non si citavano i Plugin sui quali ho principalmente lavorato perchè pensavo a delle feature singole, poi in fase di progettazione dettagliata/sviluppo ho optato per quest'idea.

Simone Lugaresi

Come primo vero progetto mi ritengo abbastanza soddisfatto di come è stato implementato, mi piace molto la facilità di comunicazione che hanno le diverse classi e i controller grazie all'impostazione che abbiamo scelto di seguire. Della mia parte in generale mi posso ritenere mediamente soddifatto, mi piace la semplicità di utilizzo, la ritengo minimale ed efficace, sicuramente con l'implementazione di qualche pattern un po' più sofisticato avrei migliorato la riusabilità.
Personalmente ritango che 80 ore siano un po' sproporzionate al carico di lavoro che necessita un progetto del genere per essere implementato almeno "benino", considerando anche la prova scritta e tutto il resto. Alla fine il prodotto finale mi soddifa molto, le idee per la continuazione e l'implementazione per nuove feature ci sono, staremo a vedere se ci sarà il tempo.

4.2 Difficoltà incontrate e commenti per i docenti

Se si avvia l'applicazione da terminale è possibile che la libreria OpenAL stampi due warning di questo tipo:

```
[ALSOFT] (EE) Failed to set real-time priority for thread:  Operation not permitted (1)
```

Questo è dato dal fatto che in alcuni sistemi Linux per una impostazione di sistema il tempo di esecuzione di un thread ha un valore che OpenAL valuta come non ottimale per una esecuzione real-time. Ricercando su vari blog specifici abbiamo appreso che è giusto ignorarlo se non si riscontrano problemi come nel nostro caso. Non siamo riusciti a sopprimeli poichè era necessaria la modifica di una variabile di sistema.

Appendice A

Guida utente

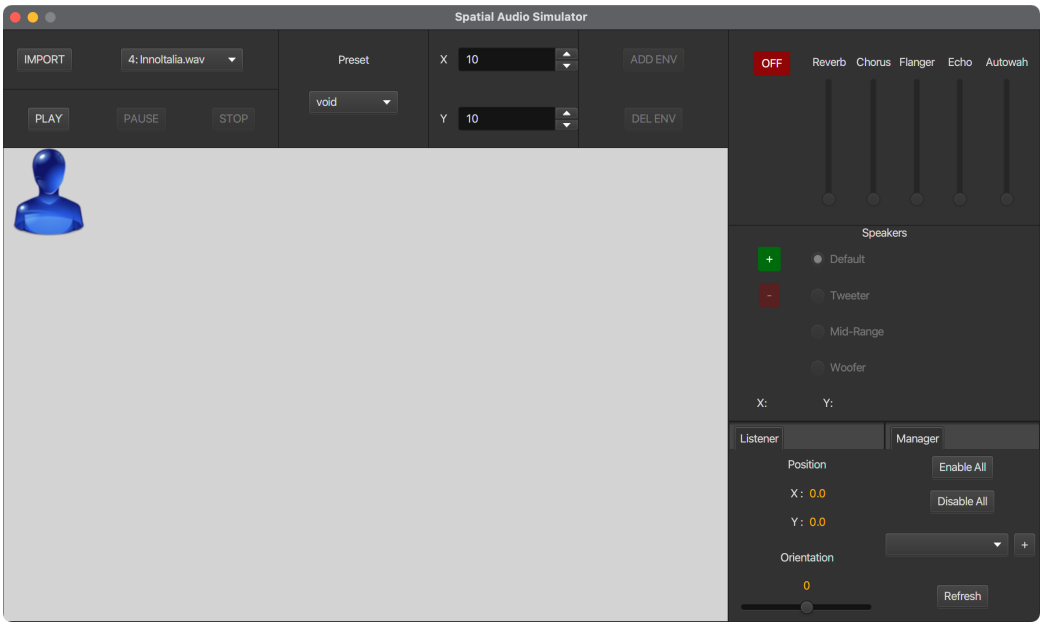

Figura A.1: Schermata completa

All'avvio l'applicazione si presenta nel seguente modo. La schermata è divisa in varie sezioni:

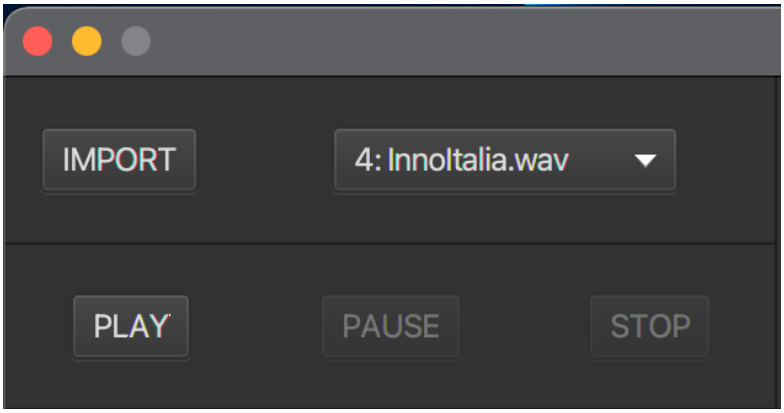

Figura A.2: Player

In questa sezione è possibile importare un file WAV o scegliere tra alcuni preimportati e avviare, mettere in pausa o stoppare la traccia selezionata.

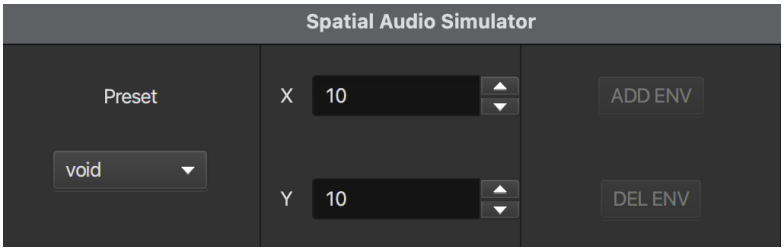

Figura A.3: Gestione dell'ambiente di ascolto

Da qui è possibile selezionare un ambiente di ascolto già pronto e modificare le dimensioni virtuali dell'ambiente stesso.

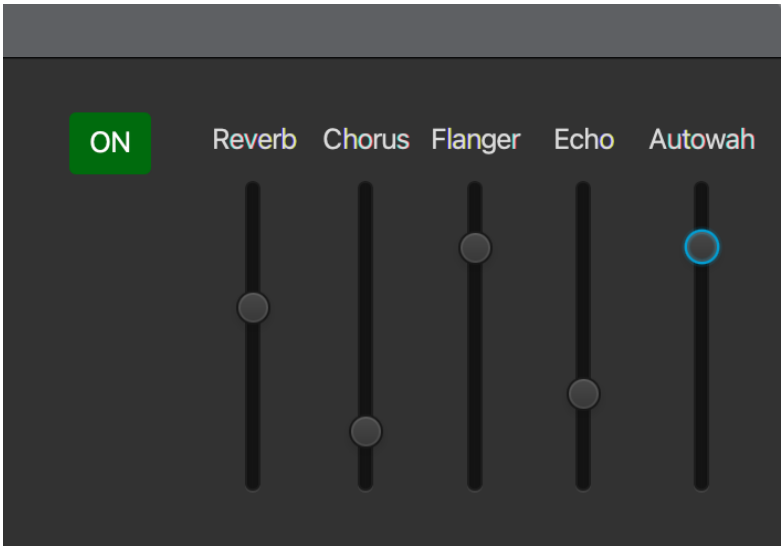

Figura A.4: Equalizzatore

Questo è un piccolo equalizzatore che è possibile accendere o spegnere tramite il bottone ON/OFF. Quando è abilitato, è possibile applicare diversi effetti sull'intero ambiente d'ascolto modificandone il livello, mentre allo spegnimento rimuove tutti gli effetti.

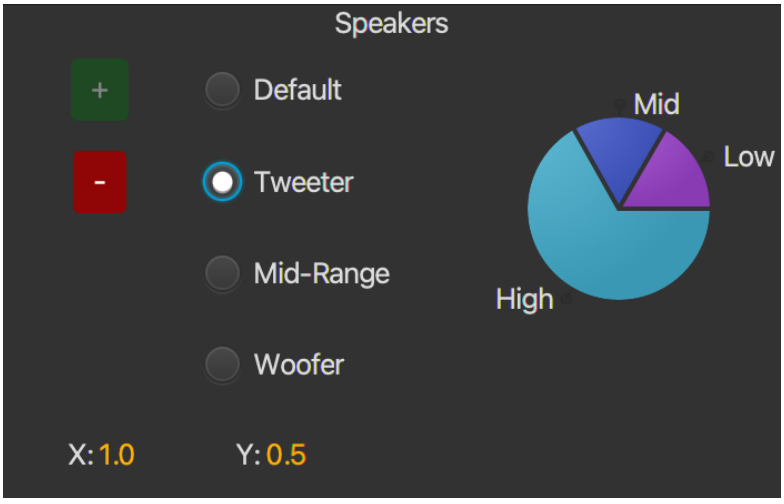

Figura A.5: Gestione degli speaker

Questa è la sezione relativa agli speakers da cui si possono aggiungere e rimuovere speakers, cambiare il range di frequenza di quello selezionato e mostrare la sua posizione. Inoltre è presente un grafico a torta che mostra

come sono distribuite le frequenze dell'insieme di speakers presenti, aiutando così l'utente a creare un ambiente di ascolto bilanciato.

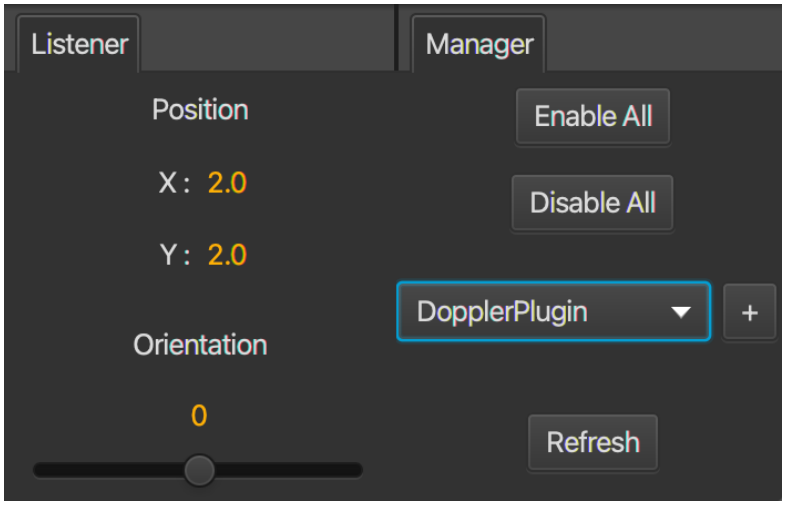

Figura A.6: Gestione dell'ascoltatore

Nella sezione dell'ascoltatore è possibile visualizzare la posizione di esso all'interno dell'ambiente, modificarne l'orientamento e aggiungere dei plugin ad esso. In caso di aggiunta o chiusura di plugin il tasto Refresh riaggionerà la lista dei plugin disponibili all'utilizzatore.

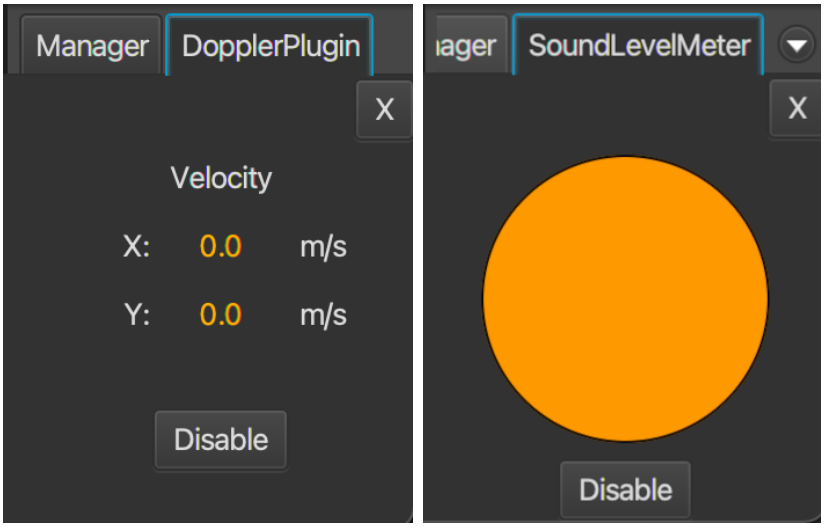

Figura A.7: Esempi di plugin

Figura A.8: Ambiente d'ascolto

Infine è presente l'ambiente d'ascolto, dove l'utente può muovere a suo piacimento l'ascoltatore e gli speakers.

Appendice B

Esercitazioni di laboratorio

Niccolò Mussoni

- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p136373
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p139071
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136368
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p139072
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p138604
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p139736

Alessandro Sciarrillo

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136652
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p139086
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p139984

Simone Lugaresi

- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p136825
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p136726
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136582
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p138776
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p140008