Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería Computación

Softwaver: Plataforma de Síntesis de Audio en Tiempo Real con Wavetable y Modulación Avanzada

---

Alumno: D. Sergio Ramos Galindo
Tutor: D. Jesús María Vegas Hernández

Agradecimientos

Quiero agradecer a Curtis Roads por escribir *The Computer Music Tutorial* libro que me ha hecho entender los fundamentos del procesamiento de audio digital y me ha empujado a perseguir una carrera profesional en este ámbito.
También quiero agradecer a Óscar Aragón Esteban por su ayuda y resolución de dudas del proyecto. Su experiencia y veteranía han sido un buen apoyo.

Resumen

El objetivo de este proyecto es diseñar e implementar una aplicación software de síntesis de audio en tiempo real, capaz de generar y manipular sonidos de manera flexible y creativa. Para lograrlo, se utilizarán osciladores wavetable, que permiten la generación de formas de onda complejas y ricas en armónicos, junto con un generador de ruido para añadir texturas sonoras adicionales.

El sistema incluirá filtros paso bajo (LPF) para modelar y suavizar las señales de audio digital, así como módulos de modulación mediante envelopes y osciladores de baja frecuencia (LFO), que permitirán variar dinámicamente parámetros como la amplitud o el filtrado a lo largo del tiempo.

La aplicación contará con una interfaz gráfica de usuario (GUI) que facilitará el diseño de sonidos personalizados, permitiendo a los usuarios ajustar y combinar los diferentes módulos de síntesis. Además, se implementará la funcionalidad de almacenar y cargar presets, lo que posibilitará guardar configuraciones de sonido para su uso posterior.

Finalmente, el software será extensible mediante la integración de controladores MIDI hardware, ofreciendo a los usuarios la posibilidad de interactuar con el sintetizador de manera física.

Abstract

The aim of this project is to design and implement a real-time audio synthesis software application capable of generating and manipulating sounds in a flexible and creative way. To achieve this, wavetable oscillators, which allow the generation of complex and harmonically rich waveforms, will be used, together with a noise generator to add additional sound textures.

The system will include low pass filters (LPF) for shaping and smoothing digital audio signals, as well as modulation modules using envelopes and low frequency oscillators (LFOs), which will allow parameters such as amplitude or filtering to be dynamically varied over time.

The application will feature a graphical user interface (GUI) that will facilitate the design of custom sounds, allowing users to adjust and combine the different synthesis modules. In addition, the functionality to store and load presets will be implemented, making it possible to save sound configurations for later use.

Finally, the software will be extensible through the integration of hardware MIDI controllers, offering users the possibility to interact with the synthesiser in a physical way.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El sintetizador es una herramienta necesaria para un músico o productor, utilizada en estudios de grabación, conciertos y experimentación de sonidos. Softwaver, una plataforma de síntesis de audio en tiempo real con tecnologías avanzadas, nace como respuesta a esta necesidad.

Este documento detalla el desarrollo e implementación de Softwaver y su potencial como herramienta creativa.

1.1. Contexto

El sintetizador ha sido una de las herramientas más revolucionarias en la producción musical desde su aparición en la segunda mitad del siglo XX. Su desarrollo permitió la creación de sonidos que no podían ser generados por instrumentos tradicionales, marcando un hito importante en la música, el cine y la experimentación sonora [29].

Los primeros sintetizadores modulares, como los creados por Moog y Buchla en la década de 1960, ofrecían un control inigualable hasta el momento sobre la forma de onda y la modulación del sonido [24]. Con el tiempo, la tecnología avanzó hacia sintetizadores compactos y accesibles, como el Yamaha DX7 en los años 80, que popularizó la síntesis FM [5]. Hoy en día, la síntesis digital y el software han vuelto la producción sonora muy accesible, permitiendo a músicos y productores utilizar herramientas avanzadas sin necesidad de costosos equipos físicos.

En este contexto, la síntesis wavetable ha obtenido gran relevancia, ya que permite generar sonidos complejos y dinámicos a partir de tablas de ondas predefinidas, siguiendo una estrategia similar a las lookup tables [28]. Su versatilidad la convierte en una técnica ideal para la creación de timbres complejos, ricos en armónicos.

El proyecto Softwaver forma parte de esta evolución tecnológica, ofreciendo una platafor-

ma accesible y flexible para la síntesis de audio en tiempo real. A través de su diseño modular y su integración con controladores MIDI, busca convertirse en una herramienta clave para músicos, productores y diseñadores de sonido que desean explorar nuevas posibilidades creativas.

1.2. Motivación

La evolución de la tecnología musical ha permitido que el sinteitzador pase de ser un instrumento exclusivo de grandes estudios a ser accesible para cualquier creador sonoro. Sin embargo, muchas soluciones disponibles en el mercado presentan una curva de aprendizaje pronunciada o interfaces poco intuitivas, con un público objetvo cualificado en ingeniería de sonido, lo que dificulta su adopción por parte de músicos y productores emergentes.

El desarrollo de Softwaver responde a la necesidad de una plataforma de síntesis de audio en tiempo real que combine flexibilidad, potencia y facilidad de uso. Su implementación basada en síntesis wavetable permite una exploración sonora rica y detallada, mientras que su interfaz gráfica intuitiva busca facilitar la entrada a usuarios sin experiencia profunda previa en síntesis. Además, la integración con controladores MIDI facilita la interacción en entornos de producción y actuaciones en directo, brindando a los músicos una experiencia de control más expresiva.

Por otro lado, otra motivación importante para la realización de este proyecto es el inicio de una carrera en investigación de sonido y desarrollo de herramientas de audio digital, iniciando después de la finalización del proyecto un programa de máster en tecnologías de acústica y audio en la universidad de Aalto [1].

1.3. Objetivos

El objetivo principal de este proyecto es el diseño e implementación de una aplicación software de síntesis de audio en tiempo real, que permita a los usuarios crear y manipular sonidos de manera flexible y eficiente. Para lograrlo, se plantean los siguientes objetivos específicos:

- Implementar un sistema de síntesis basado en osciladores wavetable, permitiendo la generación de formas de onda complejas y ricas en armónicos.
- Incorporar un generador de ruido para ampliar las posibilidades sonoras y añadir texturas al sonido sintetizado.
- Aplicar filtrado de señales de audio digital mediante filtros paso bajo (LPF), con el fin de modelar la salida del sintetizador, permitiendo un proceso de síntesis substractiva.
- Implementar herramientas de modulación, utilizando envelopes y osciladores de baja frecuencia (LFO) para modificar dinámicamente parámetros como la amplitud y el filtrado a lo largo del tiempo.

- Diseñar una interfaz gráfica de usuario (GUI) intuitiva, que facilite la manipulación de los distintos parámetros de síntesis.
- Permitir el almacenamiento y carga de presets, de modo que los usuarios puedan guardar y recuperar sonidos previamente diseñados.
- Integrar controladores MIDI hardware, posibilitando la interacción con el sintetizador mediante dispositivos físicos para mejorar la experiencia de interpretación.
- Proporcionar una experiencia de usuario sencilla y accesible, sin requerir profundos conocimientos en ingeniería de sonido.

Estos objetivos buscan garantizar que Softwaver se convierta en una herramienta versátil, accesible y potente para músicos, diseñadores de sonido y productores, combinando innovación tecnológica con facilidad de uso.

1.4. Estructura de la memoria

Este documento se organiza en varios capítulos, cada uno abordando distintos aspectos del desarrollo del proyecto:

Capítulo 2 - Planificación: Se detalla la metodología utilizada para la ejecución del proyecto, así cómo la estimación del desarrollo temporal del proyecto y su coste.

Capítulo 3 - Requisitos: Se definen los requisitos del sistema, tanto funcionales como no funcionales y de información.

Capítulo 4 - Estudio de factibilidad Se estudia la viabilidad de la ejecución del proyecto, analizando sus riesgos.

Capítulo 5 - Análisis: Se presentan el modelo de dominio, los casos de uso y los diagramas de actividad que describen el funcionamiento del sistema.

Capítulo 6 - Tecnologías y herramientas utilizadas: Se describen las tecnologías y herramientas empleadas durante el desarrollo del proyecto.

Capítulo 7 - Diseño: Incluye la arquitectura del sistema, el diseño de la base de datos y los bocetos de la interfaz gráfica de usuario.

Capítulo 8 - Implementación: Explica la forma en que se han desarrollado los componentes principales de la aplicación, así como las estrategias adoptadas.

Capítulo 9 - Pruebas: Se documentan las pruebas de aceptación realizadas para evaluar el correcto funcionamiento del sistema.

Capítulo 10 - Despliegue: Se explica la creación de un instalador de la aplicación.

Capítulo 11 - Seguimiento del proyecto: Detalla el desarrollo del proyecto en el tiempo comparando fechas planificadas y fechas reales de ejecución de cada fase del proyecto.

Capítulo 12 - Conclusiones y trabajo futuro: Se ofrece una evaluación general del proyecto, junto con posibles mejoras y futuras líneas de desarrollo.

Bibliografía: Lista de referencias bibliográficas utilizadas en la realización del proyecto.

Anexo A - Manuales: Contiene el manual de mantenimiento, con las instrucciones necesarias para continuar el desarrollo y compilar el proyecto en cualquier sistema.

Anexo B - Enlaces: Contiene enlaces al repositorio del código y a la descarga del instalador de la aplicación.

Capítulo 2

Estado del arte

En el panorama actual de la síntesis de audio digital, existen diversas herramientas avanzadas que ofrecen amplias posibilidades de diseño sonoro. Entre las más destacadas y comercialmente más exitosas se encuentran Xfer Serum, Arturia Pigments y Kilohearts Phase Plant. Estas aplicaciones se han convertido en referentes en la producción musical profesional y la creación sonora experimental, por su potencia, calidad de sonido y versatilidad.

2.1. Xfer Serum

Figura 2.1: Xfer Serum 2 [33]

Xfer Serum [33] es uno de los sintetizadores wavetable más populares del mercado. Su principal fortaleza reside en su motor de síntesis wavetable con editor visual. También incluye un profundo y complejo sistema de modulación. Sin embargo, su complejidad puede suponer una curva de aprendizaje pronunciada para usuarios no expertos.

2.2. Arturia Pigments

Figura 2.2: Arturia Pigments 6 [3]

Arturia Pigments [3] combina múltiples motores de síntesis (wavetable, granular, analógico virtual, aditivo, etc.) con una interfaz visual extremadamente rica. Integra también un sistema de modulación profundo que permite crear sonidos complejos. Pigments destaca por sus capacidades de experimentación sonora, aunque su enfoque más amplio puede resultar excesivo para usuarios que buscan simplicidad y rendimiento en tiempo real.

2.3. Kilohearts Phase Plant

Figura 2.3: Kilohearts Phase plant [20]

Phase Plant [20], desarrollado por Kilohearts, adopta un enfoque modular. Los usuarios pueden construir sus propios sintetizadores añadiendo bloques funcionales como generadores, filtros, efectos y moduladores, todo dentro de un entorno visual flexible. Esta libertad permite diseños sonoros avanzados, aunque también puede abrumar a usuarios principiantes por la ausencia de restricciones.

2.4. Softwaver

Estas herramientas no cumplen un importante objetivo de este proyecto: Proporcionar una experiencia de usuario sencilla y accesible, sin requerir profundos conocimientos en ingeniería de sonido. Frente a ello, Softwaver se posiciona como una alternativa centrada en la síntesis wavetable en tiempo real, con una interfaz clara y orientada a la usabilidad. A diferencia de soluciones comerciales complejas y orientadas a un público profesional, Softwaver busca el equilibrio entre potencia y accesibilidad, con una interfaz intuitiva que permite a músicos y diseñadores sonoros centrarse en la creatividad sin necesidad de conocimientos técnicos avanzados. Además, sus capacidades de modulación avanzada y control MIDI lo sitúan como una opción atractiva para entornos educativos, proyectos creativos o prototipado rápido de ideas sonoras.

Capítulo 3

Planificación

En este capítulo se describe la metodología adoptada para la ejecución del proyecto, el análisis de riesgos asociados y la estimación de costes. Además, se detalla el plan de trabajo seguido, asegurando que cada etapa del desarrollo se realice dentro del tiempo y los recursos disponibles.

3.1. Metodología

Para el desarrollo de este proyecto se ha adoptado una metodología de desarrollo secuencial conocida como *cascada*, en la cual cada fase se completa antes de pasar a la siguiente. Este enfoque es adecuado porque los requisitos del sistema están bien definidos desde el inicio y el alcance del proyecto es claro. [26]

El proceso de desarrollo sigue las siguientes fases:

1. **Especificación de requisitos:** Se definen los requisitos funcionales, no funcionales y de información del sistema. Esta fase es crucial, ya que en un modelo *cascada* no se permiten cambios posteriores.

2. **Estudio de factibilidad:** Se analiza la viabilidad del proyecto desde diferentes perspectivas: técnica, económica y temporal. En este análisis se evalúan los recursos disponibles, las herramientas de desarrollo adecuadas y las limitaciones que puedan surgir en la implementación. Además, se identifican posibles riesgos y se establecen estrategias para mitigarlos.

3. **Análisis:** Se estudian las características del sistema y su relación con los requisitos definidos. En esta fase se identifican los módulos principales, sus interacciones y los datos que deben ser gestionados.

4. **Diseño:** A partir del análisis, se elabora la arquitectura del sistema, definiendo la estructura interna de los módulos de síntesis de audio, la integración de los controladores MIDI y la interfaz gráfica de usuario (GUI). En esta fase se diseñan los diagramas de flujo, UML y esquemas necesarios para la implementación.
5. **Implementación:** Se lleva a cabo la programación del sistema siguiendo la estructura diseñada. Desarrollando cada módulo del sistema de forma completa antes de continuar con el siguiente.
6. **Pruebas y validación:** Se realizan pruebas unitarias y de integración para garantizar que el software cumple con los requisitos establecidos.
7. **Despliegue:** Una vez validadas todas las funcionalidades, el software se prepara para distribución.

3.2. Estimación de costes

El desarrollo de un proyecto de software requiere una planificación detallada de los costos asociados, asegurando que los recursos disponibles sean suficientes para completar cada fase sin comprometer la calidad del producto. En este apartado se presentan los costos estimados del proyecto, incluyendo el coste del personal, herramientas y tecnologías utilizadas, así como otros gastos indirectos que puedan surgir durante el desarrollo.

3.2.1. Coste del personal

El coste del personal representa uno de los factores más significativos en el presupuesto del proyecto, ya que abarca los salarios del equipo de desarrollo, así como posibles gastos adicionales relacionados con formación y capacitación. Para estimar este coste, se ha considerado una duración total de **15 semanas**, ajustando el número de horas trabajadas en función de la dedicación semanal de cada rol.

Tabla 3.1: Estimación del coste del personal

Rol	**Horas semanales**	**Total de horas**	**Coste por hora (€)**	**Coste total (€)**
Desarrollador	25	375	16,53	6.198,75
Diseñador UI/UX	10	150	16,33	2.449,50
Gestor de proyecto	7,5	112,5	26,39	2.964,38
Total				**11.612,63**

Para estimar los costes por hora de cada trabajador, se han utilizado los salarios medios anuales según *Glassdoor*, suponiendo 1800 horas anuales. [9]

3.2.2. Costes de materiales

El desarrollo del proyecto requiere una inversión en materiales esenciales para su ejecución. Los costos materiales incluyen tanto el **hardware** necesario para el desarrollo y pruebas del sistema como el **software** utilizado en la implementación.

Costes hardware

El **hardware** se refiere a los dispositivos utilizados para la programación, depuración y prueba del sistema, como ordenadores, interfaces de audio y controladores MIDI.

Dado que el hardware adquirido para este proyecto no se desechará tras su finalización y será reutilizado en desarrollos futuros, se ha aplicado un criterio de amortización proporcional. Se estima una vida útil de 5 años (aproximadamente 260 semanas) para los equipos, y una duración del proyecto de 15 semanas. Por tanto, se imputa únicamente el 5,77 % del coste total de adquisición, lo cual refleja con mayor precisión el coste real del uso del hardware durante este proyecto.

Tabla 3.2: Estimación de costes de hardware

Hardware	**Cantidad**	**Coste unitario (€)**	**Coste total (€)**	**Coste amortizado (€)**
Ordenador de desarrollo (Lenovo LOQ 15IAX9E)	1	1,200	1,200	69.24
Interfaz de audio externa (MOTU M2)	1	229	229	13.23
Controlador MIDI (Arturia Keylab 49 mk3)	1	429	429	24.76
Monitores de estudio (Adam Audio T7V)	2	185	370	21.35
Auriculares (Beyerdynamic DT-990 Pro)	1	158	158	9.12
Cables y adaptadores varios	1 set	80	80	4.62
Total amortizado				**142.32**

Costes software

El **software** abarca herramientas de desarrollo, bibliotecas especializadas y entornos de trabajo. La única herramienta software a utilizar que supone un coste es el framework JUCE.

Tabla 3.3: Estimación de costes de software

Software	Cantidad	Coste unitario (€)	Coste total (€)
Licencia frameworks y bibliotecas	1	740	740
Total			**740**

Sin embargo, este coste no es de desarrollo sino para obtener una licencia de distribución. En principio, la distribución de la aplicación está fuera del alcance del proyecto, pero quien distribuya esta aplicación debe afrontar este coste.

3.2.3. Costes totales

Los costes totales suponen la suma de los costes descritos anteriormente.

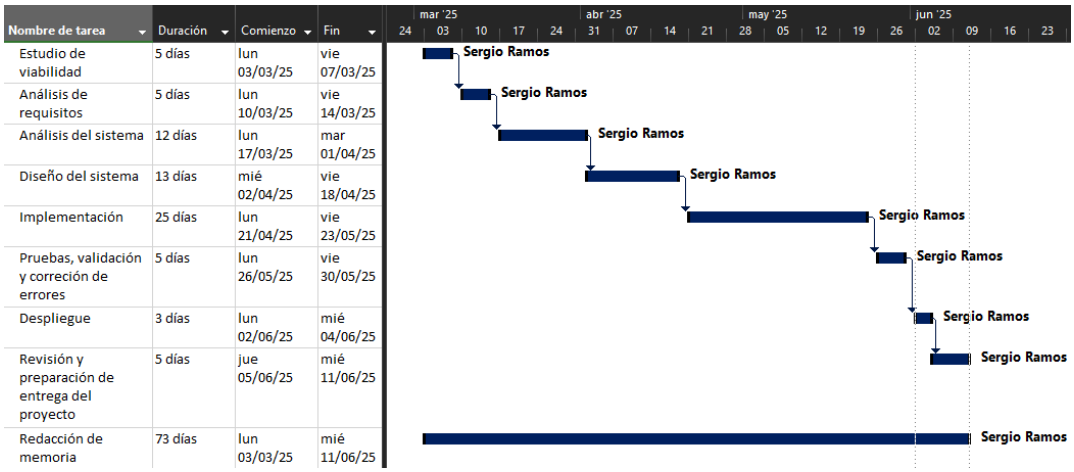
Tabla 3.4: Estimación de costes totales

Tipo de coste	Coste (€)
Personal	11,612.63
Hardware	142.32
Software	740
Total	**12,494.95**

3.3. Plan de trabajo

Para estructurar el desarrollo del proyecto y asegurar su correcta ejecución, se ha dividido el trabajo en ocho fases principales. Cada fase abarca un conjunto de actividades específicas, con el objetivo de garantizar un flujo de trabajo eficiente y una planificación adecuada.
Se ha tomado como fecha de inicio del proyecto el día 3 de febrero de 2025 y como fecha de finalización el 16 de junio del mismo año, suponiendo un total de 15 semanas.
A continuación se muestran las diferentes fases en que se divide el proyecto con sus correspondientes diagramas de Gantt representando el desarrollo del trabajo requerido en cada fase.

3.3.1. Fases del proyecto

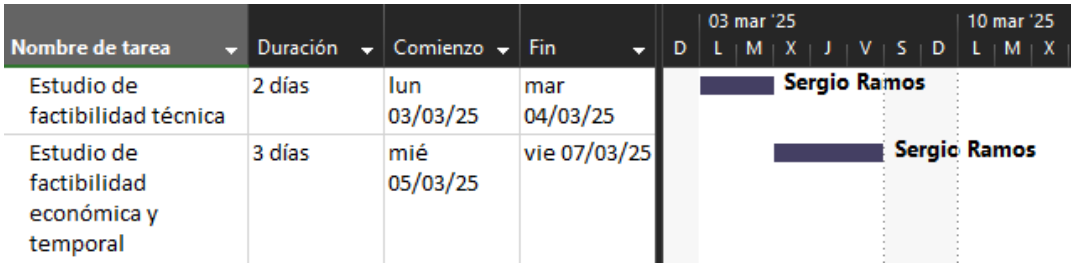


Nombre de tarea	Duración	Comienzo	Fin
Estudio de viabilidad	5 días	lun 03/03/25	vie 07/03/25
Análisis de requisitos	5 días	lun 10/03/25	vie 14/03/25
Análisis del sistema	12 días	lun 17/03/25	mar 01/04/25
Diseño del sistema	13 días	mié 02/04/25	vie 18/04/25
Implementación	25 días	lun 21/04/25	vie 23/05/25
Pruebas, validación y correción de errores	5 días	lun 26/05/25	vie 30/05/25
Despliegue	3 días	lun 02/06/25	mié 04/06/25
Revisión y preparación de entrega del proyecto	5 días	jue 05/06/25	mié 11/06/25
Redacción de memoria	73 días	lun 03/03/25	mié 11/06/25

Figura 3.1: Plan de trabajo: Fases generales

Estudio de factibilidad

Se evalúa la viabilidad del proyecto desde distintas perspectivas: técnica, económica y temporal.



Nombre de tarea	Duración	Comienzo	Fin
Estudio de factibilidad técnica	2 días	lun 03/03/25	mar 04/03/25
Estudio de factibilidad económica y temporal	3 días	mié 05/03/25	vie 07/03/25

Figura 3.2: Plan de trabajo: Estudio de factibilidad

Análisis de requisitos

Se identifican y documentan los requisitos funcionales, no funcionales y de información del sistema.

Nombre de tarea	Duración	Comienzo	Fin
Identificación de requisitos	2 días	lun 10/03/25	mar 11/03/25
Especificación de requisitos	3 días	mié 12/03/25	vie 14/03/25

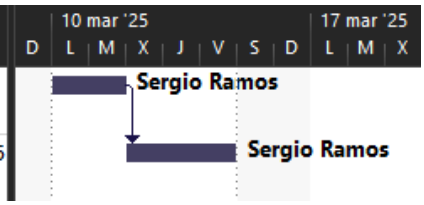


Figura 3.3: Plan de trabajo: Análisis de requisitos

Análisis del sistema

Se estudia la estructura general del sistema, definiendo los módulos principales y sus interacciones.

Nombre de tarea	Duración	Comienzo	Fin
Definición de casos de uso	2 días	lun 17/03/25	mar 18/03/25
Diagramas de casos de uso	6 días	mié 19/03/25	mié 26/03/25
Identifiacación de restricciones	1 día	jue 27/03/25	jue 27/03/25
Modelado del dominio	3 días	vie 28/03/25	mar 01/04/25

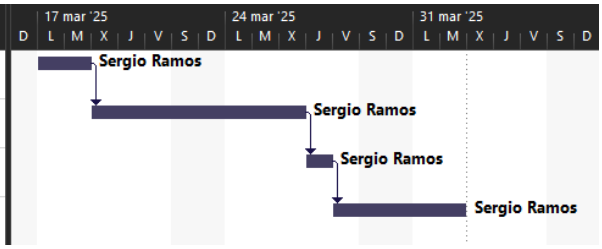


Figura 3.4: Plan de trabajo: Análisis del sistema

Diseño del sistema

Se desarrolla la arquitectura del software, estableciendo la estructura de los módulos y la comunicación entre ellos.

Nombre de tarea	Duración	Comienzo	Fin
Diseño de la arquitectura general del sistema	4 días	mié 02/04/25	lun 07/04/25
Diseño del núcleo de procesamiento de audio	4 días	mar 08/04/25	vie 11/04/25
Diseño de la interfaz gráfica de usuario	3 días	lun 14/04/25	mié 16/04/25
Mockups de la interfaz	2 días	jue 17/04/25	vie 18/04/25

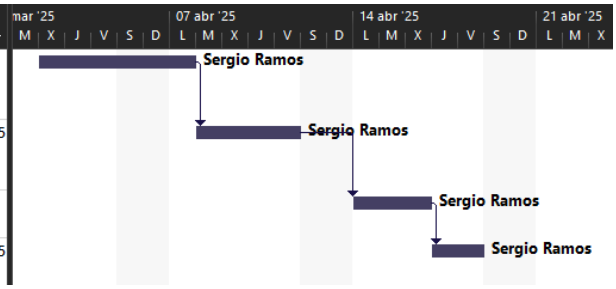


Figura 3.5: Plan de trabajo: Diseño del sistema

Implementación

En esta fase se lleva a cabo la programación del sistema.

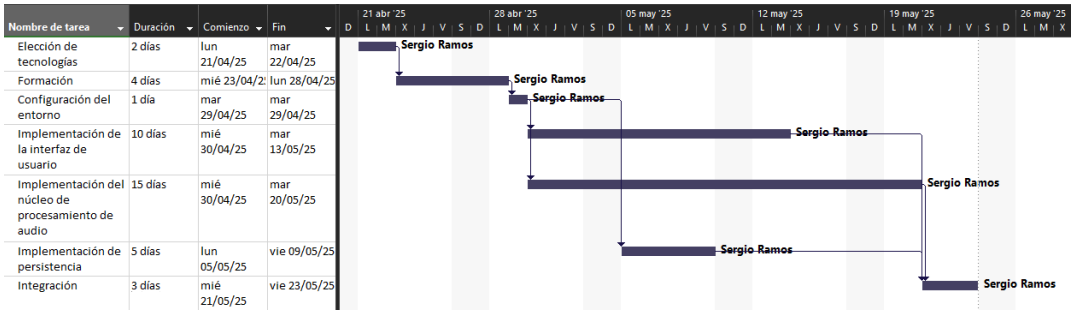


Figura 3.6: Plan de trabajo: Implementación

Pruebas y validación

Se realizan pruebas funcionales y de rendimiento para evaluar la estabilidad y eficiencia del sistema.

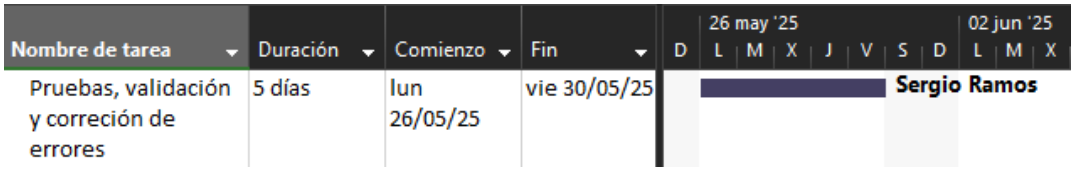


Figura 3.7: Plan de trabajo: Pruebas y validación

Despliegue

Se prepara un instalador de la aplicación.

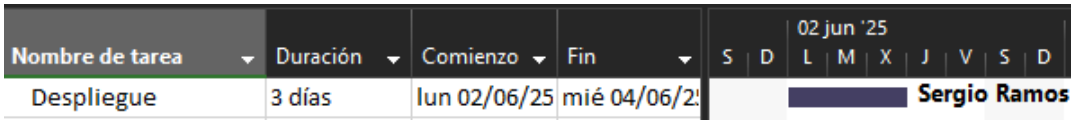


Figura 3.8: Plan de trabajo: Despliegue

Entrega

Se preparan todos los archivos y recursos necesarios para la entrega final del proyecto.

Figura 3.9: Plan de trabajo: Entrega

Capítulo 4

Requisitos

Un requisito en ingeniería de software se define como una condición o capacidad que un sistema debe cumplir para satisfacer una necesidad del usuario o una restricción impuesta por el entorno del desarrollo [10]. Según el IEEE Standard 830-1998, un requisito puede describir funciones específicas del software, características de rendimiento, restricciones de diseño o cualquier otro aspecto esencial para el correcto funcionamiento del sistema.

4.1. Requisitos funcionales

Un requisito funcional describe una función o comportamiento específico que debe cumplir un sistema de software para satisfacer las necesidades del usuario o los objetivos del proyecto. Estos requisitos detallan las interacciones entre el usuario y el sistema, especificando qué debe hacer el software en términos de entradas, procesos y salidas [32]. Los requisitos funcionales expresan las capacidades esenciales del sistema, tales como la autenticación de usuarios, la generación de informes o el procesamiento de datos [32].

4.1.1. Respecto al oscilador

Tabla 4.1: Requisitos funcionales del sistema relativos al oscilador

Código	Requisito Funcional
$RF_O - 0$	El sistema debe incluir un oscilador wavetable.
$RF_O - 1$	El sistema debe permitir al usuario modificar la frecuencia del oscilador en semitonos dentro de un rango de ±24 semitonos.
$RF_O - 2$	El sistema debe permitir al usuario modificar la frecuencia del oscilador en cents dentro de un rango de ±1 semitonos.
$RF_O - 3$	La frecuencia del oscilador en cents debe ser modulable.
$RF_O - 4$	El sistema debe permitir al usuario modificar el volumen del oscilador.
$RF_O - 5$	El volumen del oscilador debe ser modulable.
$RF_O - 6$	La wavetable debe almacenar varias formas de onda.
$RF_O - 7$	El sistema debe permitir al usuario seleccionar la posición de la wavetable (onda a utilizar).
$RF_O - 8$	La posición de la wavetable debe ser modulable.
$RF_O - 9$	El sistema debe permitir al usuario cargar diferentes wavetables en el oscilador.

4.1.2. Respecto al generador de ruido

Tabla 4.2: Requisitos funcionales del sistema relativos al generador de ruido

Código	Requisito Funcional
$RF_R - 0$	El sistema debe incluir un generador de ruido.
$RF_R - 1$	El sistema debe permitir al usuario modificar el volumen del generador de ruido.
$RF_R - 2$	El volumen del generador de ruido debe ser modulable.

4.1.3. Respecto al filtro

Tabla 4.3: Requisitos funcionales del sistema relativos al filtro

Código	Requisito Funcional
$RF_F - 0$	El sistema debe incluir un filtro paso bajo (LPF) que filtre el audio generado.
$RF_F - 1$	El número de polos del filtro paso bajo puede modificarse por el usuario, permitiendo elegir entre 2-polos (12dB/oct) y 4-polos(24dB/oct).
$RF_F - 2$	El sistema debe permitir al usuario modificar la frecuencia de corte del filtro en un rango de 20Hz a 17000Hz.
$RF_F - 3$	La frecuencia de corte del filtro debe ser modulable.
$RF_F - 4$	El sistema debe permitir al usuario modificar el factor de calidad (Q)/resonancia del filtro.
$RF_F - 5$	El factor de calidad (Q)/resonancia del filtro debe ser modulable.

4.1.4. Respecto a los moduladores / fuentes de modulación

Tabla 4.4: Requisitos funcionales del sistema relativos a los moduladores

Código	Requisito Funcional
$RF_M - 0$	El sistema debe permitir al usuario utilizar la dinámica con la que presiona una tecla (velocity) de un controlador MIDI como fuente de modulación para modular cualquier elemento modulable.
$RF_M - 1$	El sistema debe permitir al usuario utilizar la modulation wheel de un controlador MIDI como fuente de modulación para modular cualquier elemento modulable.
$RF_M - 2$	El sistema tendrá un envelope principal que modulará la amplitud de la salida del sistema.
$RF_M - 3$	El envelope principal podrá utilizarse como fuente de modulación para modular cualquier elemento modulable.
$RF_M - 4$	El envelope principal tendrá parámetros Attack, Decay, Sustain, Release modificables por el usuario.
$RF_M - 5$	El sistema tendrá un envelope secundario que podrá utilizarse como fuente de modulación para modular cualquier elemento modulable.
$RF_M - 6$	El envelope secundario tendrá parámetros Attack, Decay, Sustain, Release modificables por el usuario.
$RF_M - 7$	- El sistema tendrá un oscilador de baja frecuencia (LFO) que podrá utilizarse como fuente de modulación para modular cualquier elemento modulable.
$RF_M - 8$	- El sistema debe permitir al usuario modificar la frecuencia de oscilación del LFO en un rango de 0Hz a 20Hz.
$RF_M - 9$	- El sistema debe permitir al usuario seleccionar la forma de onda del LFO entre las siguientes posibilidades: sinusoidal, triangular, cuadrada, diente de sierra.

4.1.5. Respecto a la entrada de interpretación musical

Tabla 4.5: Requisitos funcionales del sistema relativos a la entrada de interpretación musical

Código	Requisito Funcional
$RF_E - 0$	El sistema debe permitir seleccionar el canal MIDI donde recibir mensajes MIDI de controladores externos.
$RF_E - 1$	El sistema debe interpretar los mensajes MIDI recibidos y generar las notas correspondientes en el sistema de afinación temperado de 12 tonos iguales.
$RF_E - 2$	El sistema debe interpretar los mensajes MIDI recibidos y generar las modulaciones correspondientes.

4.1.6. Respecto a funcionalidades globales

Tabla 4.6: Requisitos funcionales del sistema relativos a funcionalidades globales

Código	Requisito Funcional
$RF_G - 0$	El sistema debe permitir al usuario modificar el nivel (volumen) de la salida del sistema.
$RF_G - 0$	El sistema debe permitir al usuario reestablecer el valor por defecto facilmente de cada parámetro individual.

4.2. Requisitos no funcionales

Un requisito no funcional especifica características y restricciones que afectan el rendimiento, la usabilidad y otros atributos de calidad de un sistema, pero no describen funciones específicas. Según el IEEE Standard 830-1998, estos requisitos establecen criterios como eficiencia, escalabilidad, seguridad y compatibilidad, los cuales influyen en la experiencia del usuario y en la arquitectura del software [10].

Tabla 4.7: Requisitos no funcionales del sistema

Código	Requisito No Funcional
RNF-0	La latencia entre la pulsación de una tecla y la generación del sonido debe ser mínima.
RNF-1	El sistema debe ofrecer una polifonía de hasta 8 voces.
RNF-2	El sistema utilizará la frecuencia de muestreo y el tamaño de buffer del dispositivo de audio del host.

4.3. Requisitos de información

Los requisitos de información especifican los datos que el sistema debe almacenar, procesar o intercambiar para cumplir con su funcionalidad. Estos requisitos incluyen el tipo de información gestionada, su estructura, persistencia y acceso. Los requisitos de información describen cómo los datos deben ser organizados y gestionados dentro del sistema para garantizar su integridad y disponibilidad [31].

Tabla 4.8: Requisitos de información del sistema

Código	Requisito de Información
RI-1	El usuario puede guardar presets de la configuración del sistema (sonido diseñado).
RI-2	El usuario puede cargar presets previamente guardados.

Capítulo 5

Estudio de viabilidad

Antes de proceder con el desarrollo del sistema, es fundamental evaluar su viabilidad en distintos aspectos. El estudio de viabilidad permite determinar si el proyecto puede llevarse a cabo con los recursos disponibles y dentro de las limitaciones establecidas. Para ello, se analizarán los factores técnicos, económicos y temporales, identificando posibles riesgos y estrategias para mitigarlos.

5.1. Viabilidad técnica

El sistema a desarrollar implica la implementación de un sintetizador de audio en tiempo real, lo que requiere una serie de recursos tecnológicos que permitan su correcto funcionamiento. A continuación, se analizan los aspectos clave que determinan la viabilidad técnica del proyecto:

- **Procesamiento de audio en tiempo real:** Se debe garantizar una latencia mínima en la generación del sonido, evitando retardos perceptibles por el usuario.
- **Generación de sonido por síntesis digital:** Se requiere la implementación de osciladores, filtros y sistemas de modulación para la manipulación del sonido.
- **Interfaz gráfica interactiva:** La aplicación debe contar con una GUI que permita modificar los parámetros del sintetizador de manera intuitiva y eficiente.
- **Compatibilidad con controladores MIDI:** El sistema debe ser capaz de recibir e interpretar mensajes MIDI para la ejecución y modulación de sonidos.
- **Eficiencia computacional:** Es fundamental optimizar el uso de CPU y memoria, especialmente si se desea ejecutar el software en equipos con recursos limitados.

El proyecto es factible desde un punto de vista técnico, siempre que se utilicen herramientas adecuadas para el procesamiento de audio y se optimicen los algoritmos para minimizar el consumo de recursos.

5.2. Viabilidad económica

El coste estimado del proyecto asciende a **19,518.50 €**, considerando los siguientes aspectos:

- **Coste del personal:** Se estima una inversión en el desarrollo, diseño e implementación del sistema.
- **Coste de hardware:** Se requiere un ordenador con suficiente capacidad de procesamiento, una interfaz de audio y un controlador MIDI para pruebas.
- **Coste de software:** Incluye la adquisición de herramientas de desarrollo y bibliotecas especializadas para la síntesis de audio.

Dado que los costes son razonables y están dentro del presupuesto disponible, el proyecto es económicamente viable.

5.3. Viabilidad temporal

La duración del proyecto se ha estimado en **15 semanas**, distribuidas en distintas fases de desarrollo. A continuación, se presenta un resumen de la estimación del tiempo requerido para cada fase realizada en el capítulo anterior:

- **Estudio de viabilidad:** 1 semana.
- **Análisis de requisitos:** 1 semana.
- **Análisis del sistema:** 2 semanas.
- **Diseño del sistema:** 3 semanas.
- **Implementación:** 5 semanas.
- **Pruebas y validación:** 1 semana.
- **Despliegue:** 1 semana.
- **Entrega:** 1 semana.

El tiempo estimado es suficiente para completar el desarrollo del sistema, siempre que se mantenga una planificación adecuada y se optimicen los tiempos de implementación.

5.4. Análisis de riesgos

Un riesgo se define como un evento o condición incierta que, de materializarse, puede afectar de manera negativa el desarrollo del proyecto, impactando su alcance, costos, tiempo o calidad [27]. En gestión de proyectos, la identificación, análisis y mitigación de riesgos es fundamental para minimizar problemas y garantizar el éxito del desarrollo. Un riesgo tiene asociados una probabilidad de suceder y un impacto sobre el desarrollo del proyecto, resumidos en las tablas 5.1 y 5.2 respectivamente.

Tabla 5.1: Clasificación de la probabilidad de ocurrencia de un riesgo

Nivel	**Descripción**
Muy Alta	El riesgo es casi seguro de ocurrir. La probabilidad de que ocurra es superior al 50 %.
Alta	Existe una gran posibilidad de que el riesgo ocurra, con una probabilidad entre el 25 % y el 50 %.
Moderada	El riesgo puede ocurrir, pero no es seguro. Su probabilidad varía entre el 10 % y el 25 %.
Baja	Es poco probable que el riesgo ocurra. Su probabilidad es inferior al 10 %.

Tabla 5.2: Clasificación del impacto de un riesgo

Nivel	**Descripción**
Muy Alto	El riesgo tiene un impacto crítico en el proyecto, causando fallos graves, retrasos significativos o costes excesivos. Puede comprometer la viabilidad del proyecto.
Alto	El riesgo afecta negativamente el proyecto, generando retrasos importantes, incremento de costes o reducción en la calidad del producto.
Moderado	El riesgo puede causar inconvenientes en el proyecto, como pequeños retrasos, ajustes en el presupuesto o necesidad de esfuerzos adicionales.
Bajo	El riesgo tiene un impacto menor, con efectos poco significativos en el desarrollo del proyecto. Puede ser gestionado fácilmente sin afectar los objetivos principales.

5.4.1. Identificación de riesgos

Para este proyecto, se han identificado los siguientes riesgos y sus correspondientes probabilidades, impactos, acciones de mitigación y acciones de corrección.

Tabla 5.3: R01 - Errores en la implementación del oscilador wavetable

ID	**R01**
Descripción	Errores en la implementación del oscilador wavetable que afecten la calidad del sonido.
Probabilidad	Moderada
Impacto	Alto
Acciones de mitigación	▪ Definir pruebas unitarias para la correcta generación de formas de onda. ▪ Utilizar referencias académicas sobre síntesis wavetable. ▪ Implementar prototipos antes del desarrollo final.
Acciones de corrección	▪ Revisar la implementación y corregir errores en el código. ▪ Comparar los resultados con software de síntesis de referencia. ▪ Optimizar el algoritmo para mejorar la fidelidad del sonido.

Tabla 5.4: R02 - Problemas de compatibilidad con controladores MIDI

ID	**R02**
Descripción	Problemas de compatibilidad con controladores MIDI.
Probabilidad	Moderada
Impacto	Moderado
Acciones de mitigación	▪ Usar estándares MIDI ampliamente compatibles (MIDI 1.0 y MIDI 2.0). ▪ Probar con múltiples controladores MIDI de diferentes fabricantes. ▪ Incluir documentación sobre los controladores soportados. ▪ Incluir una herramienta de monitoreo de datos MIDI en la interfaz.
Acciones de corrección	▪ Implementar soporte adicional para controladores incompatibles. ▪ Ofrecer configuraciones manuales en la interfaz para ajustes de compatibilidad. ▪ Actualizar el software para corregir errores detectados en la recepción de mensajes MIDI.

Tabla 5.5: R03 - Latencia alta en la generación del sonido

ID	**R03**
Descripción	Latencia alta en la generación del sonido.
Probabilidad	Alta
Impacto	Muy Alto
Acciones de mitigación	▪ Optimizar la gestión del buffer de audio para minimizar la latencia. ▪ Usar bibliotecas optimizadas para procesamiento de audio en tiempo real. ▪ Probar en distintos sistemas y ajustar la configuración del motor de audio.
Acciones de corrección	▪ Reducir el tamaño del buffer en tiempo real si la latencia es alta. ▪ Optimizar el código eliminando procesos innecesarios en la cadena de audio. ▪ Permitir a los usuarios ajustar la configuración de latencia en la interfaz.

Tabla 5.6: R04 - Consumo excesivo de CPU, afectando el rendimiento en tiempo real

ID	**R04**
Descripción	Consumo excesivo de CPU, afectando el rendimiento en tiempo real.
Probabilidad	Moderada
Impacto	Alto
Acciones de mitigación	▪ Optimizar el código para minimizar el uso de recursos del sistema. ▪ Implementar técnicas de procesamiento en paralelo si es posible. ▪ Usar bibliotecas eficientes para síntesis de audio.
Acciones de corrección	▪ Permitir ajustes en la calidad del procesamiento de audio para balancear rendimiento y calidad. ▪ Identificar y corregir cuellos de botella en el código. ▪ Sugerir requisitos mínimos del sistema en la documentación.

Tabla 5.7: R05 - Dificultad en la integración de la interfaz gráfica

ID	**R06**
Descripción	Dificultad en la integración de la interfaz gráfica.
Probabilidad	Moderada
Impacto	Moderado
Acciones de mitigación	▪ Diseñar la arquitectura del sistema de manera modular y bien documentada. ▪ Usar frameworks y bibliotecas gráficas compatibles con el sistema de síntesis. ▪ Realizar pruebas de integración desde las primeras fases del desarrollo.
Acciones de corrección	▪ Optimizar la comunicación entre la interfaz y los módulos de audio.

Tabla 5.8: R06 - Fallos en la carga y almacenamiento de presets

ID	**R06**
Descripción	Fallos en la carga y almacenamiento de presets.
Probabilidad	Baja
Impacto	Moderado
Acciones de mitigación	▪ Implementar pruebas unitarias para la gestión de presets. ▪ Usar un formato de archivo estándar y bien documentado (ej. JSON, XML).
Acciones de corrección	▪ Revisar y corregir errores en la lectura y escritura de presets. ▪ Documentar los posibles errores y soluciones en el manual del usuario.

Tabla 5.9: R07 - Dificultad para ajustar correctamente los parámetros de algún módulo

ID	**R07**
Descripción	Dificultad para ajustar correctamente los parámetros de algún módulo.
Probabilidad	Moderada
Impacto	Moderado
Acciones de mitigación	▪ Proporcionar controles intuitivos y bien organizados en la interfaz gráfica. ▪ Proporcionar feedback visual a la interacción del usuario con el sistema. ▪ Incluir documentación y ejemplos sobre el uso de los distintos módulos.
Acciones de corrección	▪ Ajustar la sensibilidad de los controles. ▪ Reubicar los controles en la interfaz. ▪ Cambiar el tipo de control (rotatorio, lineal, logarítmico). ▪ Realizar pruebas con usuarios para mejorar la experiencia de ajuste.

Tabla 5.10: R08 - Errores en la modulación de parámetros por LFOs y envelopes

ID	**R08**
Descripción	Errores en la modulación de parámetros por LFOs y envelopes.
Probabilidad	Moderada
Impacto	Alto
Acciones de mitigación	▪ Implementar pruebas automatizadas para verificar el correcto funcionamiento de la modulación. ▪ Usar una estructura de datos eficiente para manejar las modulaciones en tiempo real. ▪ Documentar detalladamente el comportamiento esperado de cada tipo de modulación.
Acciones de corrección	▪ Revisar y corregir errores en el uso de modulaciones. ▪ Optimizar el rendimiento para evitar artefactos en la modulación en tiempo real.

Tabla 5.11: R09 - Estimación de tiempo errónea

ID	**R09**
Descripción	La planificación del proyecto puede subestimar o sobrestimar el tiempo necesario para completar ciertas tareas, lo que podría generar retrasos o trabajo apresurado.
Probabilidad	Alta
Impacto	Alto
Acciones de mitigación	▪ Dividir el proyecto en tareas pequeñas con tiempos bien definidos. ▪ Incluir margen de error en la planificación para cubrir posibles imprevistos.
Acciones de corrección	▪ Reajustar el cronograma conforme se avanza en el desarrollo. ▪ Priorizar las funcionalidades críticas si hay retrasos. ▪ Redistribuir tareas y optimizar la asignación de recursos si es necesario.

Tabla 5.12: R10 - Falta de conocimiento de las herramientas de desarrollo

ID	**R10**
Descripción	La falta de experiencia en el uso de ciertas herramientas o lenguajes de programación puede ralentizar el desarrollo y generar errores.
Probabilidad	Moderada
Impacto	Alto
Acciones de mitigación	▪ Realizar formación previa sobre las herramientas clave del proyecto. ▪ Consultar documentación y recursos oficiales antes de empezar la implementación.
Acciones de corrección	▪ Buscar soluciones en foros y comunidades de desarrollo. ▪ Ajustar el desarrollo para utilizar enfoques más familiares si es posible. ▪ Considerar la integración de herramientas alternativas con una curva de aprendizaje más baja.

Tabla 5.13: R11 - Sobrecarga de trabajo del desarrollador.

ID	**R11**
Descripción	El desarrollador asume múltiples responsabilidades en el proyecto, lo que puede generar una carga de trabajo excesiva, afectando la productividad y la calidad del software.
Probabilidad	Alta
Impacto	Alto
Acciones de mitigación	▪ Definir un cronograma de trabajo con tareas bien distribuidas. ▪ Priorizar las funcionalidades esenciales en la primera versión del software. ▪ Aplicar técnicas de gestión del tiempo y productividad. ▪ Evaluar la posibilidad de delegar tareas secundarias.
Acciones de corrección	▪ Redistribuir la carga de trabajo si se detecta sobrecarga en fases críticas. ▪ Ajustar la planificación del proyecto para evitar retrasos innecesarios. ▪ Evitar la implementación de características adicionales no previstas en los requisitos iniciales.

El análisis de riesgos permite anticipar posibles problemas y establecer soluciones preventivas para evitar retrasos o fallos en el desarrollo.

5.5. Conclusión

Tras el análisis de viabilidad, se concluye que el proyecto es técnicamente viable, económicamente sostenible y factible en términos de tiempo. Sin embargo, es fundamental realizar una gestión eficiente del desarrollo, evitando sobrecargar al equipo y optimizando los recursos disponibles. La correcta planificación y mitigación de riesgos serán clave para el éxito del proyecto.

Capítulo 6

Análisis

6.1. Introducción

El objetivo de este capítulo es analizar en detalle el sistema, definiendo su estructura, comportamiento y principales interacciones. Para ello, se utilizará la técnica de *Use-Case Analysis*, que permite identificar los requisitos desde la perspectiva del usuario y modelar los elementos clave del sistema.

Este análisis servirá como base para el diseño e implementación, asegurando que el software cumpla con los requisitos establecidos en los capítulos anteriores. Se describirán los actores que interactúan con el sistema, los casos de uso principales, las clases identificadas y sus responsabilidades, así como la relación entre ellas.

6.2. Realización del Sistema

La realización del sistema es el primer paso dentro del análisis del software y tiene como objetivo establecer una visión general del mismo. En esta sección se identifican los elementos clave que conforman el sistema, incluyendo los actores involucrados, los procesos principales y la información relevante.

Este análisis inicial permitirá definir la estructura básica del sistema y sentará las bases para la posterior especificación detallada de los casos de uso, las clases y sus interacciones.

6.2.1. Identificación de Actores

Un actor es cualquier entidad externa que interactúa con el sistema para realizar una tarea específica. Los actores pueden ser usuarios humanos o sistemas externos que envían o

reciben información del software. En el caso de este proyecto, se han identificado los siguientes actores principales:

- **Usuario:** Es el actor principal del sistema. Manipula los parámetros del sintetizador mediante la interfaz gráfica, ajustando valores como la frecuencia de los osciladores, la resonancia del filtro y la cantidad de modulación. También puede guardar y cargar presets.
- **Controlador MIDI:** Dispositivo externo que envía señales MIDI al sintetizador. Estas señales pueden representar notas musicales, cambios de controladores (como la rueda de modulación) y otros eventos que afectan el sonido generado.
- **Sistema de Audio:** Representa el hardware y software encargado de la reproducción del sonido generado por el sintetizador. Incluye la salida de audio del ordenador y la interfaz de audio utilizada. El sistema se comunica con él a través del buffer de audio.

6.2.2. Procesos Principales del Sistema

El sistema puede descomponerse en una serie de procesos clave, que representan sus principales funcionalidades:

- **Generación de sonido en tiempo real:** El motor de síntesis produce ondas de audio utilizando osciladores wavetable y un generador de ruido. Estas señales son procesadas por filtros y moduladores antes de ser enviadas a la salida de audio.
- **Modulación de parámetros:** Diferentes elementos del sintetizador pueden ser modulados en tiempo real por fuentes como envelopes, LFOs y datos MIDI. Esto permite generar variaciones dinámicas en el sonido.
- **Interacción con el usuario:** La interfaz gráfica permite al usuario ajustar los parámetros del sintetizador de manera visual e intuitiva. También se proporciona retroalimentación gráfica sobre la señal generada y los valores de los parámetros.
- **Gestión de presets:** Los usuarios pueden guardar configuraciones personalizadas del sintetizador y recuperarlas posteriormente. Esto permite reutilizar sonidos diseñados previamente sin necesidad de reajustar manualmente los parámetros.
- **Procesamiento de entrada MIDI:** El sistema recibe mensajes MIDI en tiempo real, los interpreta y los traduce en comandos para la generación y modulación del sonido.

6.2.3. Flujo de Datos en el Sistema

El sistema procesa información proveniente de dos fuentes principales: los mensajes MIDI enviados desde un controlador externo y los ajustes realizados por el usuario en la interfaz gráfica. A continuación, se describe el flujo de datos desde la entrada hasta la salida del sistema:

1. **Entrada desde un controlador MIDI:**
   - El controlador MIDI envía un mensaje al sistema (ejemplo: se ha presionado una tecla o se ha girado un potenciómetro).
   - El mensaje es interpretado por el sistema y se actualizan los parámetros correspondientes.
2. **Entrada desde la interfaz gráfica:**
   - El usuario ajusta un parámetro del sintetizador (ejemplo: aumenta la frecuencia de corte del filtro).
   - La interfaz gráfica envía el nuevo valor al motor de síntesis.
3. **Procesamiento de la señal:**
   - El motor de síntesis recalcula la señal generada considerando los parámetros actuales.
4. **Salida de audio:**
   - La señal procesada se escribe en el buffer de audio.
5. **Gestión de presets:**
   - Si el usuario decide guardar un preset, los valores actuales de los parámetros se almacenan en un archivo de configuración.

6.2.4. Diagrama General del Sistema

A continuación, se presenta un diagrama de alto nivel que ilustra la relación entre los actores y los procesos internos del sistema. Este esquema proporciona una visión general de cómo los distintos elementos interactúan para generar y manipular el sonido.

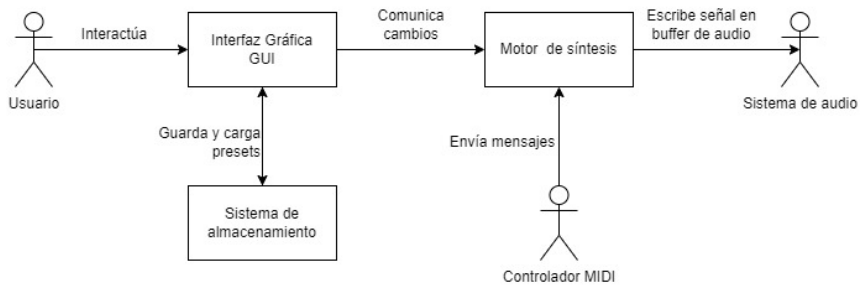


Figura 6.1: Diagrama general del sistema.

6.2.5. Conclusión

La realización del sistema ha permitido establecer los elementos clave que lo conforman, incluyendo los actores externos, los procesos principales y el flujo de datos. Esta información servirá como base para la especificación de casos de uso y la definición detallada de las clases del sistema en las siguientes secciones.

6.3. Descripción del Sistema

En esta sección se describen los comportamientos del sistema visibles al usuario, detallando los casos de uso principales y los diagramas de casos de uso.

6.3.1. Identificación de Casos de Uso

Se han identificado los siguientes casos de uso principales, que han sido agrupados en *Configuracion del sintetizador*, *Modulación y control*, *MIDI*, *Gestión de presets*, *Configuración avanzada*:

Configuración del sintetizador

Tabla 6.1: Caso de Uso: CU-1 Ajustar la frecuencia del oscilador

CU-1	**Ajustar la frecuencia del oscilador**
Actor Principal	Usuario
Descripción	El usuario modifica la desviación de la frecuencia del oscilador en semitonos.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.

Continúa desde la página anterior

CU-1	**Ajustar la frecuencia del oscilador**
Secuencia Normal	1. El usuario ajusta la desviación de frecuencia en semitonos del oscilador mediante un control deslizante en la interfaz gráfica. 2. El sistema actualiza la desviación de frecuencia en semitonos del oscilador en tiempo real. 3. La nueva desviación de frecuencia en semitonos se refleja visualmente en la interfaz gráfica. 4. El usuario ajusta la desviación de frecuencia en cents del oscilador mediante un control deslizante en la interfaz gráfica. 5. El sistema actualiza la desviación de frecuencia en cents del oscilador en tiempo real. 6. La nueva desviación de frecuencia en cents se refleja visualmente en la interfaz gráfica.
Secuencia Alternativa	1a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la frecuencia en semitonos al valor máximo o mínimo permitido. 2a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la frecuencia en cents al valor máximo o mínimo permitido.
Postcondiciones	▪ La frecuencia del oscilador ha sido actualizada correctamente. ▪ El sonido generado refleja la nueva frecuencia seleccionada. ▪ La interfaz muestra la nueva frecuencia seleccionada.

Tabla 6.2: Caso de Uso: CU-2 Seleccionar forma de onda

CU-2	**Seleccionar forma de onda**
Actor Principal	Usuario
Descripción	El usuario elige una forma de onda para ser generada por el oscilador seleccionando una wavetable y la posición de lectura de la wavetable.

Continúa desde la página anterior

CU-2	**Seleccionar forma de onda**
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario solicita un listado de las wavetables disponibles a través de la interfaz gráfica. 2. El sistema muestra el listado de las wavetables disponibles a través de la interfaz gráfica. 3. El usuario selecciona una wavetable del listado. 4. El sistema actualiza la wavetable de la que leer formas de onda para el oscilador. 5. El sistema actualiza la forma de onda del oscilador. 6. La nueva wavetable y la nueva forma de onda se reflejan visualmente en la interfaz gráfica. 7. El usuario ajusta la posición de la wavetable mediante un control deslizante en la interfaz gráfica. 8. El sistema actualiza la forma de onda del oscilador en tiempo real. 9. La nueva forma de onda se refleja visualmente en la interfaz gráfica.
Secuencia Alternativa	3a Si el usuario cierra el listado de wavetables sin seleccionar una, el sistema mantiene la wavetable previamente seleccionada. 7a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la posición de la wavetable al valor máximo o mínimo permitido.

Continúa desde la página anterior

CU-2	**Seleccionar forma de onda**
Postcondiciones	▪ La wavetable ha sido actualizada correctamente. ▪ La posición de la wavetable ha sido actualizada correctamente. ▪ La forma de onda del oscilador ha sido actualizada correctamente.u ▪ El sonido generado refleja la nueva forma de onda. ▪ La interfaz muestra la nueva wavetable seleccionada. ▪ La interfaz muestra la nueva posición de la wavetable seleccionada. ▪ La interfaz muestra la nueva forma de onda.

Tabla 6.3: Caso de Uso: CU-3 Ajustar amplitud del oscilador

CU-3	**Ajustar amplitud del oscilador**
Actor Principal	Usuario
Descripción	El usuario modifica la amplitud de la onda generada por el oscilador.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario ajusta la amplitud del oscilador mediante un control deslizante en la interfaz gráfica. 2. El sistema actualiza la amplitud del oscilador en tiempo real. 3. La nueva amplitud se refleja visualmente en la interfaz gráfica.
Secuencia Alternativa	1a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la amplitud al valor máximo o mínimo permitido.

Continúa desde la página anterior

CU-3	**Ajustar amplitud del oscilador**
Postcondiciones	▪ La amplitud del oscilador ha sido actualizada correctamente. ▪ El sonido generado refleja la nueva amplitud. ▪ La interfaz muestra la nueva amplitud seleccionada.

Tabla 6.4: Caso de Uso: CU-4 Ajustar el filtro

CU-4	**Ajustar el filtro**
Actor Principal	Usuario
Descripción	El usuario modifica los parámetros del filtro paso bajo.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario selecciona el número de polos del filtro. 2. El sistema actualiza el número de polos del filtro. 3. El nuevo número de polos del filtro se refleja visualmente en la interfaz gráfica. 4. El usuario ajusta la frecuencia de corte del filtro mediante un control deslizante en la interfaz gráfica. 5. El sistema actualiza la frecuencia de corte del filtro en tiempo real. 6. La nueva frecuencia de corte del filtro se refleja visualmente en la interfaz gráfica. 7. El usuario ajusta la resonancia del filtro mediante un control deslizante en la interfaz gráfica. 8. El sistema actualiza la resonancia del filtro en tiempo real. 9. La nueva resonancia del filtro se refleja visualmente en la interfaz gráfica.

Continúa desde la página anterior

CU-4	**Ajustar el filtro**
Secuencia Alternativa	4a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la frecuencia de corte al valor máximo o mínimo permitido. 7a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la resonancia al valor máximo o mínimo permitido.
Postcondiciones	▪ El número de polos del filtro ha sido actualizado correctamente. ▪ La frecuencia de corte del filtro ha sido actualizada correctamente. ▪ La resonancia del filtro ha sido actualizada correctamente. ▪ El sonido generado refleja la nueva configuración del filtro. ▪ La interfaz muestra la nueva configuración del filtro.

Tabla 6.5: Caso de Uso: CU-5 Ajustar la amplitud del ruido

CU-5	**Ajustar la amplitud del ruido**
Actor Principal	Usuario
Descripción	El usuario modifica la amplitud de la señal del generador de ruido.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario ajusta la amplitud del ruido mediante un control deslizante en la interfaz gráfica. 2. El sistema actualiza la amplitud del ruido en tiempo real. 3. La nueva amplitud del ruido se refleja visualmente en la interfaz gráfica.

Continúa desde la página anterior

CU-5	**Ajustar la amplitud del ruido**
Secuencia Alternativa	1a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la amplitud al valor máximo o mínimo permitido.
Postcondiciones	▪ La amplitud de la señal del generador de ruido ha sido actualizada correctamente. ▪ El sonido generado refleja la nueva amplitud del ruido. ▪ La interfaz muestra la nueva amplitud del ruido.

Tabla 6.6: Caso de Uso: CU-6 Ajustar la amplitud de la salida del sistema

CU-6	**Ajustar la amplitud de la salida del sistema**
Actor Principal	Usuario
Descripción	El usuario modifica la amplitud de la señal de salida final del sistema.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario ajusta la amplitud de la salida del sistema mediante un control deslizante en la interfaz gráfica. 2. El sistema actualiza la amplitud de su salida en tiempo real. 3. La nueva amplitud de la salida del sistema se refleja visualmente en la interfaz gráfica.
Secuencia Alternativa	1a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la amplitud al valor máximo o mínimo permitido.

Continúa desde la página anterior

CU-6	**Ajustar la amplitud de la salida del sistema**
Postcondiciones	▪ La amplitud de la salida del sistema ha sido actualizada correctamente. ▪ El sonido generado refleja la nueva amplitud de la salida del sistema. ▪ La interfaz muestra la nueva amplitud de la salida del sistema.

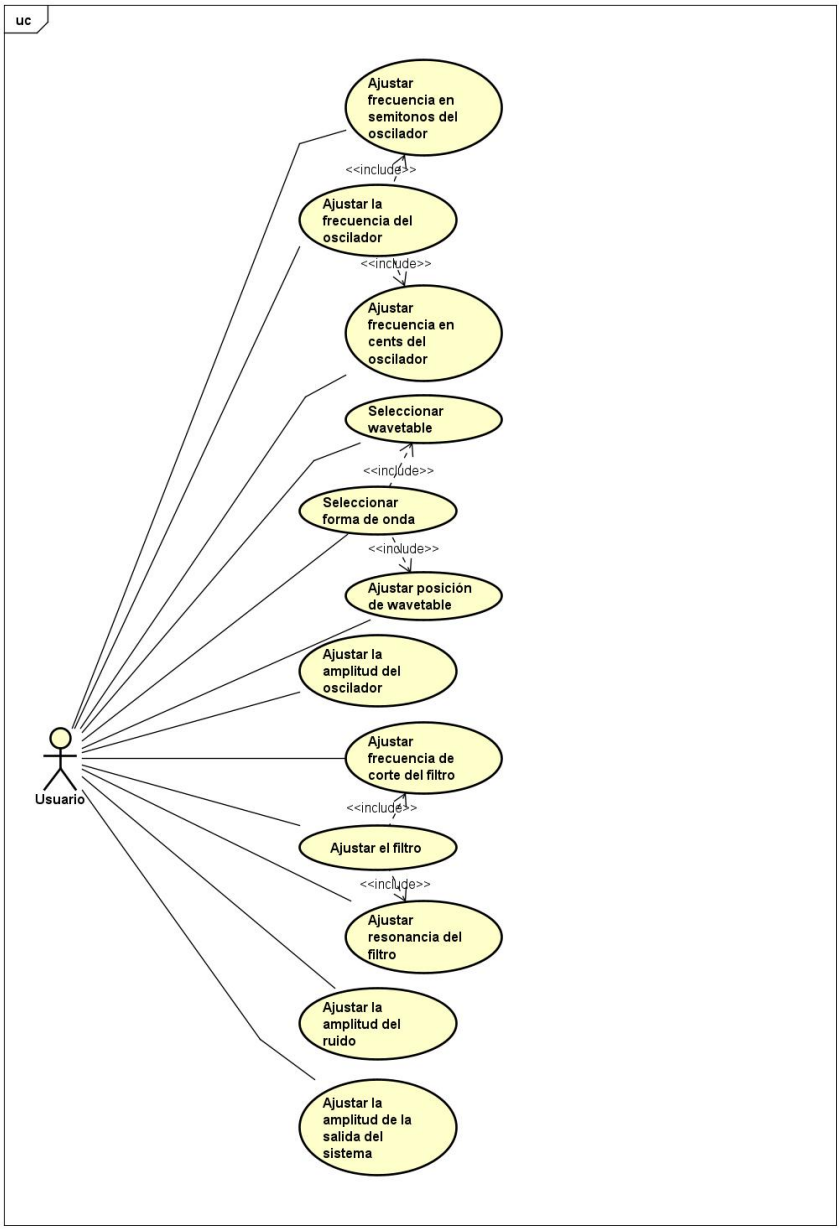


Figura 6.2: Diagrama de casos de uso: Configuración de generación de sonido

Modulación

Tabla 6.7: Caso de Uso: CU-7 Asignar un modulador a un parámetro

CU-7	**Asignar un modulador a un parámetro**
Actor Principal	Usuario
Descripción	El usuario asigna una fuente de modulación a un parámetro.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario solicita un listado de los moduladores. 2. El sistema muestra el listado de los moduladores. 3. El usuario selecciona un modulador de la lista. 4. El sistema ajusta internamente la asignación del modulador 5. El sistema refleja gráficamente la elección del usuario. 6. El usuario solicita un listado de los parámetros modulables. 7. El sistema muestra el listado de los parámetros modulables. 8. El usuario selecciona un parámetro modulable de la lista. 9. El sistema ajusta internamente la asignación del parámetro modulable. 10. El sistema refleja gráficamente la elección del usuario. 11. El usuario ajusta la cantidad de modulación mediante un control deslizante en la interfaz gráfica. 12. El sistema actualiza la modulación del parámetro en tiempo real. 13. La nueva cantidad de modulación se refleja visualmente en la interfaz gráfica.

Continúa desde la página anterior

CU-7	**Asignar un modulador a un parámetro**
Secuencia Alternativa	3a Si el usuario cierra el listado de moduladores sin seleccionar uno, el sistema mantiene la configuración anterior. 8a Si el usuario cierra el listado de parámetros modulables sin seleccionar uno, el sistema mantiene la configuración anterior. 10a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la cantidad de modulación al valor máximo o mínimo permitido.
Postcondiciones	▪ El parámetro seleccionado está siendo modulado por la fuente de modulación asignada con la cantidad especificada. ▪ El sonido generado refleja la nueva modulación. ▪ La interfaz muestra la nueva asociación entre modulador, parámetro modulable y cantidad de modulación.

Tabla 6.8: Caso de Uso: CU-8 Ajustar los parámetros del LFO

CU-8	**Ajustar los parámetros del LFO**
Actor Principal	Usuario
Descripción	El usuario ajusta la forma de onda y frecuencia del LFO.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.

Continúa desde la página anterior

CU-8	**Ajustar los parámetros del LFO**
Secuencia Normal	1. El usuario selecciona la forma de onda para el LFO. 2. El sistema actualiza la forma de onda del LFO. 3. La nueva forma de onda del LFO se refleja visualmente en la interfaz gráfica. 4. El usuario ajusta la frecuencia del LFO mediante un control deslizante en la interfaz gráfica. 5. El sistema actualiza la frecuencia del LFO en tiempo real. 6. La nueva frecuencia del LFO se refleja visualmente en la interfaz gráfica.
Secuencia Alternativa	4a Si el usuario introduce un valor fuera del rango permitido, el sistema limita la frecuencia del LFO al valor máximo o mínimo permitido.
Postcondiciones	▪ La forma de onda del LFO es la seleccionada por el usuario. ▪ La frecuencia del LFO es la ajustada por el usuario. ▪ El sonido generado refleja los nuevos parámetros del LFO. ▪ La interfaz muestra los nuevos parámetros del LFO.

Tabla 6.9: Caso de Uso: CU-9 Ajustar los parámetros de un envelope

CU-9	**Ajustar los parámetros de un envelope**
Actor Principal	Usuario
Descripción	El usuario ajusta el ataque, decay, sustain y release de un envelope.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.

Continúa desde la página anterior

CU-9	**Ajustar los parámetros de un envelope**
Secuencia Normal	1. El usuario ajusta el ataque de un envelope mediante un control deslizante en la interfaz gráfica. 2. El sistema actualiza el ataque del envelope en tiempo real. 3. El nuevo ataque del envelope se refleja visualmente en la interfaz gráfica. 4. El usuario ajusta el decay del envelope mediante un control deslizante en la interfaz gráfica. 5. El sistema actualiza el decay del envelope en tiempo real. 6. El nuevo decay del envelope se refleja visualmente en la interfaz gráfica. 7. El usuario ajusta el sustain del envelope mediante un control deslizante en la interfaz gráfica. 8. El sistema actualiza el sustain del envelope en tiempo real. 9. El nuevo sustain del envelope se refleja visualmente en la interfaz gráfica. 10. El usuario ajusta el release del envelope mediante un control deslizante en la interfaz gráfica. 11. El sistema actualiza el release del envelope en tiempo real. 12. El nuevo release del envelope se refleja visualmente en la interfaz gráfica.

Continúa desde la página anterior

CU-9	**Ajustar los parámetros de un envelope**
Secuencia Alternativa	1a Si el usuario introduce un valor fuera del rango permitido, el sistema limita el ataque del envelope al valor máximo o mínimo permitido. 4a Si el usuario introduce un valor fuera del rango permitido, el sistema limita el decay del envelope al valor máximo o mínimo permitido. 7a Si el usuario introduce un valor fuera del rango permitido, el sistema limita el sustain del envelope al valor máximo o mínimo permitido. 10a Si el usuario introduce un valor fuera del rango permitido, el sistema limita el release del envelope al valor máximo o mínimo permitido.
Postcondiciones	▪ El ataque, decay, sustain y release del envelope son los seleccionados por el usuario. ▪ El sonido generado refleja los nuevos parámetros del envelope. ▪ La interfaz muestra los nuevos parámetros del envelope.

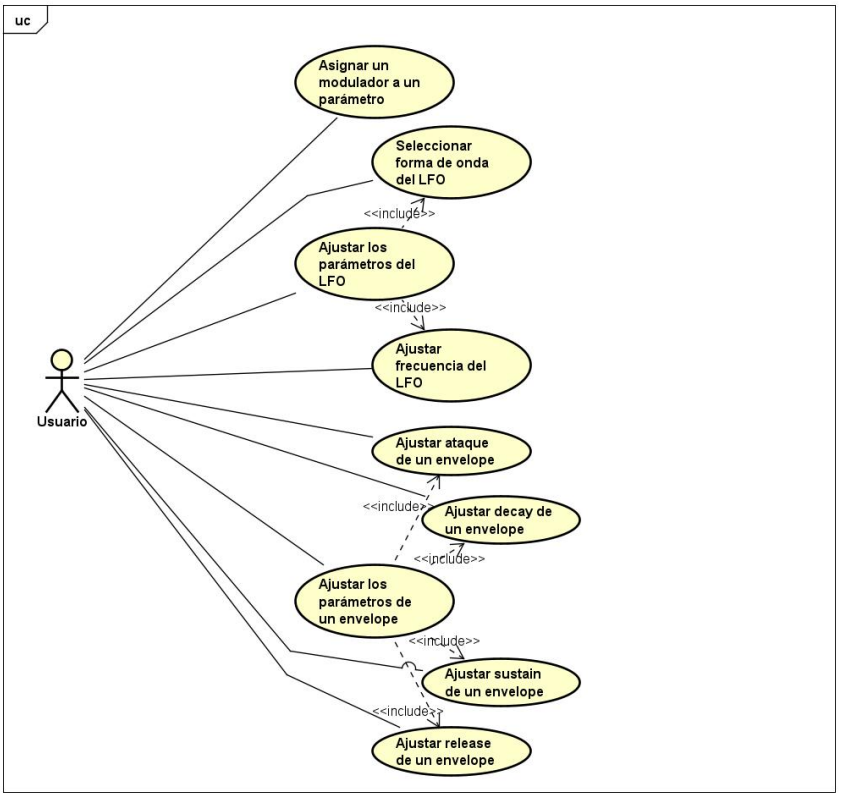


Figura 6.3: Diagrama de casos de uso: Modulación

MIDI

Tabla 6.10: Caso de Uso: CU-10 Seleccionar el canal MIDI de entrada

CU-10	**Seleccionar el canal MIDI de entrada**
Actor Principal	Usuario
Descripción	El usuario selecciona el canal MIDI de entrada para el sistema.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.

Continúa desde la página anterior

CU-10	**Seleccionar el canal MIDI de entrada**
Secuencia Normal	1. El usuario solicita un listado de los canales MIDI. 2. El sistema muestra el listado de los canales MIDI. 3. El usuario selecciona un canal MIDI de la lista. 4. El sistema cambia su canal MIDI de entrada. 5. El sistema refleja gráficamente el nuevo canal MIDI de entrada.
Secuencia Alternativa	3a Si el usuario cierra el listado de canales MIDI sin seleccionar uno, el sistema mantiene la configuración anterior.
Postcondiciones	▪ El canal MIDI de entrada del sistema es el nuevo seleccionado por el usuario.

Tabla 6.11: Caso de Uso: CU-11 Tocar una tecla del controlador MIDI

CU-11	**Tocar una tecla del controlador MIDI**
Actor Principal	Controlador MIDI
Actores Secundarios	Usuario, Sistema de audio
Descripción	El usuario toca una tecla del controlador MIDI.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento. ▪ El sistema debe tener conectado un controlador MIDI externo. ▪ El controlador MIDI debe tener teclado. ▪ El sistema debe tener como canal de entrada MIDI seleccionado, el canal por el que está transmitiendo el controlador MIDI.

Continúa desde la página anterior

CU-11	**Tocar una tecla del controlador MIDI**
Secuencia Normal	1. El usuario presiona una tecla del controlador MIDI. 2. El controlador MIDI genera y envía al sitema el mensaje MIDI Note On correspondiente al evento. 3. El sistema recibe el mensaje MIDI y obtiene su número de nota y velocity. 4. El sistema aplica las modulaciones correspondientes en función de los valores de número de nota y velocity. 5. El sistema genera la señal correspondiente y la comunica al sistema de audio. 6. El usuario libera la presión de la tecla del controlador MIDI. 7. El controlador MIDI genera y envía al sistema el mensaje MIDI Note Off correspondiente al evento. 8. El sistema recibe el mensaje MIDI. 9. El sistema finaliza la generación de la señal anterior.
Postcondiciones	▪ El sistema ha generado la señal correspondiente a la pulsación de la tecla del controlador MIDI.

Tabla 6.12: Caso de Uso: CU-12 Ajustar la modulation wheel del controlador MIDI

CU-12	**Ajustar la modulation wheel del controlador MIDI**
Actor Principal	Controlador MIDI
Actores Secundarios	Usuario
Descripción	El usuario ajusta la modulation wheel del controlador MIDI.

Continúa desde la página anterior

CU-12	**Ajustar la modulation wheel del controlador MIDI**
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento. ▪ El sistema debe tener conectado un controlador MIDI externo. ▪ El controlador MIDI debe tener modulation wheel. ▪ El sistema debe tener como canal de entrada MIDI seleccionado, el canal por el que está transmitiendo el controlador MIDI.
Secuencia Normal	1. El usuario ajusta la posición de la modulation wheel del controlador MIDI. 2. El controlador MIDI genera y envía al sitema el mensaje MIDI correspondiente al evento. 3. El sistema recibe el mensaje MIDI y obtiene la posición de la modulation wheel. 4. El sistema se actualiza aplicando las modulaciones correspondientes en función del valor de la posición de la modulation wheel en tiempo real.
Postcondiciones	▪ El sistema ha actualizado su estado según el nuevo valor de la posición de la modulation wheel del controlador MIDI.

Tabla 6.13: Caso de Uso: CU-13 Ajustar pitch bend del controlador MIDI

CU-13	**Ajustar pitch bend del controlador MIDI**
Actor Principal	Controlador MIDI
Actores Secundarios	Usuario
Descripción	El usuario ajusta la posición del pitch bend del controlador MIDI.

Continúa desde la página anterior

CU-13	**Ajustar pitch bend del controlador MIDI**
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento. ▪ El sistema debe tener conectado un controlador MIDI externo. ▪ El controlador MIDI debe tener control de pitch bend. ▪ El sistema debe tener como canal de entrada MIDI seleccionado, el canal por el que está transmitiendo el controlador MIDI.
Secuencia Normal	1. El usuario ajusta la posición del pitch bend del controlador MIDI. 2. El controlador MIDI genera y envía al sitema el mensaje MIDI correspondiente al evento. 3. El sistema recibe el mensaje MIDI y obtiene la posición de pitch bend. 4. El sistema cambia la frecuencia del oscilador en función del valor de la posición de pitch bend en tiempo real.
Postcondiciones	▪ El sistema ha actualizado la frecuencia del oscilador según el nuevo valor de la posición de pitch bend del controlador MIDI.

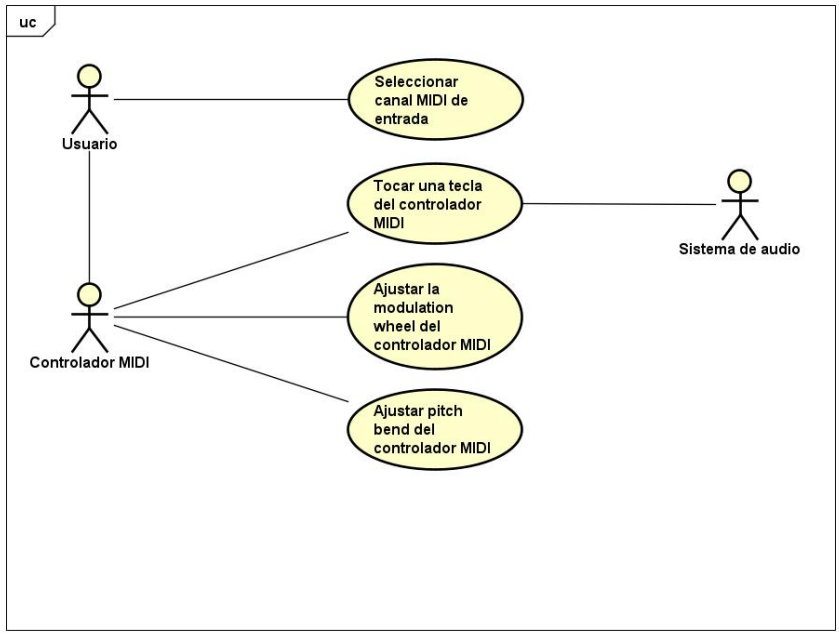


Figura 6.4: Diagrama de casos de uso: MIDI

Gestión de presets

Tabla 6.14: Caso de Uso: CU-14 Guardar la configuración actual como un preset

CU-14	**Guardar la configuración actual como un preset**
Actor Principal	Usuario
Descripción	El usuario guarda la configuración actual del sistema como un preset.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.

Continúa desde la página anterior

CU-14	**Guardar la configuración actual como un preset**
Secuencia Normal	1. El usuario indica que desea guardar la configuración actual del sistema como un preset. 2. El sistema pide al usuario que indique la ruta de almacenamiento del preset. 3. El usuario indica la ruta de almacenamiento. 4. El sistema genera y almacena un archivo con su configuración actual.
Secuencia Alternativa	3a Si ya existe un archivo en la ruta indicada por el usuario, el sistema preguntará al usuario si quiere sobreescribir el archivo. ▪ Si la respuesta del usuario es afirmativa, el sistema generará y almacenará el preset sobreescribiendo el archivo con su misma ruta y el caso de uso finalizará. ▪ Si la respuesta es negativa, el caso de uso volverá al paso 2. 3b Si el usuario cancela la acción el caso de uso finaliza efecto.
Postcondiciones	▪ El sistema ha generado y almacenado un archivo con los datos necesarios para recuperar su configuración actual.

Tabla 6.15: Caso de Uso: CU-15 Cargar un preset

CU-15	**Cargar un preset**
Actor Principal	Usuario
Descripción	El usuario carga un preset de configuración del sistema.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento. ▪ El sistema debe tener acceso y permisos de lectura al archivo que contiene el preset.

Continúa desde la página anterior

CU-15	**Cargar un preset**
Secuencia Normal	1. El usuario indica que desea cargar un preset. 2. El sistema pide al usuario que indique la ruta del preset. 3. El usuario indica la ruta del preset. 4. El sistema actualiza su configuración a la indicada por el preset.
Secuencia Alternativa	3a Si la ruta o el archivo no son válidos, el sistema muestra un mensaje de error y el caso de uso vuelve al paso 2. 3b Si el usuario cancela la acción, el sistema mantiene su configuración y el caso de uso finaliza sin efecto.
Postcondiciones	▪ La configuración del sistema es la indicada por el preset cargado.

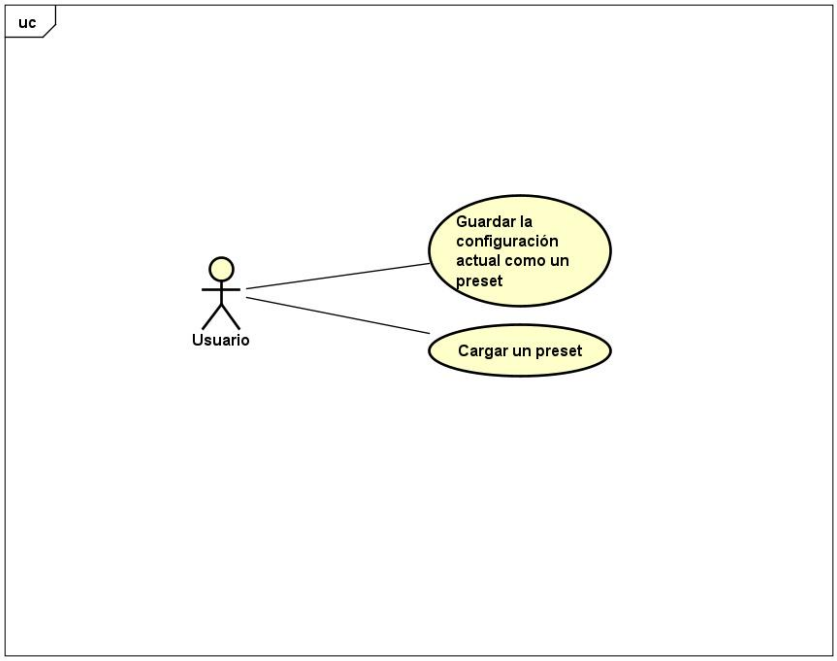


Figura 6.5: Diagrama de casos de uso: Gestión de presets

Configuración general

Tabla 6.16: Caso de Uso: CU-16 Restablecer un parámetro a su valor por defecto

CU-16	**Restablecer un parámetro a su valor por defecto**
Actor Principal	Usuario
Descripción	El usuario restablece un parámetro a su valor por defecto.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario indica que desea restaura el valor por defecto de un determinado parámetro del sistema. 2. El sistema actualiza el parámetro indicado a su valor por defecto. 3. El nuevo valor del parámetro se refleja visualmente en la interfaz gráfica.

Continúa desde la página anterior

CU-16	**Restablecer un parámetro a su valor por defecto**
Postcondiciones	▪ El parámetro indicado por el usuario tiene su valor por defecto .

Tabla 6.17: Caso de Uso: CU-17 Seleccionar la cantidad máxima de voces en polifonía

CU-17	**Seleccionar la cantidad máxima de voces en polifonía**
Actor Principal	Usuario
Descripción	El usuario selecciona el número máximo de voces simultáneas en polifonía.
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario ajusta el número de máximo voces simultáneas mediante un control en la interfaz gráfica. 2. El sistema actualiza el número máximo de voces simultáneas. 3. El nuevo número máximo de voces simultáneas se refleja visualmente en la interfaz gráfica.
Secuencia Alternativa	1a Si el usuario introduce un valor fuera del rango permitido, el sistema limita el número máximo de voces al valor máximo o mínimo permitido.
Postcondiciones	▪ El número máximo de voces simultáneas es el establecido por el usuario.

Tabla 6.18: Caso de Uso: CU-18 Cambiar la configuración de salida de audio

CU-18	**Cambiar la configuración de salida de audio**
Actor Principal	Usuario
Actores Secundarios	Sistema de audio
Descripción	El usuario cambia configuración de salida de audio.

Continúa desde la página anterior

CU-18	**Cambiar la configuración de salida de audio**
Precondiciones	▪ El sistema debe estar encendido y en funcionamiento.
Secuencia Normal	1. El usuario solicita cambiar la configuración de salida de audio. 2. El sistema muestra a través de la interfaz gráfica un panel con la configuración de audio, mostrando la configuración actual. 3. El usuario solicita cambiar el dispositivo de salida de audio. 4. El sistema solicita al sistema de audio un listado de los dispositivos de salida disponibles. 5. El sistema muestra a través de la interfaz gráfica el listado de los dispositivos de salida disponibles. 6. El usuario selecciona un dispositivo de salida. 7. El sistema actualiza su salida de audio comunicándose con el sistema de audio. 8. La nueva salida de audio se refleja gráficamente a través de la interfaz. 9. El usuario solicita cambiar la configuración del driver del dispositivo de salida de audio. 10. El sistema transfiere la solicitud al sistema de audio. 11. El usuario solicita cerrar el panel de configuración. 12. El sistema deja de mostrar a través de la interfaz gráfica el panel de configuración.
Secuencia Alternativa	6a Si el usuario cierra el panel de configuración, el sistema mantiene la configuración establecida. 1. Si en cualquier momento a partir del paso 2 el usuario cierra el panel de configuración, el sistema mantiene la configuración establecida hasta el momento.

Continúa desde la página anterior

CU-18	**Cambiar la configuración de salida de audio**
Postcondiciones	▪ La configuración de salida de audio es la establecida por el usuario.

Tabla 6.19: Caso de Uso: CU-19 Iniciar sistema

CU-19	**Iniciar sistema**
Actor Principal	Usuario
Actores Secundarios	Sistema de audio
Descripción	El usuario inicia la aplicación.
Precondiciones	▪ El sistema debe estar correctamente instalado en la máquina que utiliza el usuario.
Secuencia Normal	1. El usuario ejecuta la aplicación del sistema. 2. El sistema realiza las operaciones necesarias para comenzar su utilización, conectándose con el sistema de audio. 3. El sistema carga su configuración por defecto. 4. El sistema muestra que está listo para utilizarse y esta configuración a través de la interfaz gráfica.
Postcondiciones	▪ El sistema se ha iniciado y está listo para su utilización.

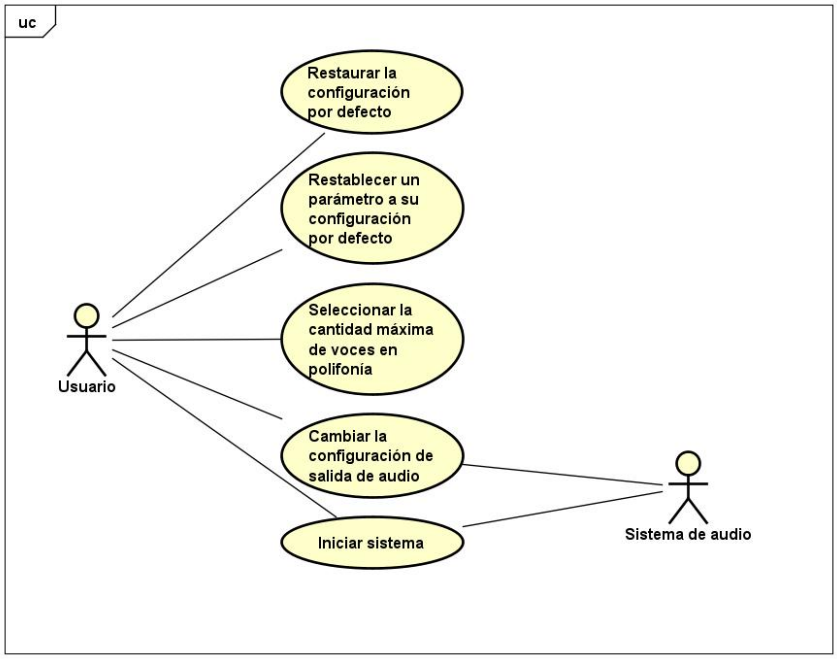


Figura 6.6: Diagrama de casos de uso: Configuración general

6.4. Modelo de Dominio

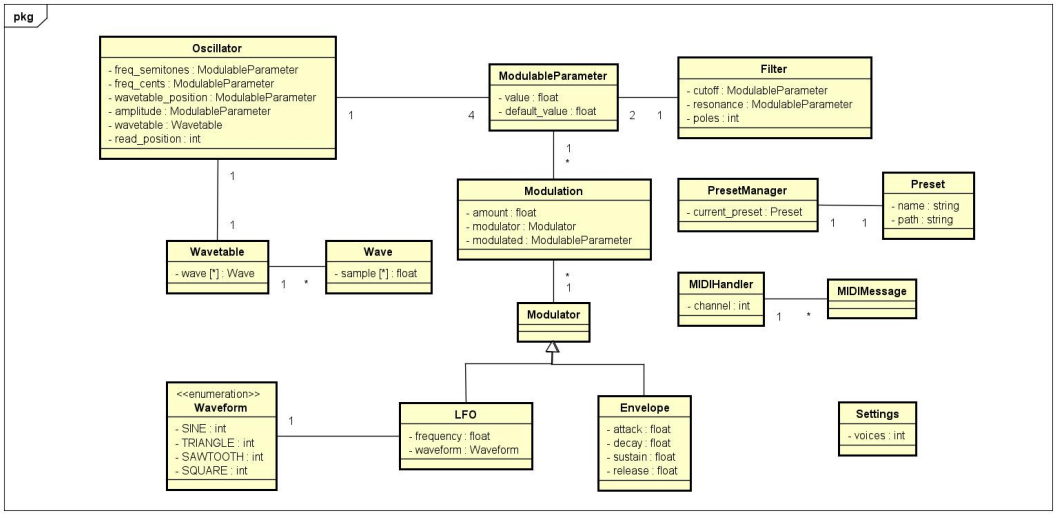


Figura 6.7: Modelo de dominio

Capítulo 7

Tecnologías utilizadas

7.1. Lenguaje de programación

Figura 7.1: Logo de C++ [11]

Para la implementación de la aplicación de este proyecto se han explorado diversos lenguajes de programación, buscando el que mejor se adapte a sus necesidades.
El principal factor determinante para elegir ha sido la necesidad de generar respuestas en tiempo real. Por ello se ha optado por un lenguaje compilado frente a uno interpretado.
Otra característica importante es la complejidad y modularidad del sistema, lo que ha fomentado el uso de un lenguaje orientado a objetos.
El proceso de selección llegó hasta dos candidatos finales: **C++** [11] y **Rust** [30]. Finalmente se decidió utilizar **C++** debido a ser el soporte del framework descrito en la siguiente sección.

7.2. Framework

Figura 7.2: Logo de JUCE [13]

Para facilitar la implementación y mejorar las cualidades de la aplicación se ha buscado un framework de *Digital Signal Processing* (DSP). Se ha decidido utilizar **JUCE** debido a ser el principal estándar actual en esta industria y a facilitar la portabilidad de la aplicación.

JUCE es el framework más utilizado para el desarrollo de aplicaciones de audio y plug-ins. Se trata de una base de código C++ de código abierto que puede utilizarse para crear software independiente en Windows, macOS, Linux, iOS y Android, así como plug-ins VST, VST3, AU, AUv3, AAX y LV2.

JUCE permite a los desarrolladores centrarse en las partes más valiosas de su software al ocuparse de las diferencias entre sistemas operativos (tanto de escritorio como móviles) y formatos de plug-ins. Gracias a la biblioteca de bloques de procesamiento digital de audio (DSP) de JUCE, es posible crear prototipos y lanzar rápidamente aplicaciones y plug-ins nativos con una experiencia de usuario coherente en todas las plataformas compatibles. El uso de JUCE también garantiza el futuro de sus productos frente a las actualizaciones del sistema operativo y del host de plug-ins. [**?**]
Por otro lado, JUCE proporciona facilidades para gestionar automáticamente la comunicación con el sistema de audio y la lectura e interpretación de mensajes MIDI.

7.3. Entorno de desarrollo

Figura 7.3: Logo de Microsoft Visual Studio [23]

Siguiendo las pautas de utilización de JUCE, se ha utilizado la herramienta de configuración **Projucer**, así cómo el IDE **Microsoft Visual Studio**.

Visual Studio es una plataforma de lanzamiento creativa que puede utilizar para editar, depurar y compilar código y, finalmente, publicar una aplicación. Además del editor y depurador estándar que ofrecen la mayoría de IDE, Visual Studio incluye compiladores, herramientas de

completado de código, diseñadores gráficos y muchas más funciones para mejorar el proceso de desarrollo de software. [23]

7.4. Planificación

Figura 7.4: Logo de Microsoft Project [22]

Microsoft project [22] es un estándar software para la planificación y gestión de proyectos. Se ha utilizado para la planificación temporal del proyecto y la realización de diagramas de Gantt.

Figura 7.5: Logo de Microsoft Excel [21]

Microsoft Excel [21] es un software de hojas de cálculo .Se ha utilizado para calcular los costes y el presupuesto.

7.5. Análisis y Diseño

Figura 7.6: Logo de Astah UML [4]

Astah es una herramienta de modelado UML desarrollada por Change Vision [4]. Se ha utilizado para trazar los diagramas de análisis y diseño.

Figura 7.7: Logo de Draw.io [6]

Draw.io es una herramienta de diseño gráfico utilizada en este proyecto para el diseño de la interfaz gráfica [6].

7.6. Control de versiones

Figura 7.8: Logo de Gitlab [8]

GitLab es una herramienta de gestión de alojamiento de repositorios desarrollada por GitLab Inc y utilizada para el proceso de desarrollo de software. Proporciona una variedad de gestión mediante la cual se puede agilizar el flujo de trabajo colaborativo para completar el ciclo de vida de desarrollo de software. [8]

7.7. Despliegue

Figura 7.9: Logo de Inno Setup [12]

Inno Setup [12] es un software gratuito para la creación de instaladores en Windows. Se ha empleado para el despliegue del proyecto, generando un ejecutable de instalación del proyecto con configuraciones personalizadas.

7.8. Memoria

Figura 7.10: Logo de Overleaf [25]

Overleaf [25] es una plataforma en línea de escritura colaborativa basada en LaTeX. Se ha utilizado para la redacción y edición de este documento.

Capítulo 8

Diseño

8.1. Arquitectura general del sistema

La arquitectura general del sistema es una arquitectura MVC por capas más una capa de servicio (*Arquitectura MVC + service layer*). Además, se ha añadido una capa de persistencia para lecturas y escrituras de datos.

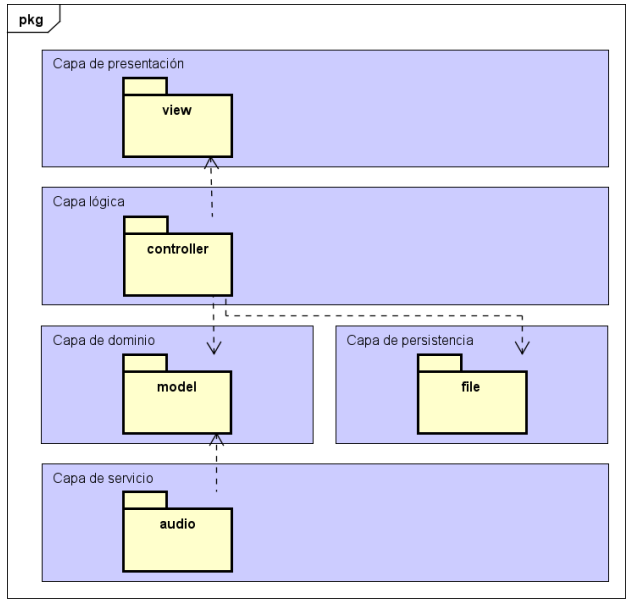


Figura 8.1: Arquitectura general del sistema. MVC + service layer

- **Capa de presentación:**
  La capa de presentación se encarga de gestionar la interfaz gráfica de usuario, mostrándola debidamente y capturando eventos de interacciones del usuario con esta interfaz.

- **Capa lógica:**
  La capa lógica se encarga de conectar la capa de presentación con la capa de dominio, manteniendo una coherencia entre el estado de ambas. Actualizando la capa de dominio cuando la capa de presentación cambia y a su vez actualizando la capa de presentación cuando la capa de dominio cambia.
  También se encarga de transferir los datos necesarios entre la capa de dominio y la capa de persistencia para la lectura y escritura de datos.

- **Capa de dominio:**
  La capa de dominio contiene el estado interno de la aplicación, es decir, los valores de todos los parámetros en cada momento. También se encarga de serializar su estado y deserializar un vector de bytes para actualizar su estado.

- **Capa de persistencia:**
  La capa de persistencia se encarga de escribir en disco datos serializados y de leer datos serializados de disco.

- **Capa de servicio:**
  La capa de servicio se encarga de proporcionar el motor de audio al sistema. Utiliza la capa de dominio para determinar el sonido a generar. Además se encarga de la gestión de mensajes MIDI y se comunica con el sistema de audio.

8.2. Estructura de paquetes

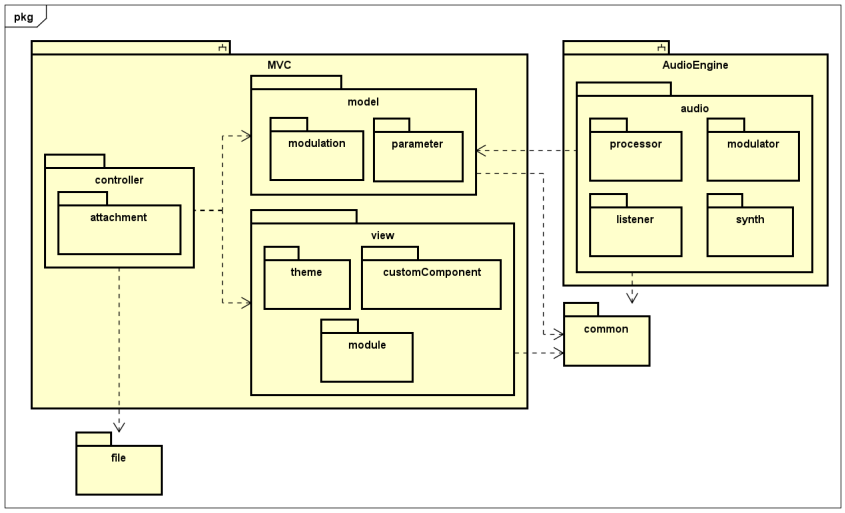


Figura 8.2: Diagrama de paquetes del sistema

8.3. Interfaz gráfica de usuario

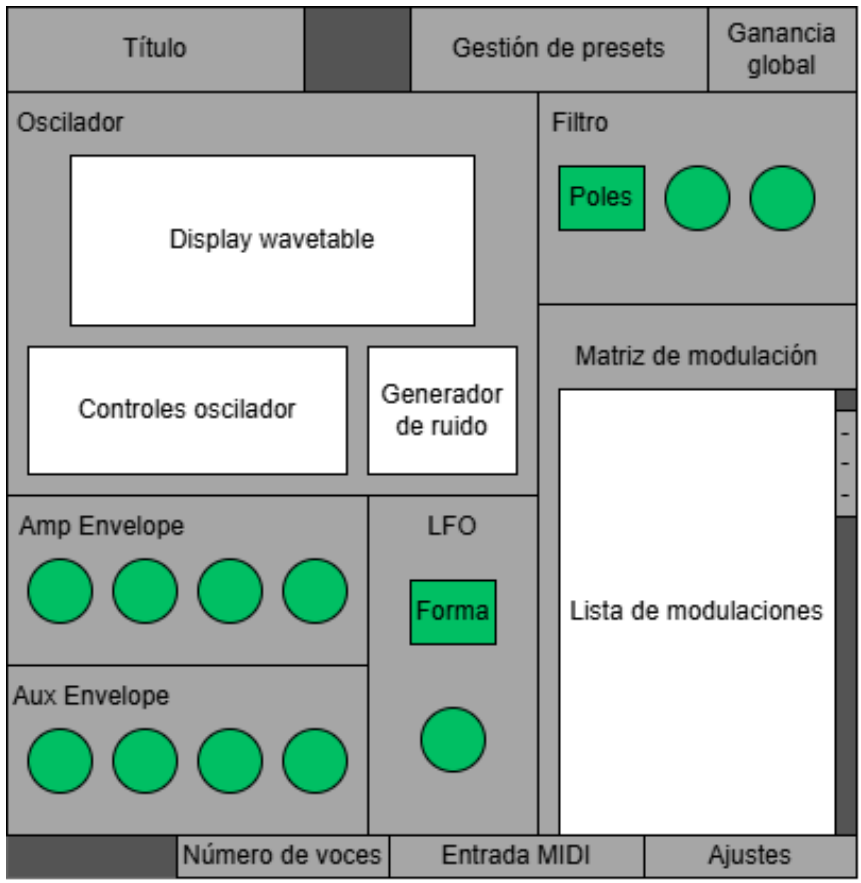


Figura 8.3: Diseño de la interfaz gráfica de usuario GUI

Para diseñar la interfaz se ha optado por seguir un paradigma común en la industria consistente en evitar la navegación profunda de menús y pestañas, acercándose más a la interfaz de un instrumento analógico, donde todos los controles son accesibles inmediatamente.
Por otro lado, se ha recurrido a un diseño compacto para facilitar la utilización de la aplicación como ventana flotante sin ocupar todo el espacio de la pantalla, lo cual es una práctica común ya que estos instrumentos se suelen utilizar en conjunto con otras aplicaciones.
Se ha elegido una paleta de colores suave a la vista para no suponer una carga durante largas sesiones de trabajo. Con una estética moderna que refleja esta misma cualidad del instrumento.

8.4. Principales patrones de diseño

A lo largo de la aplicación se han aplicado diferentes patrones de diseño software siendo estos los más notorios.

8.4.1. Fachada

El patrón de diseño Fachada se emplea para ocultar la complejidad de subsistemas proporcionando una única interfaz de alto nivel [7].

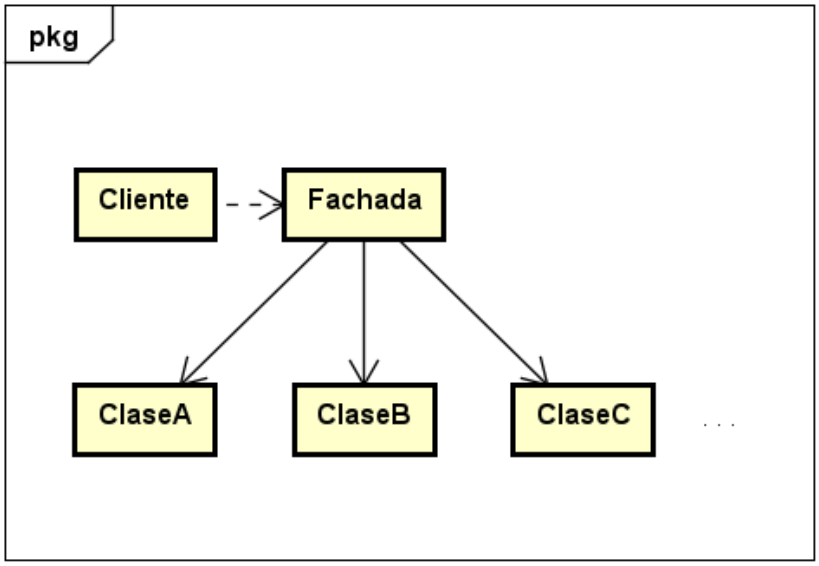


Figura 8.4: Patrón fachada genérico

Este patrón se ha aplicado en el modelo *SynthModel*, que implementa la interfaz *SynthModelFacade* lo que permite el acceso a la información contenida en el modelo evitando dependencias internas al paquete *model*. El motor de audio utiliza esta interfaz fachada para acceder al modelo manteniendo un alto nivel y evitando el acoplamiento entre estos paquetes diferentes.

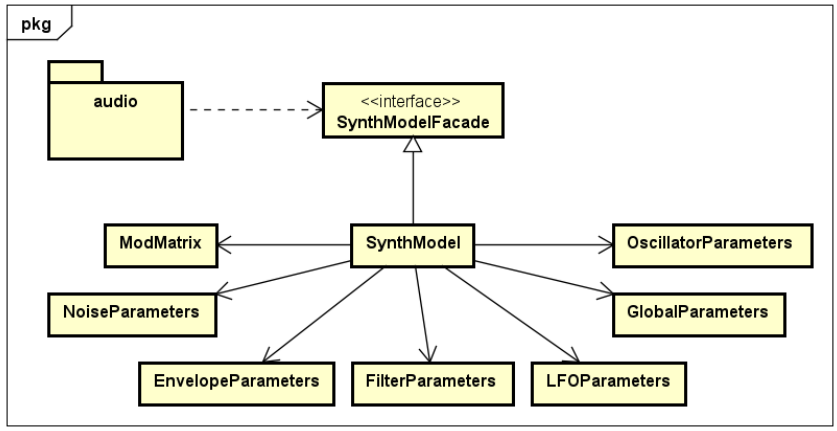


Figura 8.5: Patrón fachada aplicado al modelo SynthModel

Esta fachada se apoya en las estructuras de datos (*structs*) *ModelState*, *ModelRanges* y *ModulationState* simplemente para facilitar la comunicación.

8.4.2. Patrón estrategia

El patrón de diseño Estrategia define una familia de algoritmos, los encapsula y los hace intercambiables. Este patrón permite que el algoritmo varíe independientemente de los clientes que lo utilizan, favoreciendo la extensión y reutilización de código [7].

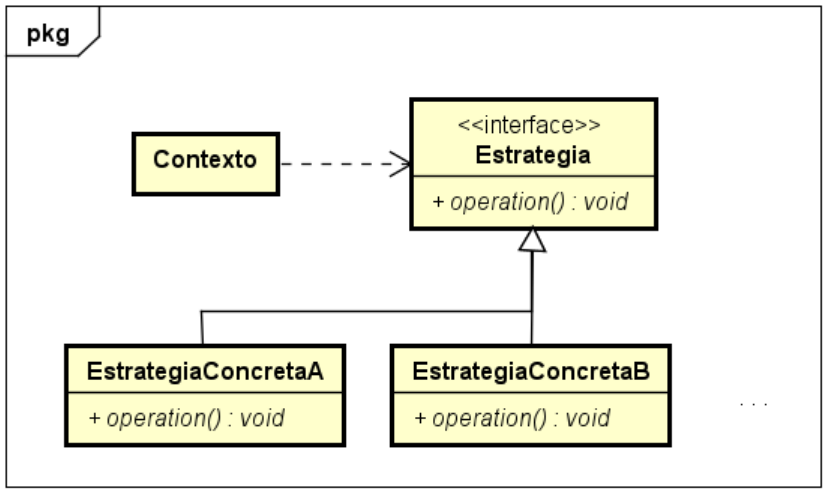


Figura 8.6: Patrón estrategia genérico

Este patrón se ha aplicado en el motor de audio para la utilización de distintos moduladores. El objetivo de todos los moduladores es alterar un valor, pero cada uno lo cumple de forma diferente. Para aplicar este patrón, se ha definido una interfaz común *AudioModulator* que implementa cada modulador.

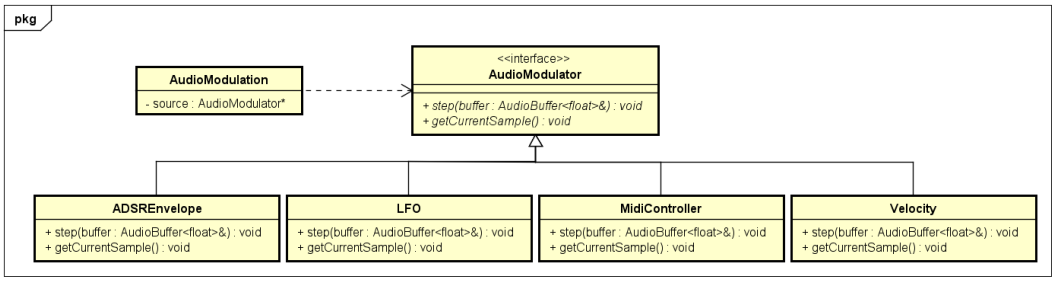


Figura 8.7: Patrón estrategia aplicado a moduladores

AudioModulation utiliza esta interfaz *AudioModulator* para definir una modulación de un parámetro por un modulador, desconociendo el comportamiento concreto del modulador. Separando así las responsabilidades de cada clase y reduciendo el acoplamiento.

8.4.3. Patrón observador

El patrón observador define una dependencia uno a muchos entre objetos, de manera que cuando uno cambia su estado, todos sus dependientes son notificados y actualizados automáticamente. Este patrón es útil para mantener sincronizados múltiples componentes evitando un fuerte acoplamiento [7].

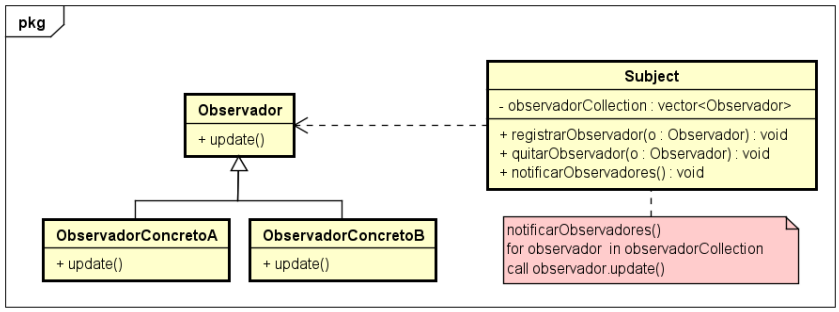


Figura 8.8: Patrón observador genérico

Este patrón se ha aplicado mayoritariamente para conectar el modelo y la vista a través del controlador. Por ejemplo para conectar *sliders* de de la interfaz gráfica con parámetros del modelo.

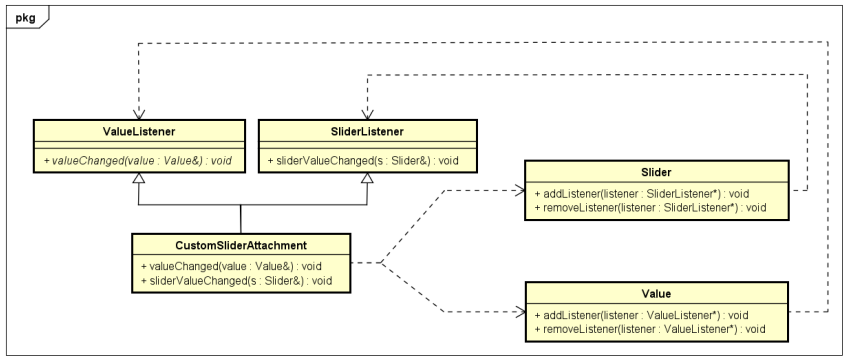


Figura 8.9: Patrón observador aplicado a sliders

También se ha aplicado el patrón observador para conectar *ComboBoxes* de la interfaz gráfica con el modelo. A su vez, en este caso este patrón se ha combinado con el patrón estrategia para gestionar *ComboBoxes* con valores numéricos o categóricos de manera uniforme.

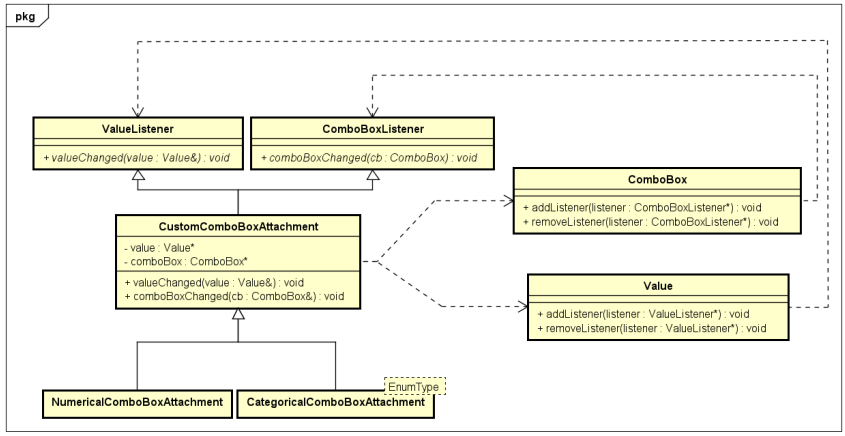


Figura 8.10: Patrón observador aplicado ComboBoxes

Es muy importante que cada observador, a la hora de ser destruido, deje de observar los sujetos que observa (llamando al método del sujeto *removeListener(this)*) ya que, en caso contrario, el sujeto mantendría una referencia a un objeto destruido y trataría de notificarlo, referenciando a una zona de memoria desconocida.

8.5. Desglose de paquetes

A continuación se detallan los diagramas de clases de cada paquete. Las clases cuyo nombre comienza por *juce::* son clases del framework *JUCE*, el resto de clases serán totalmente

implementadas.

8.5.1. Paquete view

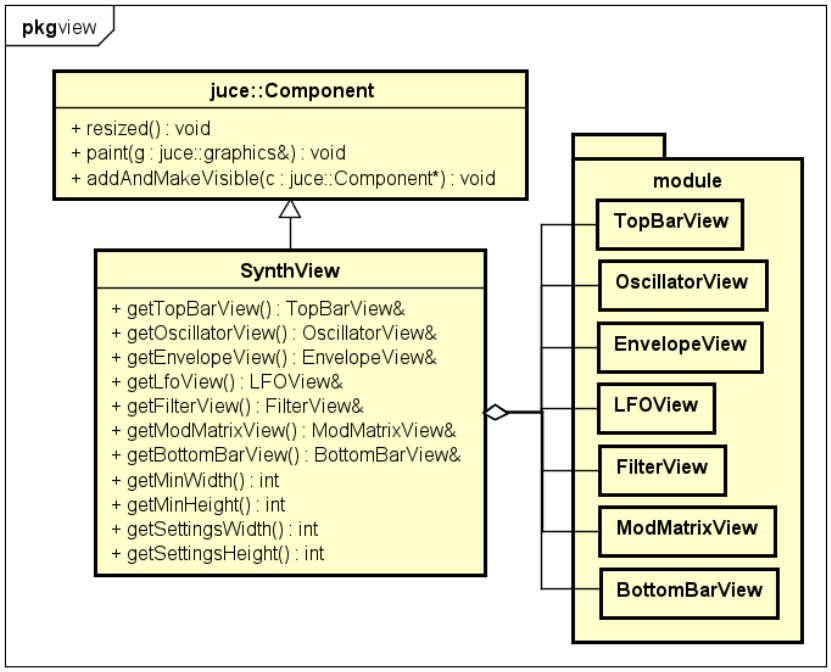


Figura 8.11: Diagrama de clases del paquete view

Subpaquete module

- Oscillator

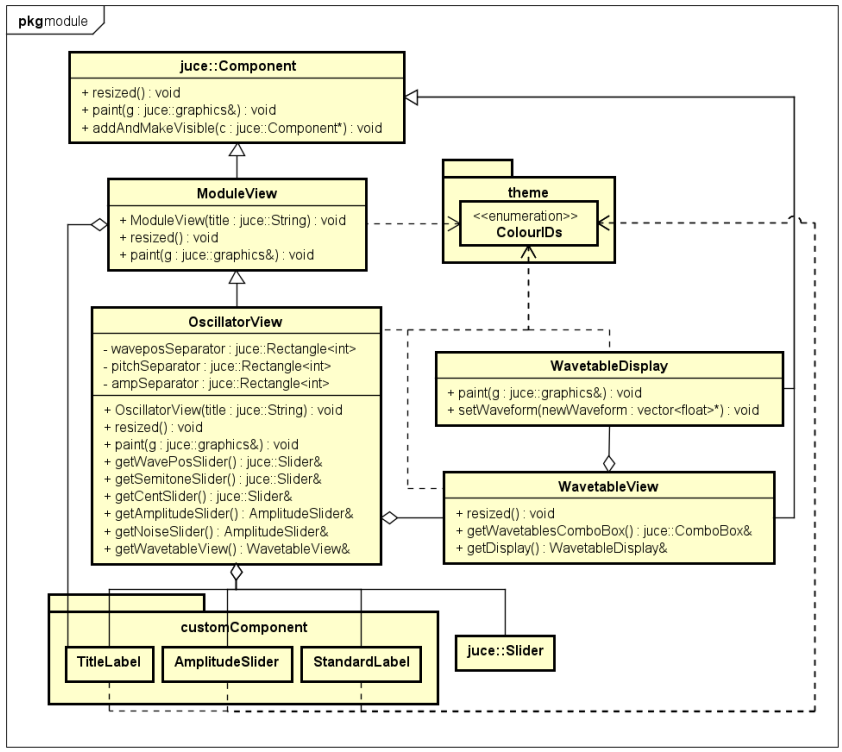


Figura 8.12: Diagrama de clases de la vista del oscilador

- Envelope

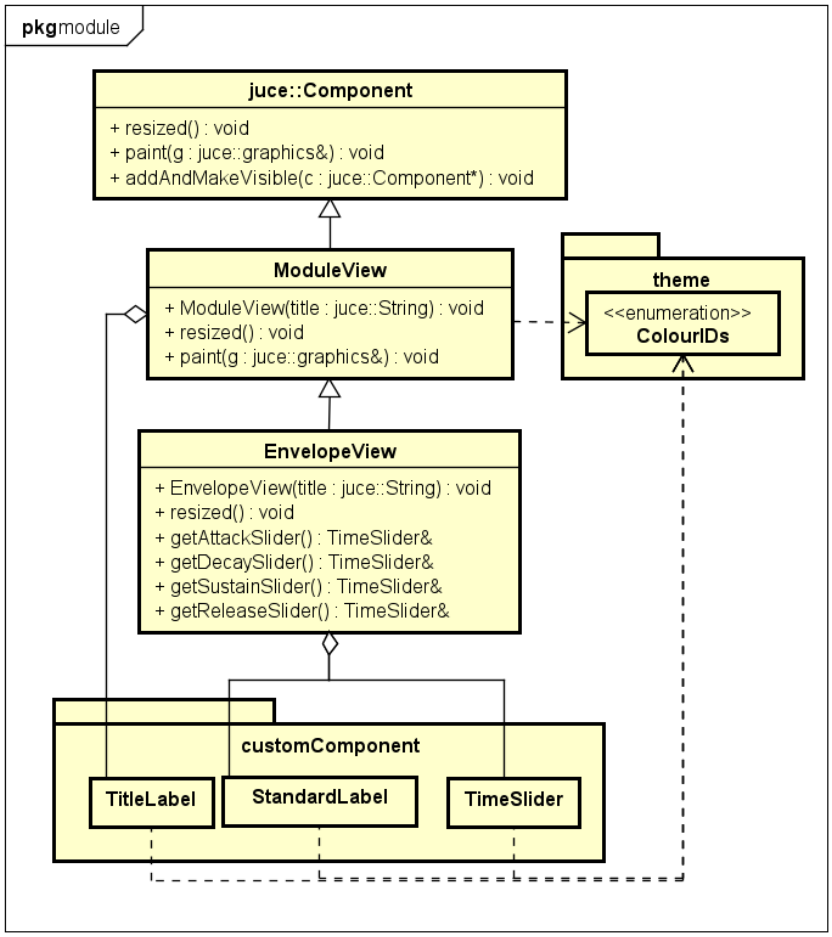


Figura 8.13: Diagrama de clases de la vista de un envelope

- LFO

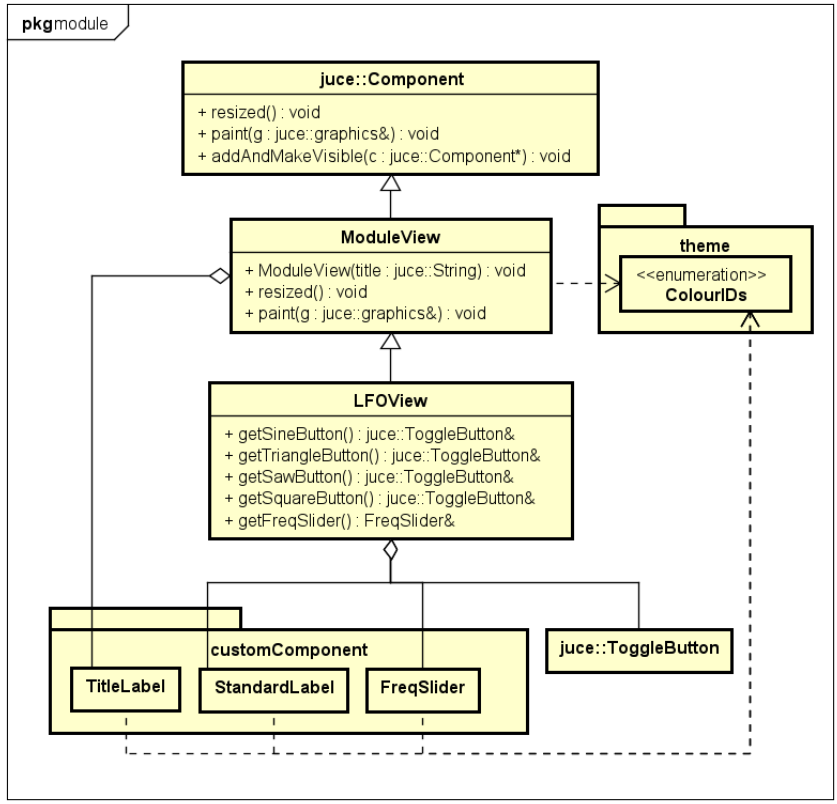


Figura 8.14: Diagrama de clases de la vista del LFO

- Filter

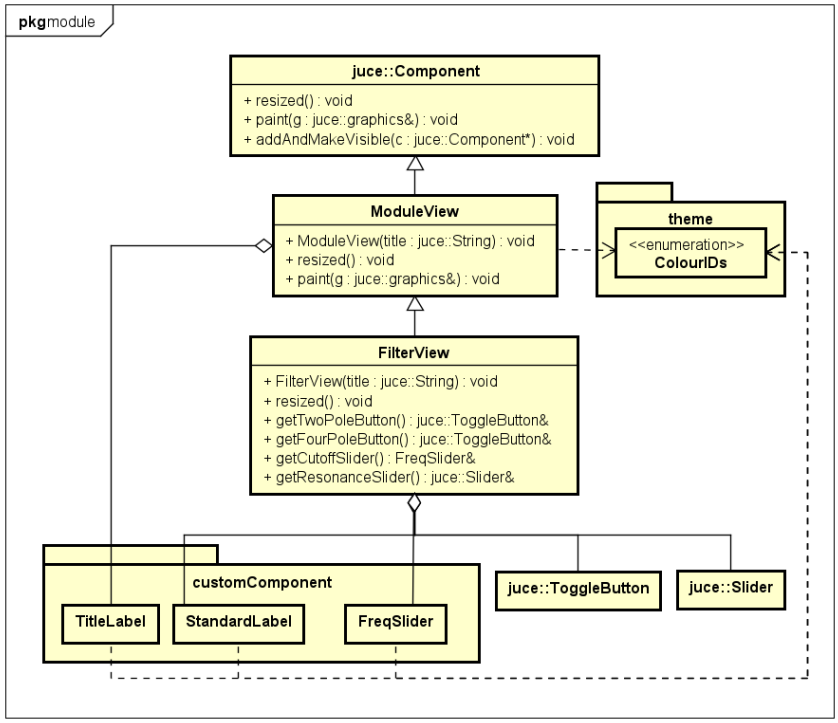


Figura 8.15: Diagrama de clases de la vista del filtro

- Modulation Matrix

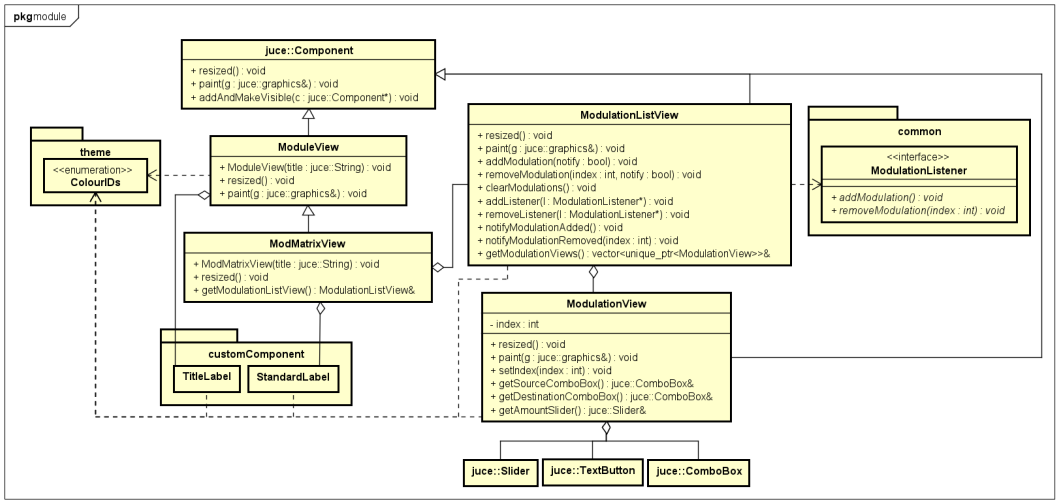


Figura 8.16: Diagrama de clases de la vista de la matriz de modulación

- Bar

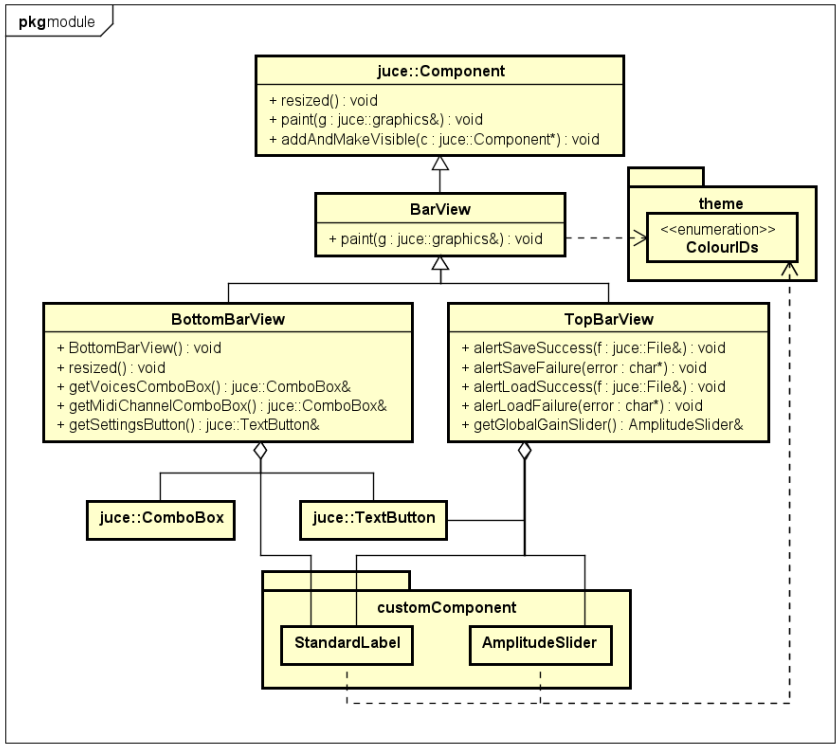


Figura 8.17: Diagrama de clases de la vista de las barras superior e inferior

Subpaquete customComponent

En este subpaquete se encuentran clases que heredan de otras clases del framework JUCE con el objetivo de establecer un comportamiento determinado. Por ejemplo, mostrar amplitudes en decibelios, parámetros temporales en segundos y milisegundos o frecuencias en Hercios y Kilohercios.

- Sliders

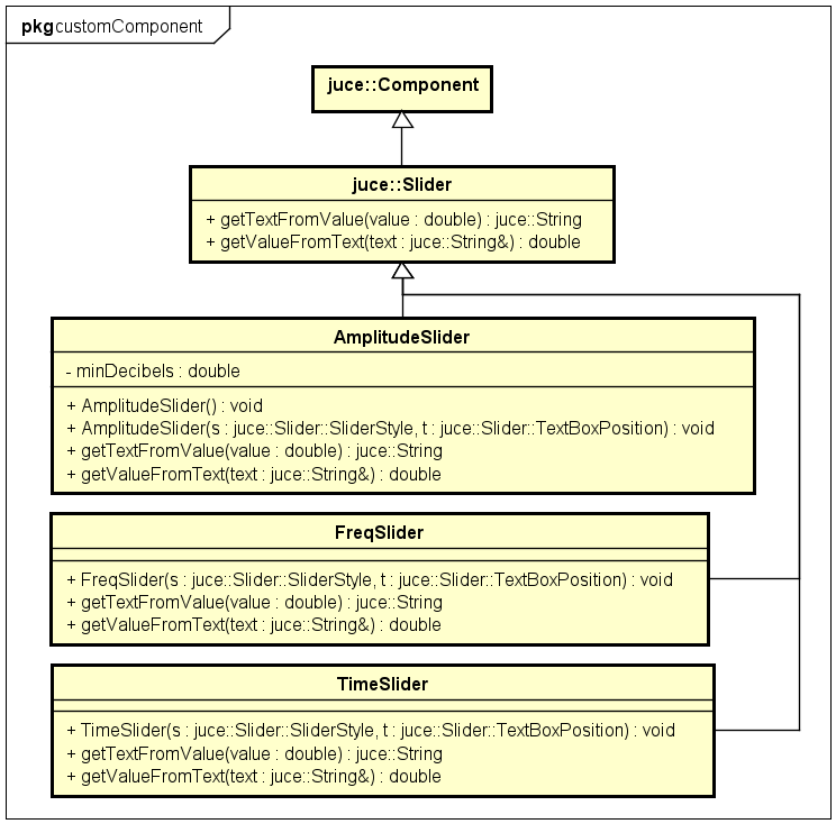


Figura 8.18: Diagrama de clases de los sliders personalizados

- Labels

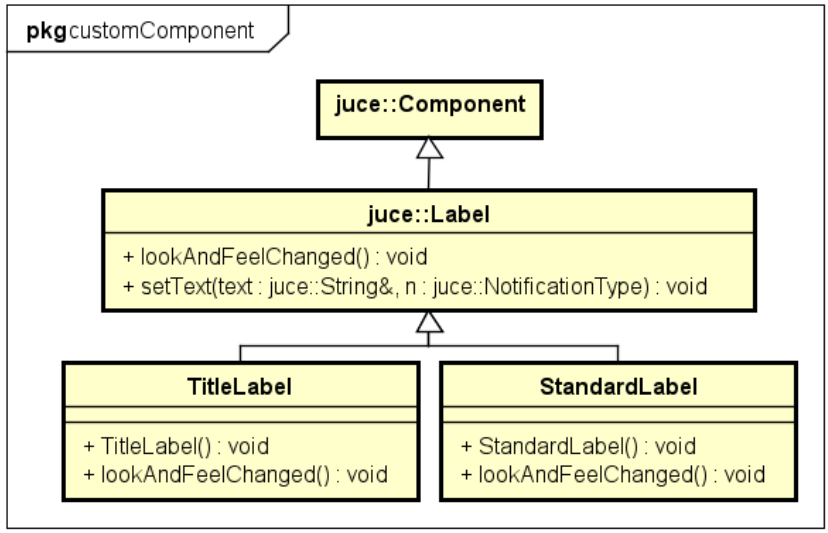


Figura 8.19: Diagrama de clases de las etiquetas personalizadas

Subpaquete theme

Este subpaquete se encarga de gestionar los colores de la aplicación.

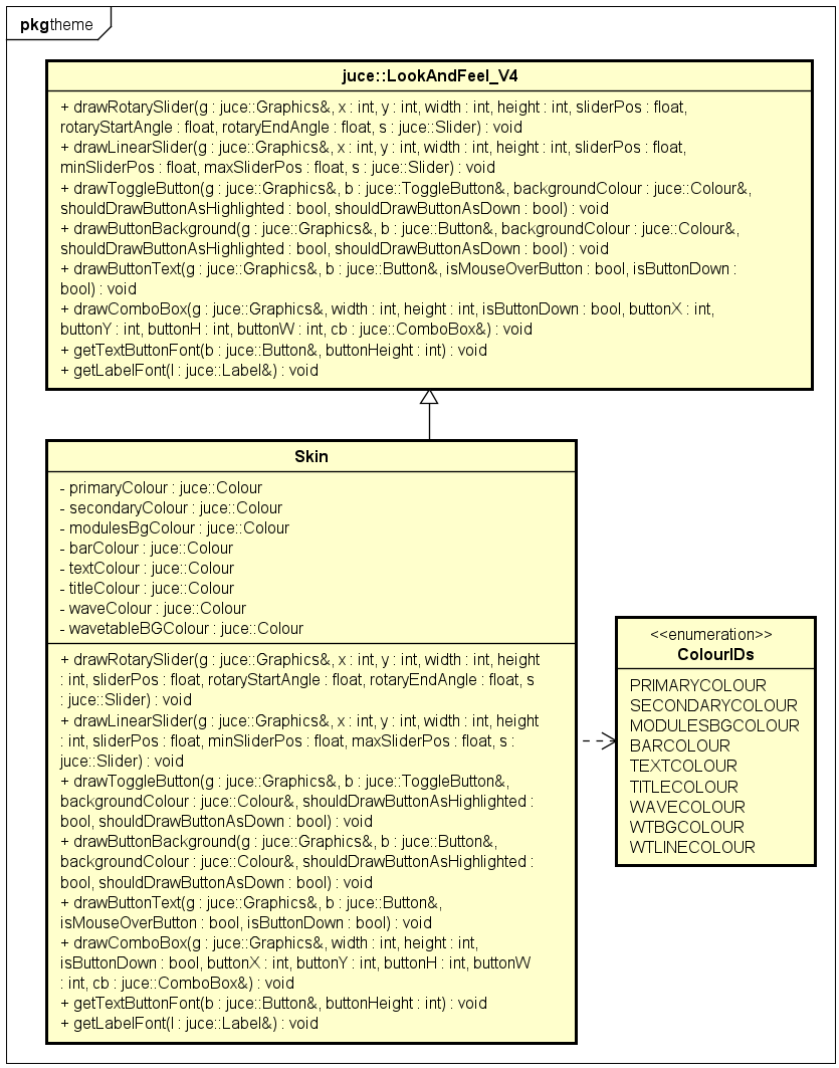


Figura 8.20: Diagrama de clases del paquete theme

8.5.2. Paquete controller

Este paquete contiene clases attachment, que conectan componentes de la vista con componentes del modelo bidireccionalmente. Es decir, si se produce un cambio en la vista en un elemento, el attachment correspondiente actualiza el elemento del modelo asociado; también

si se produce un cambio en un elemento del modelo, el attachment correspondiente actualiza el elemento de la vista asociado.
Los attachments son creados y gestionados desde la clase SynthController.

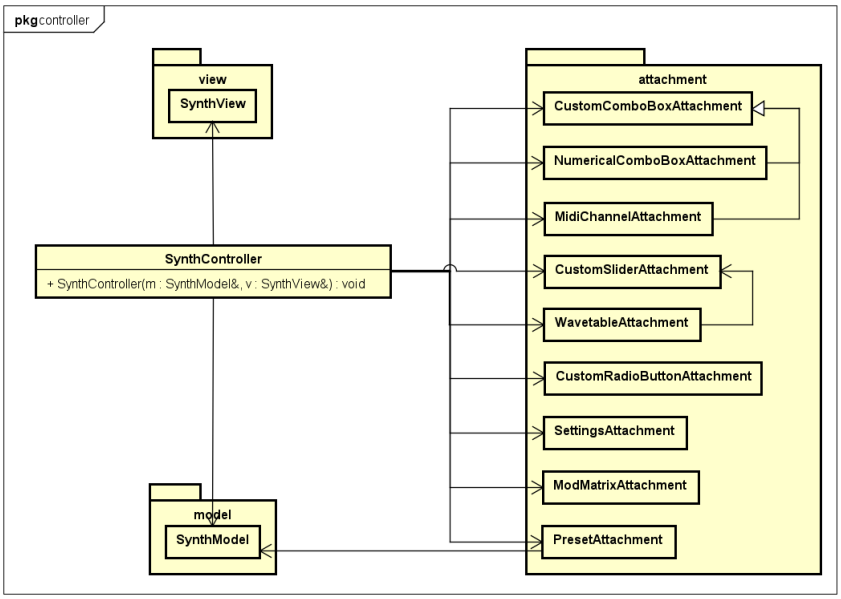


Figura 8.21: Diagrama de clases del paquete controller

Subpaquete attachment

- Attachments principales

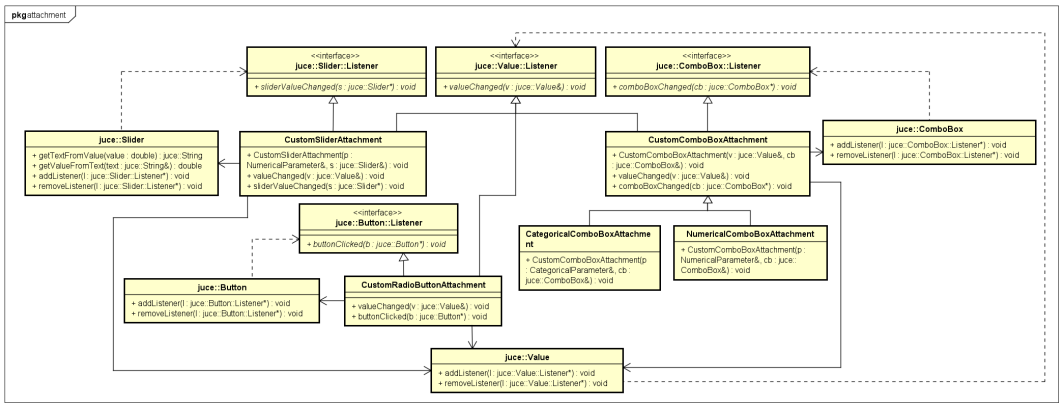


Figura 8.22: Diagrama de clases de attachments principales

- Attachments especiales
  Algunas partes del sistema requieren attachments más personalizados en vez de utilizar los attachments principales.

  - Attachment canal MIDI

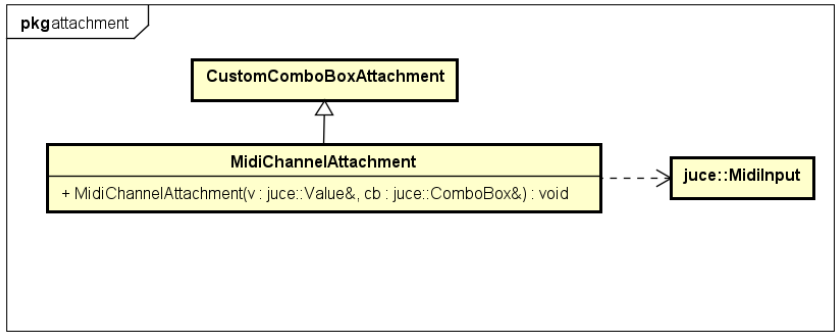


Figura 8.23: Diagrama de clases del attachment del canal MIDI

  - Attachment Matriz de modulación

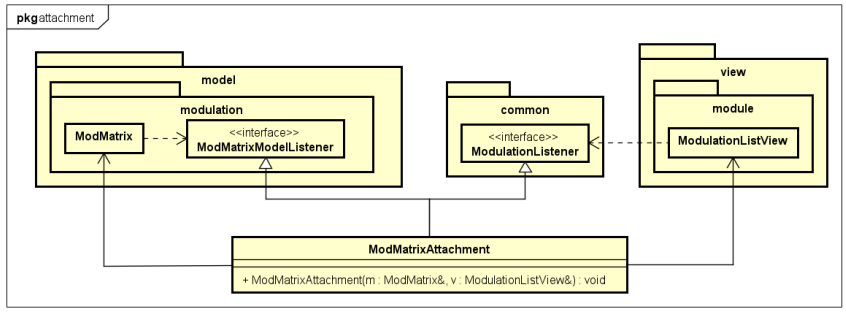


Figura 8.24: Diagrama de clases del attachment de la matriz de modulación

- Attachment presets

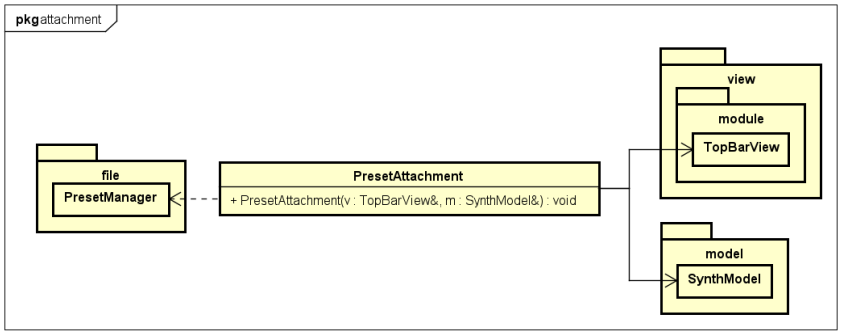


Figura 8.25: Diagrama de clases del attachment de los presets

- Attachment ajustes

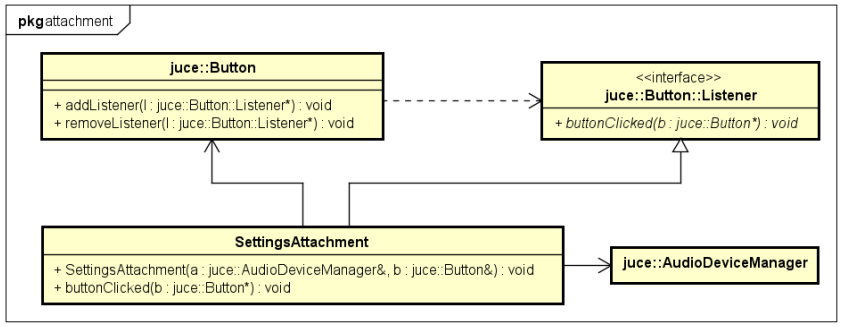


Figura 8.26: Diagrama de clases del attachment de la configuración

- Attachment wavetable
  Es necesario que este attachment maneje también el parámetro wavepos en vez de utilizar otro CustomSliderAttachment por separado porque cuando la wavetable seleccionada cambia, es necesario actualizar los rangos de wavepos y de su slider.

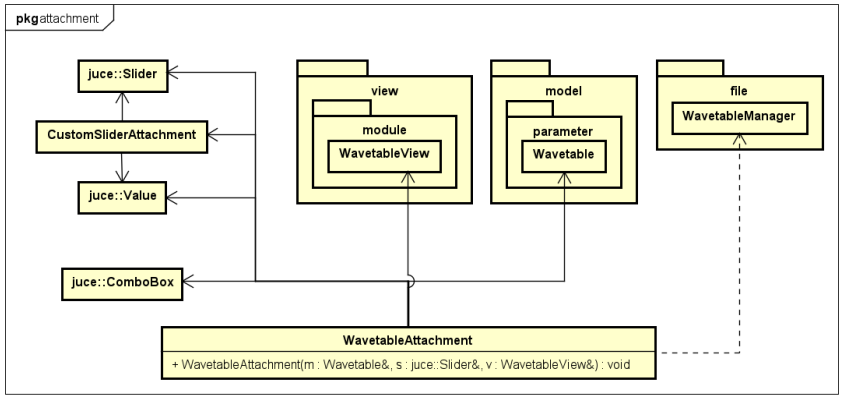


Figura 8.27: Diagrama de clases del attachment de wavetable

8.5.3. Paquete model

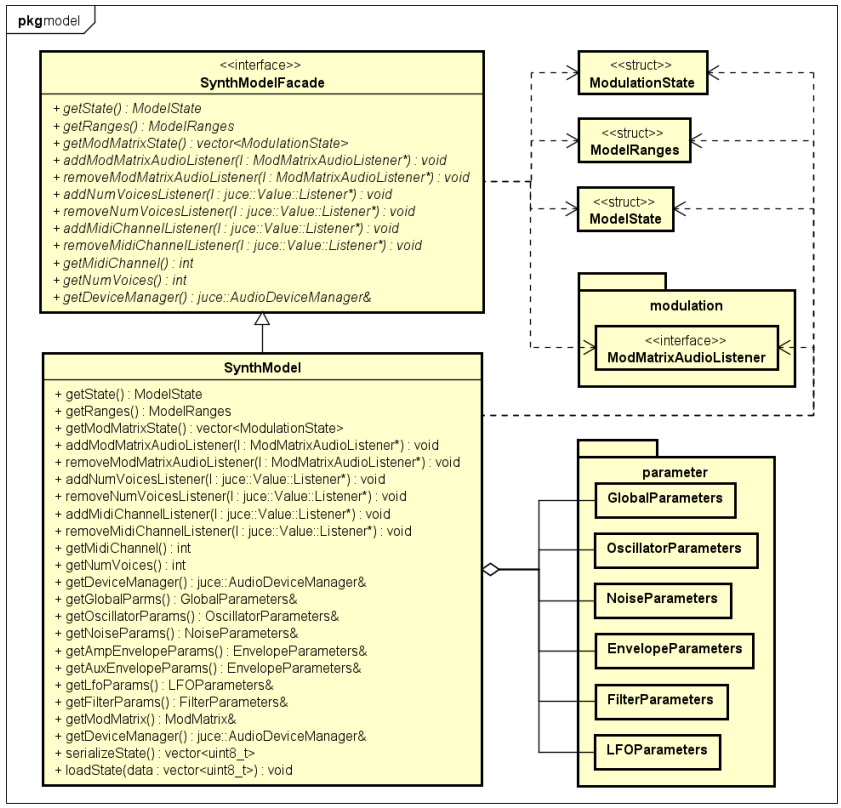


Figura 8.28: Diagrama de clases del paquete model

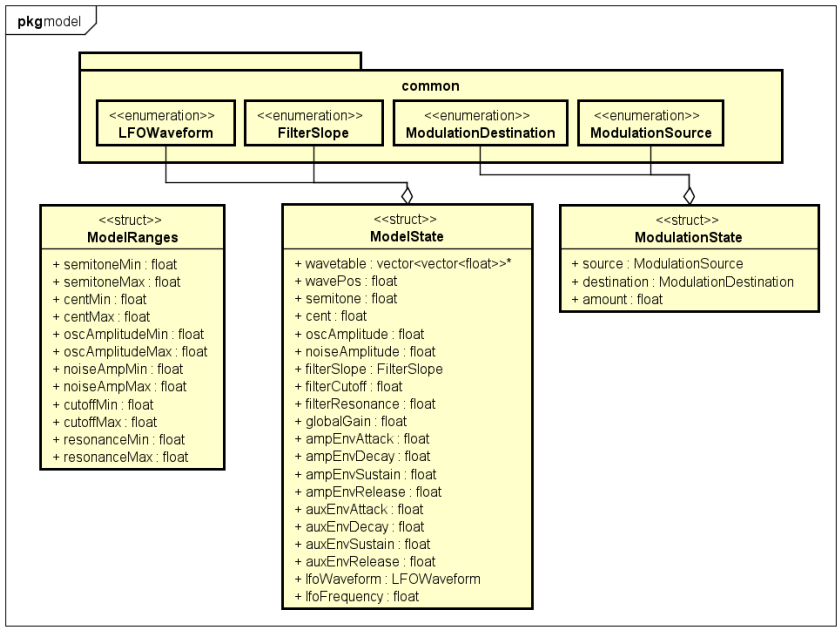


Figura 8.29: Diagrama de structs del paquete model

Subpaquete parameter

Este subpaquete contiene las clases que estructuran los parámetros que conforman el estado del modelo.

- Parámetros básicos

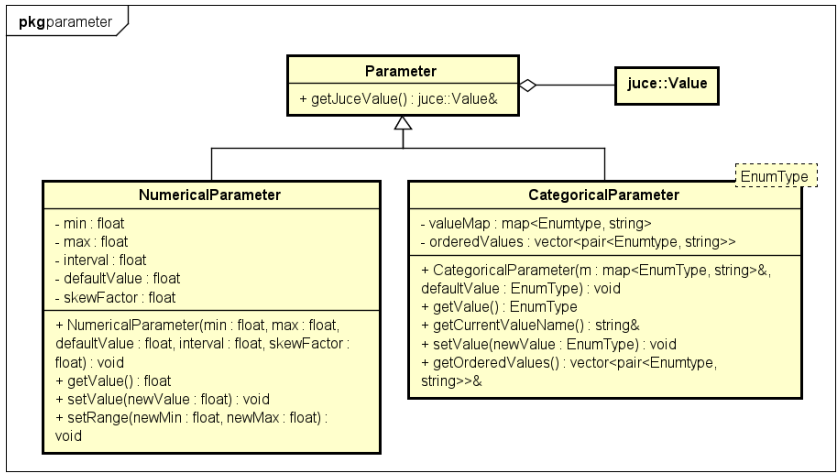


Figura 8.30: Diagrama de clases de los parámetros básicos del subpaquete parameter

- Parámetros del modelo

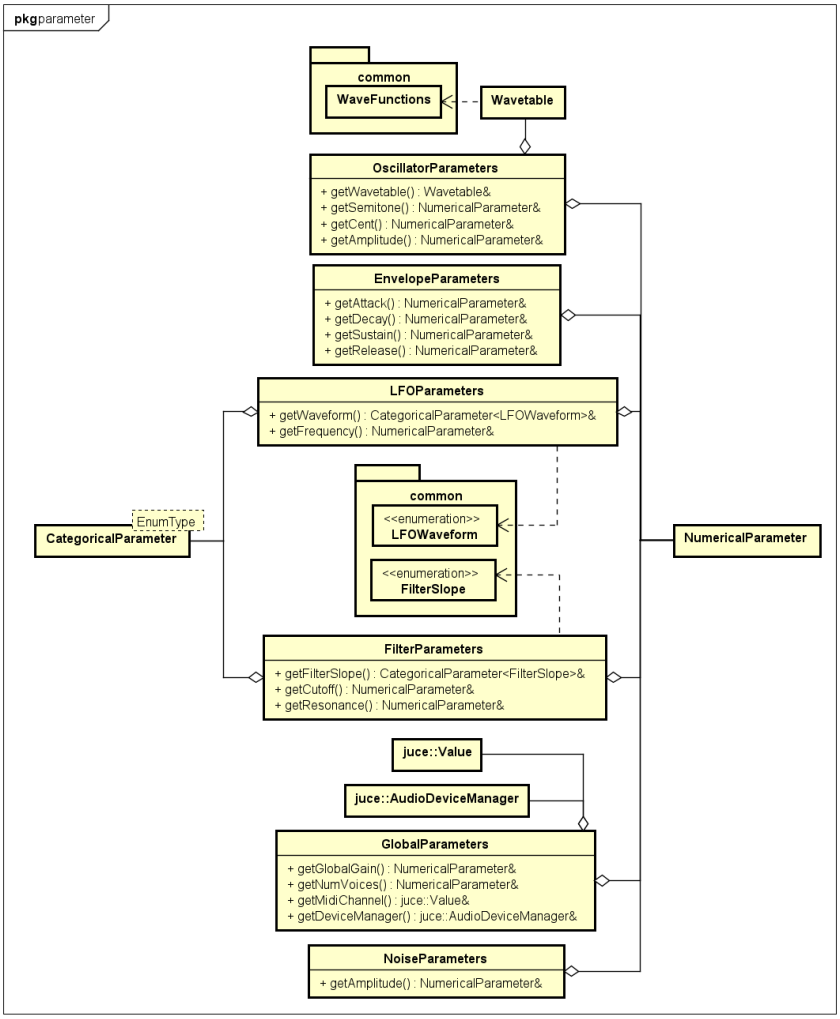


Figura 8.31: Diagrama de clases del subpaquete parameter

Subpaquete modulation

Este subpaquete contiene las clases relativas al estado de la matriz de modulación y sus modulaciones. Cabe notar que se utilizan dos tipos diferentes de listeners, uno que se utilizará para conectar el modelo con la vista y otro para notificar al sistema de audio debido a que la gestión de modulaciones es un proceso que puede ser computacionalmente exigente en comparación con los demás y requiere un tratamiento más personalizado.

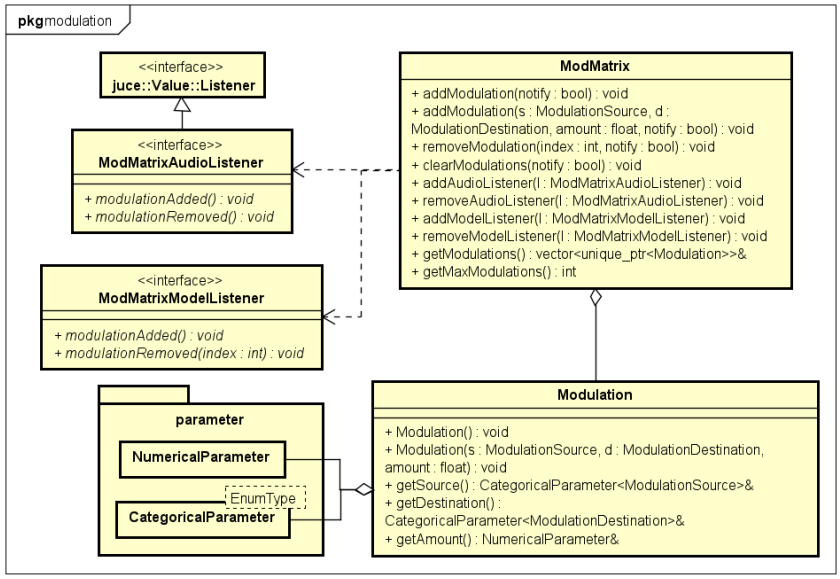


Figura 8.32: Diagrama de clases del subpaquete modulation

8.5.4. Paquete audio

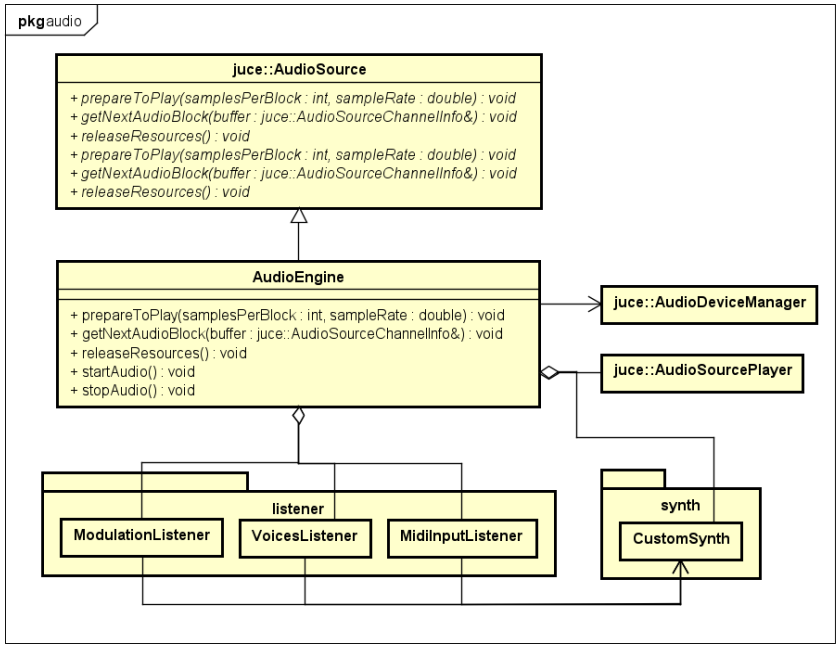


Figura 8.33: Diagrama de clases del paquete audio

Subpaquete processor

Este subpaquete contiene las clases relativas a procesadores de audio, módulos que crean o modifican una señal.
Es necesario que cada procesador conozca los valores mínimos y máximos de sus parámetros para poder calcular sus modulaciones.

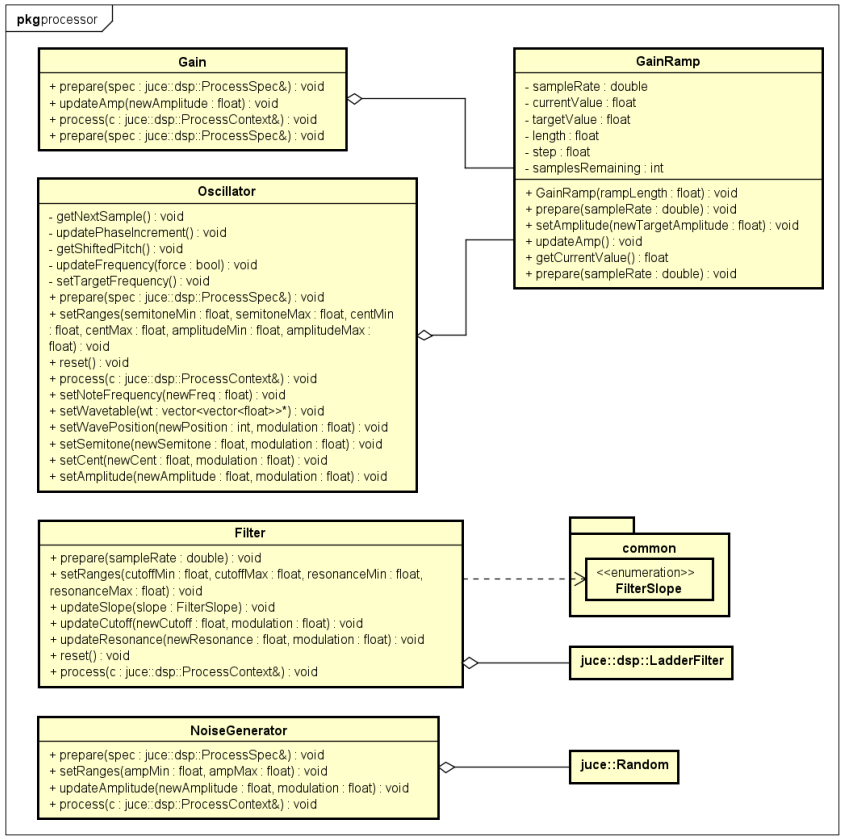


Figura 8.34: Diagrama de clases del subpaquete processor

Subpaquete modulator

Este subpaquete contiene las clases relativas a moduladores, módulos que alteran los parámetros de los procesadores.

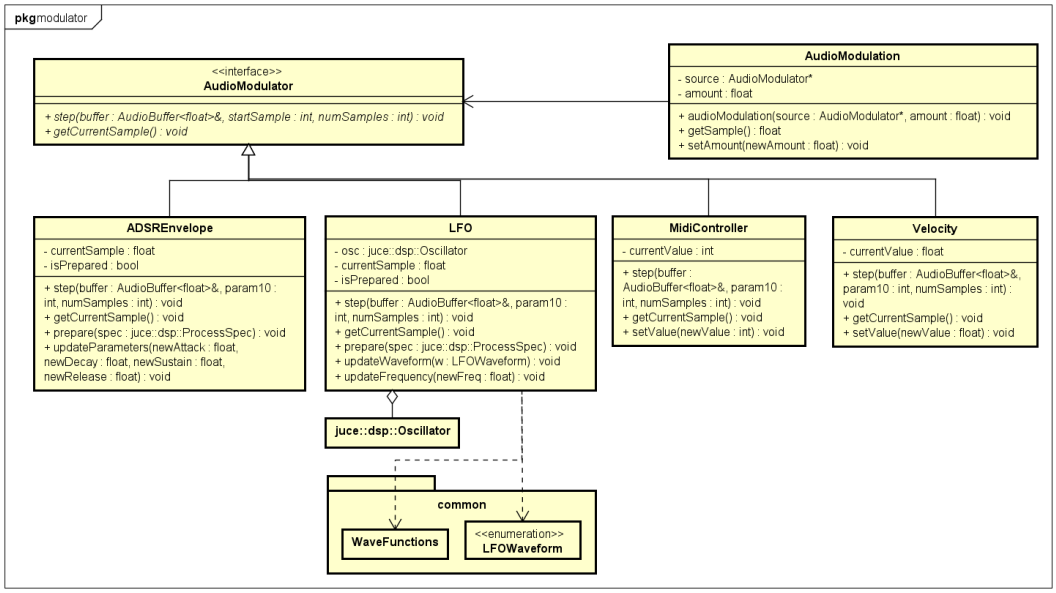


Figura 8.35: Diagrama de clases del subpaquete modulator

Subpaquete listener

Este subpaquete contiene las clases necesarias para conectar entre el modelo y el motor de audio ciertos parámetros especiales que requieren un tratamiento diferente al resto o que es demasiado costoso. Estos son:

- Número de voces. Requiere limpiar y reconstruir las voces del sintentizador. No es un parámetro que deba actualizarse en tiempo real.

- Canal MIDI. Requiere cambiar la entrada de mensajes MIDI. No es un parámetro que deba actualizarse en tiempo real.

- Matriz de modulación. Su actualización es una operación costosa y no es necesaria en tiempo real.

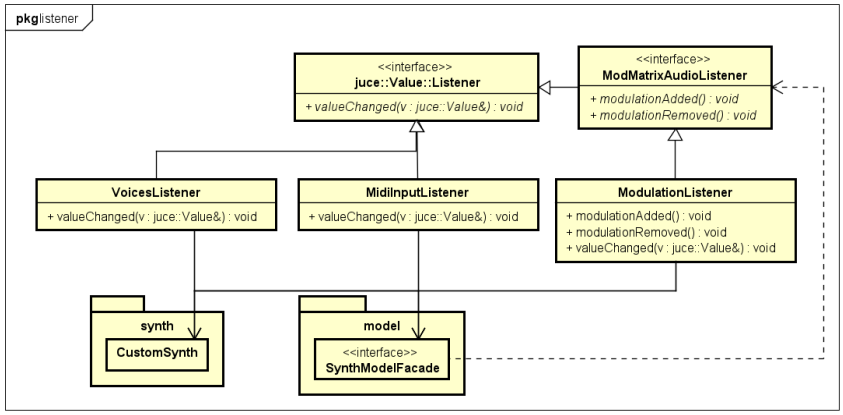


Figura 8.36: Diagrama de clases del subpaquete listener

Subpaquete synth

Este subpaquete contiene las clases principales necesarias para la construcción del sintetizador.

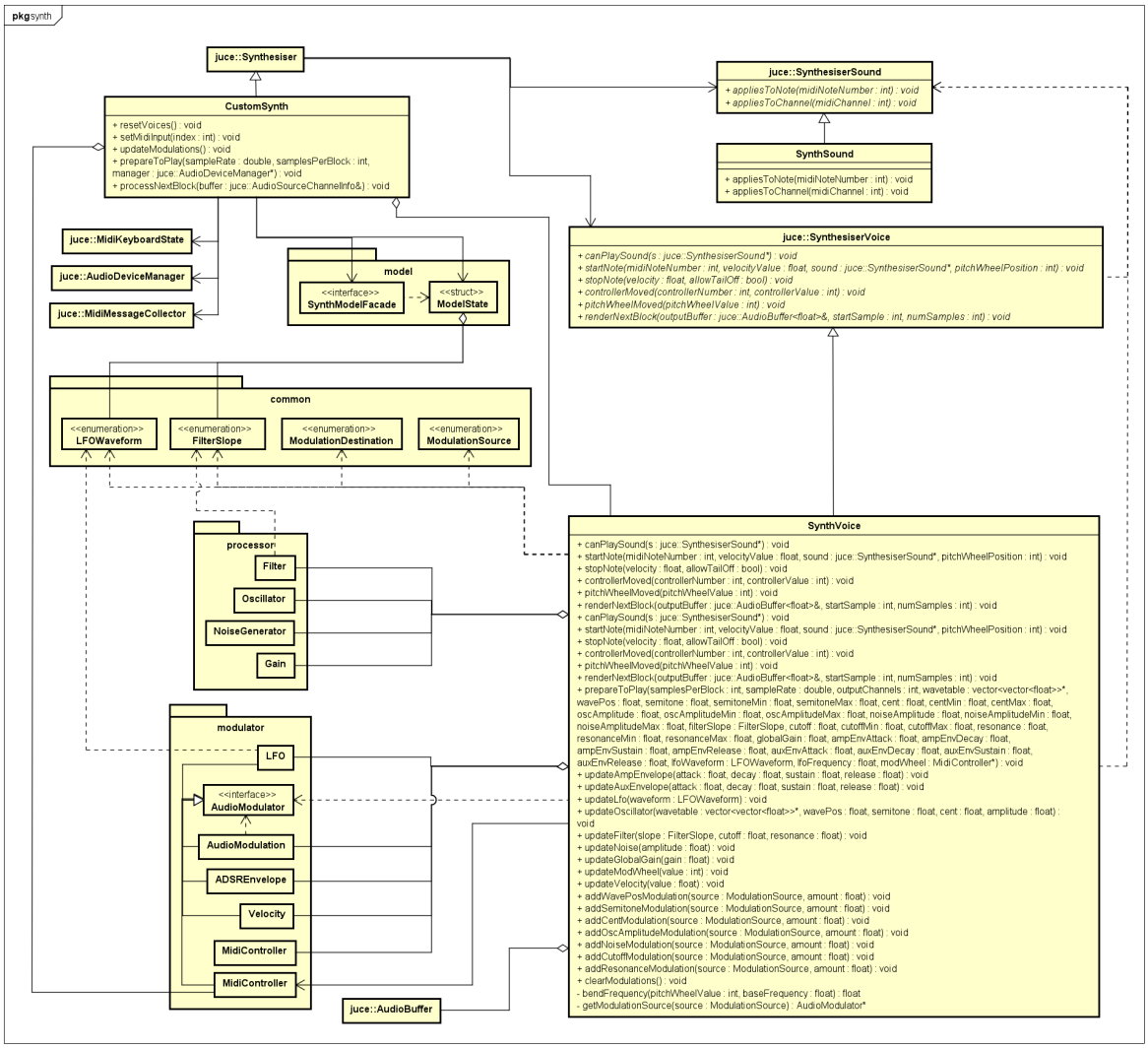


Figura 8.37: Diagrama de clases del subpaquete synth

Secuencia de procesamiento de audio

El procesamiento de audio sigue una importante secuencia lógica que se repite con cada petición del sistema de audio del próximo bloque que contiene los samples de audio a reproducir.

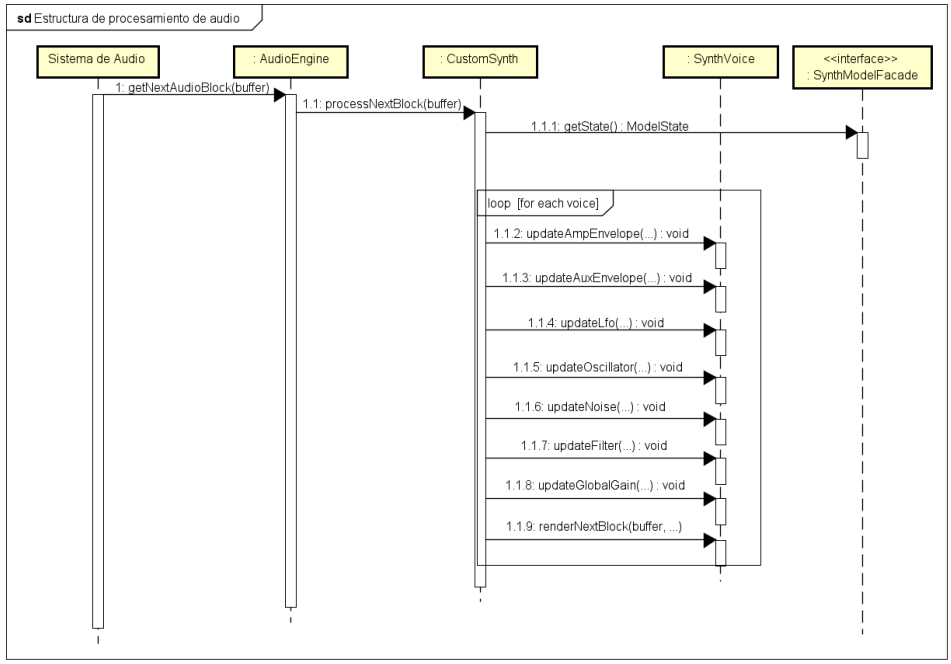


Figura 8.38: Diagrama de secuencia general del procesamiento de audio

Como se observa en la figura 8.38, el sistema de audio solicita a *AudioEngine* el siguiente bloque de audio, este a su vez se lo pide a *CustomSynth*. En el caso de añadir más componentes al sistema más allá del sintetizador como pueden ser efectos de audio, *AudioEngine* pediría el siguiente bloque de audio a estos componentes aquí también.
CustomSynth procesa la petición obteniendo el estado del modelo en ese instante y orquestando las voces *SynthVoice*, primero actualizándolas con los parámetros obtenidos del modelo y después pidiendo que procesen el bloque de audio.

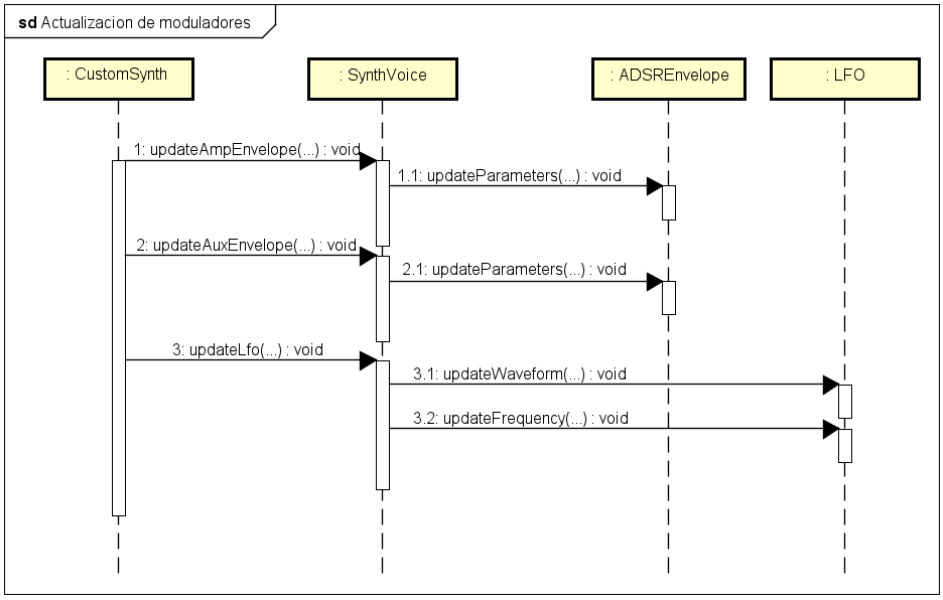


Figura 8.39: Diagrama de secuencia de la actualización de los moduladores de una voz

La figura 8.39 muestra lo que ocurre cuando *CustomSynth* solicita a una voz que actualice el estado de sus moduladores. Cada voz *SynthVoice* transfiere la petición a cada uno de sus moduladores con los valores actualizados del modelo enviados por *CustomSynth*.

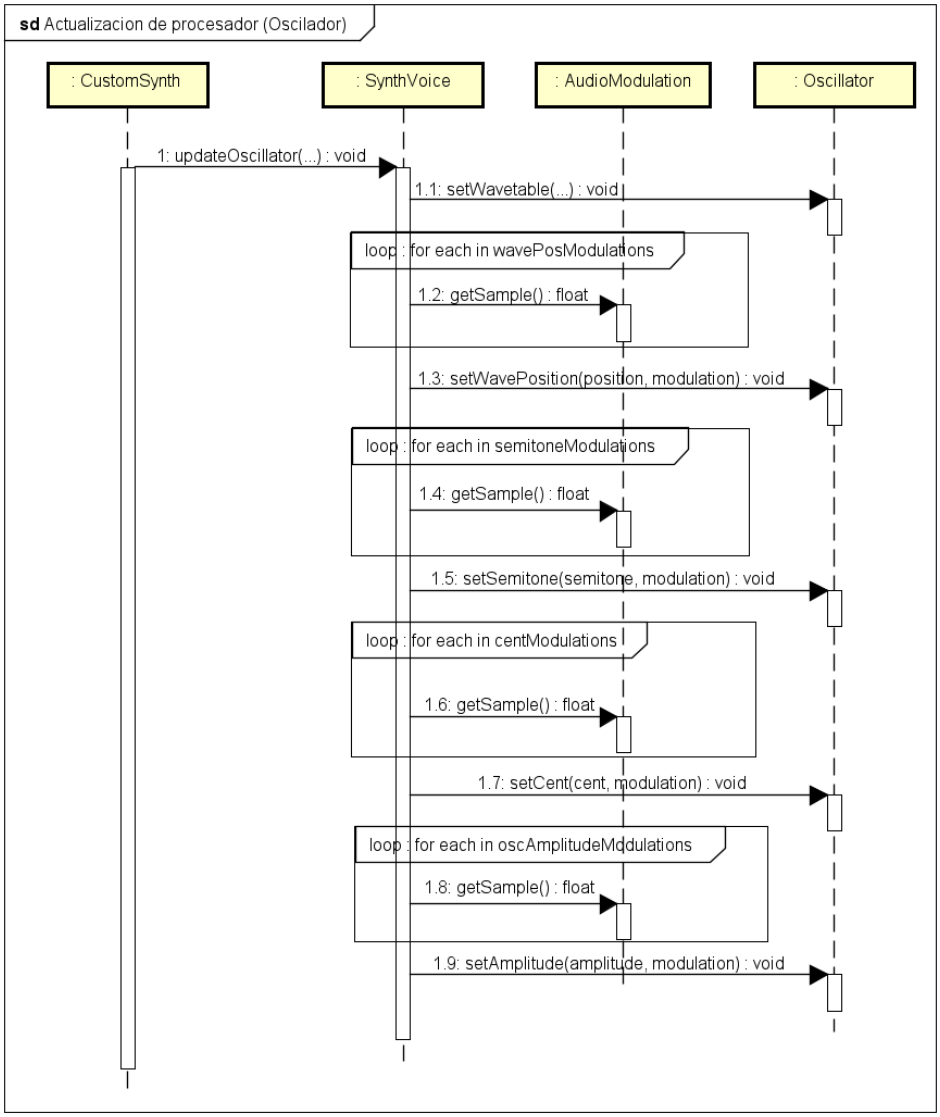


Figura 8.40: Diagrama de secuencia de la actualización de los procesadores de una voz

De manera similar cada voz actualiza el estado de sus procesadores con los nuevos valores del modelo, pero calculando previamente el valor total de modulación de cada parámetro sumando el valor de cada modulación. Por ejemplo, si el parámetro *cent* del oscilador está siendo modulado un 5 % por el modulador *auxEnvelope* y un -25 % por el modulador *LFO*, la modulación total del parámetro *cent* será de -20 %.
Este proceso se describe en la figura 8.40 para la actualización del oscilador. El resto de procesadores de audio siguen un proceso de actualización similar.

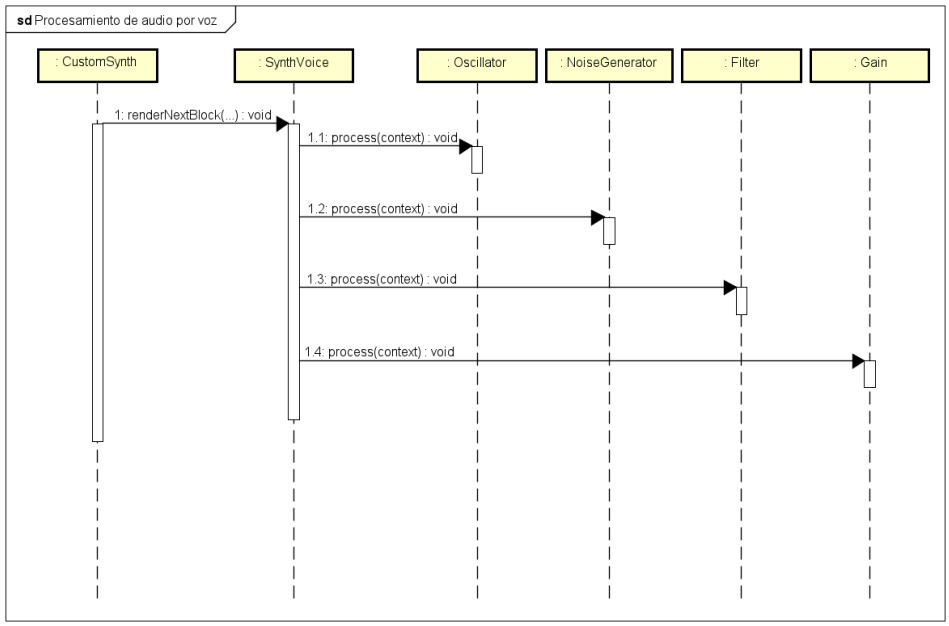


Figura 8.41: Diagrama de secuencia del procesamiento de un bloque de audio por una voz

Por último, como describe la figura 8.41, la voz frente a la petición de procesar un bloque de audio, a su vez orquesta sus procesadores de audio solicitándoles que procesen el bloque de audio. El bloque de audio sigue el siguiente procesamiento:

1. El oscilador genera los valores correspondientes y los almacena en el bloque de audio reemplazando su contenido.

2. El generador de ruido añade ruido al contenido del bloque de audio.

3. El filtro filtra las frecuencias altas del contenido del bloque de audio.

4. La ganancia ajusta la amplitud del contenido del bloque de audio.

8.5.5. Paquete common

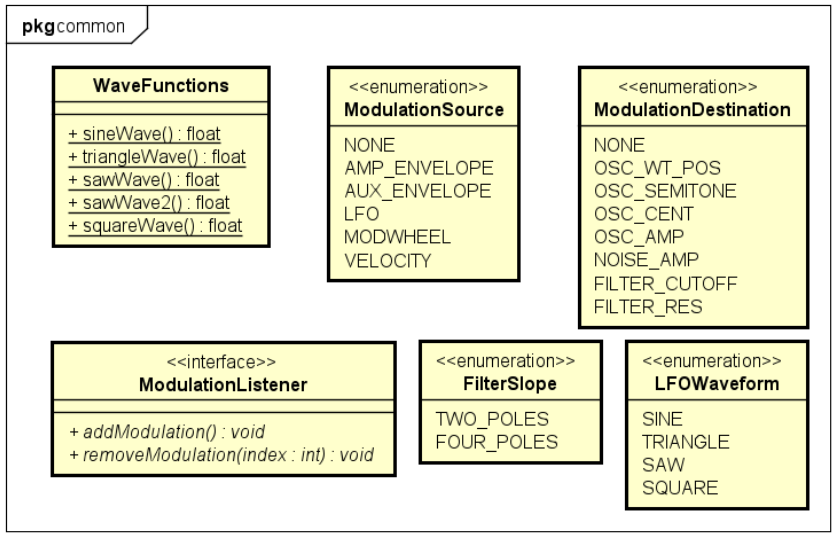


Figura 8.42: Diagrama de clases del paquete common

8.5.6. Paquete file

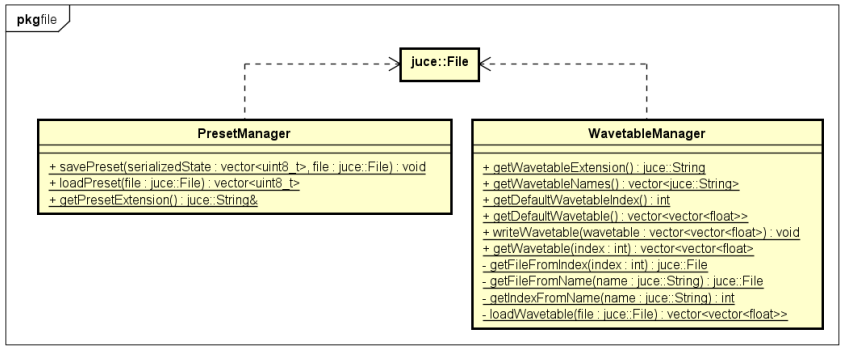


Figura 8.43: Diagrama de clases del paquete file

Capítulo 9

Implementación

9.1. Formación previa

Debido a la utilización del framework JUCE, el cual requiere un conocimiento de sus herramientas, y a la falta de experiencia con este. Ha sido necesaria una formación previa a la implementación de la aplicación. Esta formación ha consistido en realizar todos los tutoriales oficiales de JUCE [18].

9.2. Configuración inicial

Para implementar una aplicación JUCE, nos apoyamos en la herramienta PROJUCER que viene incluida en el framework.

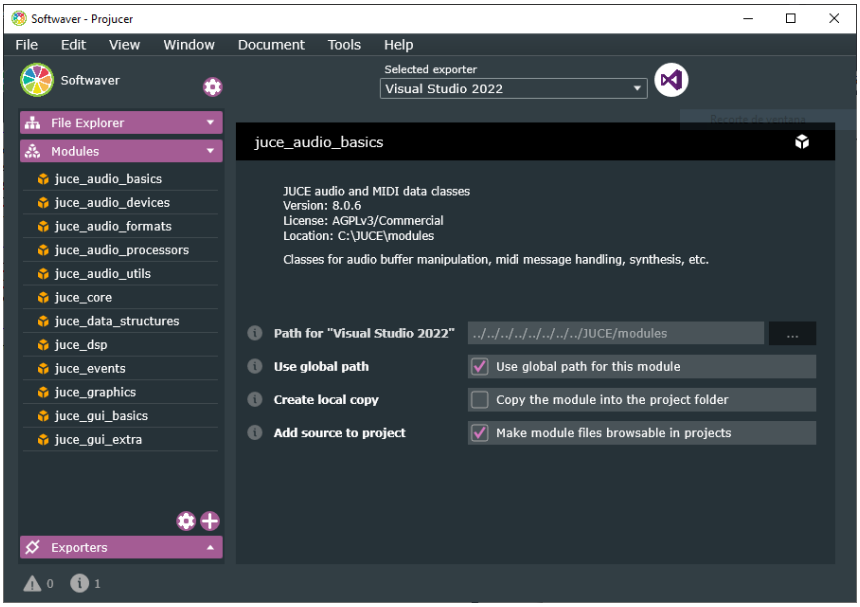


Figura 9.1: Projucer [19]

Al crear un proyecto desde Projucer, este se encarga de la configuración del entorno, ofreciendo diferentes plantillas dependiendo del tipo de aplicación que se desea desarrollar. Para este caso, se ha utilizado la plantilla *Audio Application*.
Adicionalmente, Projucer permite gestionar los módulos de JUCE que se necesitan importar en el proyecto. En este caso, además de los módulos incluidos por defecto al utilizar la plantilla seleccionada, se ha importado el módulo *juce_dsp* para tener acceso a algunas herramientas específicas de procesamiento de señales digitales[1]. Todos los módulos utilizados se muestran en la figura 9.1.
Desde Projucer también se gestionan los ficheros y paquetes, facilitando su creación y destrucción, pero no su edición, la cual se hace con Visual Studio Code. Projucer automáticamente se conecta con Visual Studio y lo configura para trabajar en el proyecto deseado, para ello es importante ejecutar Visual Studio desde Projucer.

9.3. Interfaz gráfica de usuario

La implementación de la interfaz gráfica de usuario se ha apoyado principalmente en la clase *Component* del framework JUCE. Esta es la clase base para todos los objetos de la interfaz de usuario [14].
Estos son sus métodos principales que se han utilizado:

- *addAndMakeVisible*: Añade un componente hijo al componente llamado siguiendo un

[1]Las siglas DSP corresponden con Digital Signal Processing

patrón de diseño *Composite*.

- *paint*: Dibuja el componente en pantalla.

- *resized*: Este método es llamado cada vez que el tamaño del componente cambia. En él, se determina la disposición de los componentes que contiene el componente padre. La implementación de este método en cada módulo de la interfaz gráfica se ha hecho de tal manera que permita el ajuste del tamaño de la ventana de la aplicación manteniendo las proporciones de todos los elementos visuales.

- *setBounds*: Establece los límites de un componente en relación a su componente padre que lo contiene. Normalmente este método es llamado desde el método *resized* del componente padre.

```
void LFOView::resized()
{
    ModuleView::resized();

    int sideMargin = getWidth() * ViewModuleConstants::Lfo::MARGIN_PROP;
    int halfHeight = (getHeight() - titleLabel.getHeight()) / 2;
    int buttonHeight = halfHeight * ViewModuleConstants::Lfo::BUTTON_HEIGHT_PROP / 2;
    int width = getWidth() - sideMargin * 2;
    int freqSliderHeight = halfHeight * ViewModuleConstants::Lfo::FREQSLIDER_HEIGHT_PROP;
    int freqSliderHPos = titleLabel.getHeight() + halfHeight;

    sineButton.setBounds(sideMargin, titleLabel.getHeight(), width / 2, buttonHeight);
    triangleButton.setBounds(sideMargin + width / 2, titleLabel.getHeight(), width / 2, buttonHeight);
    sawButton.setBounds(sideMargin, titleLabel.getHeight() + buttonHeight, width / 2, buttonHeight);
    squareButton.setBounds(sideMargin + width / 2, titleLabel.getHeight() + buttonHeight, width / 2, buttonHeight);
    waveformLabel.setBounds(sideMargin, titleLabel.getHeight() + buttonHeight * 2, width, halfHeight - buttonHeight * 2);

    freqSlider.setBounds(sideMargin, freqSliderHPos, width, freqSliderHeight);
    freqLabel.setBounds(sideMargin, freqSliderHPos + freqSliderHeight, width, halfHeight - freqSliderHeight);
}
```

Figura 9.2: Ejemplo del método resized. Nótese las llamadas al método setBounds de los componentes hijos

Como se ha descrito en el capítulo de diseño, todas las clases relativas a componentes de la interfaz gráfica de usuario heredan de la clase *Component*. A su vez, estas clases utilizan las clases *Slider, Button, ComboBox* del framework Juce que son también herederas de la clase *Component*.

Por otro lado, para la clase *Skin* principal del paquete *theme* se ha utilizado *LookAndFeel* como clase base. Esto permite establecer una apariencia homogénea a lo largo de la aplicación, determinando la forma de dibujar ciertos componentes como *Sliders, Buttons* o *ComboBoxes*[16].

```
void drawRotarySlider(juce::Graphics& g, int x, int y, int width, int height,
    float sliderPos, float rotaryStartAngle, float rotaryEndAngle,
    juce::Slider& slider) override;

void drawLinearSlider(juce::Graphics& g, int x, int y, int width, int height,
    float sliderPos, float minSliderPos, float maxSliderPos,
    const juce::Slider::SliderStyle style, juce::Slider& slider) override;

void drawToggleButton(juce::Graphics& g, juce::ToggleButton& button,
    bool shouldDrawButtonAsHighlighted,
    bool shouldDrawButtonAsDown) override;
```

Figura 9.3: Algunos métodos de la clase Skin heredados de LookAndFeel

Finalmente, la interfaz gráfica de usuario se ha implementado siguiendo el diseño previsto.

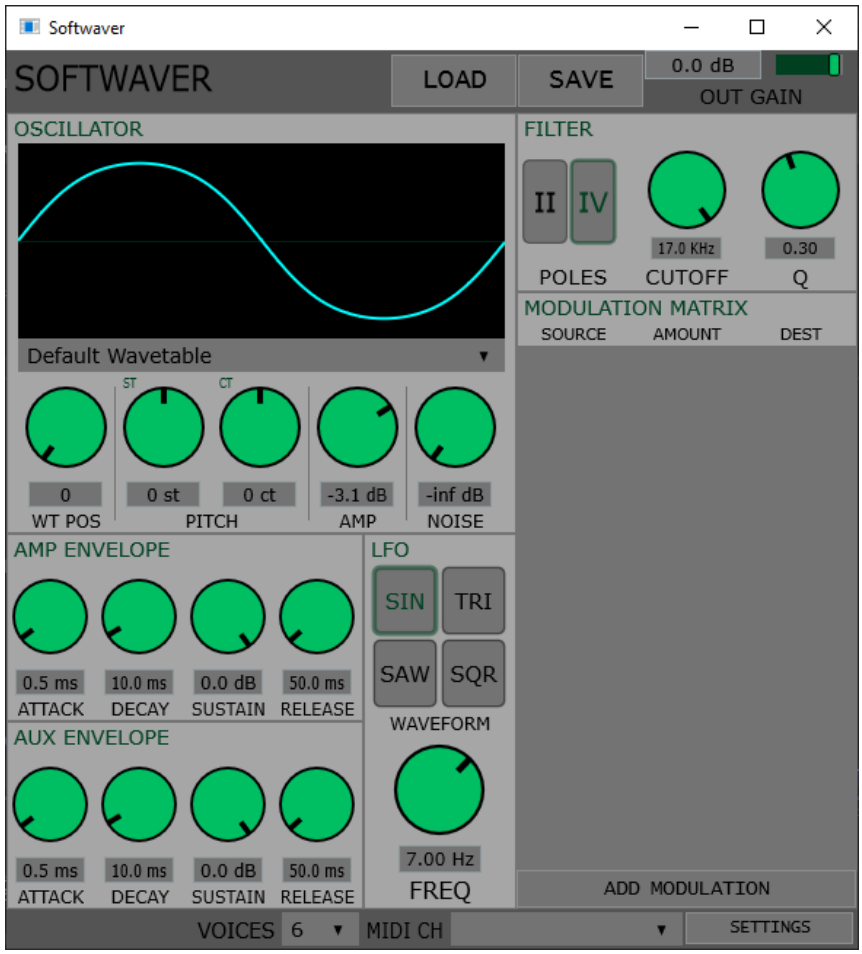


Figura 9.4: Interfaz gráfica de usuario de la aplicación implementada

9.4. Modelo

La implementación del modelo se apoya principalmente en las subclases de *Parameter*: *NumericalParameter* y *CategoricalParameter*. Estas clases almacenan un valor, su rango y la manera en la que cambia.
Como se ha descrito en el capítulo de Diseño, los parámetros de la aplicación están separados por módulos. Cada clase correspondiente a un módulo contiene varios objetos *NumericalParameter* o *CategoricalParameter*.

9.4.1. Wavetable

Un caso especial es la implementación de la wavetable. Una wavetable es una lookup table aplicada a formas de onda para síntesis de audio. Este principio es explotado para aumentar la capacidad de síntesis, almacenando un número elevado de formas de onda (entre 128 y 1024), cada una con su lookup table, creando así una matriz donde cada fila corresponde con una forma de onda. Normalmente las formas de onda se almacenan en un orden tal que las filas contiguas contienen formas de onda más similares lo que provoca una transición suave y fluida del sonido al cambiar la fila de la forma de onda a reproducir. La forma de onda seleccionada se elige con el parámetro *position* que simplemente determina el índice de una fila.

Para asegurar que el sistema siempre tiene una wavetable disponible, se ha implementado la generación de una wavetable por defecto, la cual consiste en situar en la matriz de formas de onda que es la wavetable cuatro formas de onda básicas: seno, triangular, cuadrada y diente de sierra; en filas igualmente distanciadas y después rellenando las filas entre cada par de formas de onda interpolando linealmente para generar una transición entre ellas.

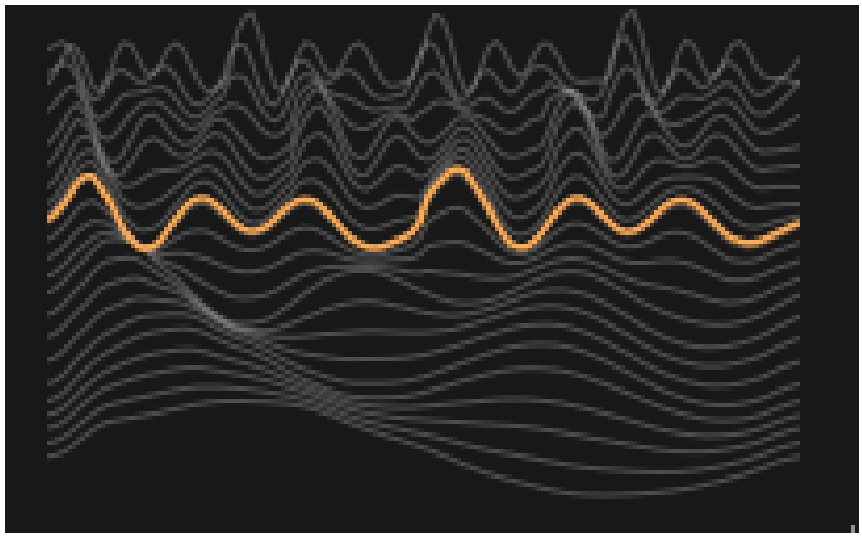


Figura 9.5: Ejemplo de representación gráfica de una wavetable. La forma de onda seleccionada se representa en amarillo [2]

9.4.2. Serialización

Se ha implementado la serialización y deserialización del estado del modelo, lo que supone codificar este en un vector de bytes y poder interpretar un vector de bytes como un estado del modelo. Esto permite, entre otras cosas, la lectura y almacenamiento de presets en archivos binarios.
La implementación de esta caracterísitca se ha logrado a través de los métodos *serializeState* y *loadState* de la clase *SynthModel*. Estos métodos se encargan de convertir los valores de los parámetros del estado del modelo a un vector de bytes siguiendo siempre un determinado orden para garantizar un buen formato del vector.
Como caso especial está la serialización del estado de la matriz de modulación ya que el número de modulaciones es variable. Para abordar esto, se almacena en el vector primero un número que indica el número de modulaciones existentes.
Al deserializar un vector de bytes con el método *loadState*. Antes de actualizar el estado del modelo, se comprueba que el tamaño del vector es el correcto y que todos los parámtros leídos son válidos. En caso de no satisfacerse estos requisitos, el estado del modelo se mantiene como estaba previamente y se notifica al usuario del fallo en la lectura del preset.

9.5. Audio

9.5.1. Breve introducción al procesamiento de audio digital

Para programar herramientas de procesamiento de audio digital, es esencial comprender cómo se representa y manipula una señal de audio en el dominio digital. Para representar digitalmente una señal analógica continua primero debe ser muestreada, es decir, convertida en una secuencia de valores discretos en el tiempo. Esto implica seleccionar una frecuencia de muestreo (como 44.1 kHz o 48 kHz), que define cuántas veces por segundo se toma una muestra de la señal, y una profundidad de bits (como 16 o 32 bits), que determina la precisión con la que se codifica la amplitud de cada muestra.
Finalmente, una señal de audio digital es un vector de números, llamados muestras (samples), que representan el valor de la señal en cada instante discreto de tiempo. Estos valores suelen estar normalizados dentro del intervalo $[-1, 1]$, donde el signo indica la dirección de desplazamiento respecto al punto de reposo, y su magnitud está relacionada con la amplitud relativa del sonido.

9.5.2. Oscilador

El oscilador se encarga de leer muestras de la wavetable de una forma determinada y variable en función de la frecuencia y amplitud del sonido a generar.

Lectura de wavetable

El vector seleccionado por la posición en la wavetable contiene un ciclo de una forma de onda. El oscilador va leyendo las muestras de este vector repetidamente, multiplicándolas por el valor de la amplitud que está contenido en el rango $[0, 1]$ y escribiendo el resultado en el buffer.
Para ajustar la frecuencia del sonido obtenido, el oscilador debe iterar el vector de la forma de onda tantas veces por segundo como hercios tenga la frecuencia. Para lograr esto, calcula un incremento de fase de la onda, es decir, un incremento del índice de lectura del vector:

$$\Delta\phi = \frac{f \cdot N}{f_s}$$

donde:

- $\Delta\phi$ es el incremento de fase por muestra (en posiciones del vector),
- f es la frecuencia deseada del sonido (en Hz),
- N es el tamaño del vector de forma de ondas (número de muestras),
- f_s es la frecuencia de muestreo (en Hz).

Para cada lectura de una muestra, el oscilador calcula la posición en el vector donde se encuentra esta sumando $\Delta\phi$ a la posición anterior. Esta operación se realiza en forma cíclica, aplicando el módulo con respecto al tamaño del vector N. Cuando la posición obtenida no es un número entero, calcula la muestra resultante interpolando linealmente entre las dos muestras adyacentes.

Amplitud

La amplitud del oscilador es simplemente un valor contenido en el intervalo [0, 1].
El cambio de este valor requiere un tratamiento especial ya que cambios repentinos pueden producir discontinuidades de fase o saltos abruptos en la forma de onda, lo que genera ruidos indeseados llamados artefactos (artifacts).
Para evitar esto se emplea la estrategia conocida como *ramping*.
El *ramping* consiste en realizar la transición entre el valor actual y el nuevo valor deseado de forma progresiva, a lo largo de un número determinado de muestras. En lugar de aplicar el nuevo valor inmediatamente, se calcula un pequeño incremento que se suma en cada iteración del procesamiento de una muestra de audio, suavizando así la variación.
Este procedimiento reduce significativamente los artefactos que pueden aparecer en el audio, especialmente cuando los cambios de amplitud son controlados desde la interfaz gráfica o desde un mensaje MIDI.
Para calcular el incremento del valor por cada muestra procesada se calcula el número de muestras que supone la duración del *ramping* y se interpola linealmente entre el valor actual y el valor objetivo:

$$\Delta V = \frac{V_1 - V_0}{t \cdot f_s}$$

donde:

1. ΔV es el incremento del valor a sumar al procesar una muestra.
2. V_1 es el valor objetivo.
3. V_0 es el valor inicial.
4. t es la duración del ramping (en segundos).
5. f_s es la frecuencia de muestreo (en Hz).

Frecuencia

La actualización de la frecuencia sigue un proceso similar al de la amplitud, aplicando *ramping* con el añadido de que es necesario mantener la posición relativa en la forma de onda al cambiar el incremento de fase (cambiar la frecuencia).

Como se ha descrito anteriormente, cuando se cambia la frecuencia de un oscilador, también cambia el valor del incremento de fase que se utiliza para recorrer el vector de forma de onda. Si este cambio se aplicara directamente, se produciría una discontinuidad en la señal, ya que el nuevo incremento leería una muestra distinta a la que le correspondería según la progresión anterior. Esto genera artefactos audibles como clics o transitorios indeseados.

Para evitarlo, se ajusta el valor del acumulador de fase mediante un factor de escala que mantiene la misma posición relativa dentro de la forma de onda. Este factor es el cociente entre la nueva frecuencia deseada y la frecuencia actual:

$$\text{factor} = \frac{f_{\text{nueva}}}{f_{\text{actual}}}$$

Aplicando este factor al acumulador de fase se consigue que, pese a cambiar el tamaño del paso (el incremento de fase), el oscilador siga apuntando a la misma proporción dentro del ciclo de la onda (por ejemplo, al 25 % del ciclo, independientemente de la longitud en muestras del nuevo incremento).

De este modo, el cambio de frecuencia no genera saltos perceptibles en la señal y la transición es suave. Además, para calcular la frecuencia objetivo, se utilizan los valores de *semitone* y *cent* para alterar una frecuencia base.

9.5.3. Generador de ruido

El ruido tiene una estrecha relación con la aleatoriedad. En este caso, el generador de ruido produce ruido blanco. Este tipo de ruido se caracteriza por tener la misma cantidad de energía por frecuencia. Para producir este ruido blanco, se generan muestras con valores aleatorios igualmente distribuidos en el intervalo $[-1, 1]$. El generador de ruido suma estos valores al contenido del buffer aplicando lo que se denomina *softclip* para mantener el valor resultante en el intervalo $[-1, 1]$. Para hacer *softclip* simplemente basta con calcular la tangente hiperbólica del valor resultante de sumar el ruido. Al hacer *softclip*, a parte de mantener el valor de las muestras en su rango correspondiente, se añade una suave saturación a la señal añadiendo contenido armónico previamente inexistente.

Para modificar la amplitud del ruido se aplica *ramping* de igual forma que en la amplitud del oscilador. La clase *GainRamp* se encarga de encapsular esta técnica de *ramping*.

9.5.4. Filtro

El procesamiento del filtro se delega a la clase del framework JUCE *LadderFilter*. La clase *Filter* simplemente se encarga de gestionar los parámetros del filtro.

La clase *LadderFilter* implementa un filtro inspirado en el filtro *ladder* diseñado por Robert Moog. [15]

9.5.5. Envelope

Un envelope ADSR (Attack, Decay, Sustain, Release) es un tipo de modulador que controla cómo evolucionan los parámetros que modula a lo largo del tiempo en cuatro etapas:

1. Attack: Define el tiempo que tarda el envelope en alcanzar la cantidad de modulación máxima después de presionar una tecla o activar una nota.
2. Decay: Es el tiempo que tarda el envelope en pasar de la cantidad de modulación máxima al nivel de sustain.
3. Sustain: Es la cantidad de modulación relativa al máximo que se mantiene mientras la nota se sostenga.
4. Release: Es el tiempo que tarda la cantidad de modulación en pasar del nivel de sustain a 0 cuando la nota termina.

La cantidad máxima de modulación viene determinada por el parámetro *amount*.

De igual manera que en el caso del filtro, el procesamiento del envelope se ha delegado a la clase del framework JUCE *ADSR*. La clase *Envelope* se encarga de controlar los parámetros del envelope y actualizar su posición temporal mediante el método *step*.

9.5.6. LFO

LFO son las siglas en inglés para *Low Frequency Oscillator*, es decir, oscilador de baja frecuencia. Como su nombre indica, es un oscilador pero que genera señales a frecuencias inferiores al rango audible humano. Pero, a diferencia del oscilador principal, los valores que genera este oscilador no se utilizan directamente para producir sonido, sino que se utilizan para modular el valor de algún parámetro.
La clase *LFO* utiliza la clase del framework JUCE *Oscillator* y controla sus parámetros. [17]

9.5.7. Control MIDI

La clase *MidiController* representa un control MIDI como puede ser una perilla, un deslizador o la rueda de modulación, aunque en esta primera versión de la aplicación sólo se permite la utilización de la rueda de modulación.
Utilizando el valor de la posición del control, calcula el valor de modulación para alterar otro parámetro.

Cuando un control MIDI cambia su posición, genera un mensaje MIDI capturado por el métoddo de *SynthVoice controllerMoved*. Este actualiza el objeto *MidiController* correspondiente utilizando el valor de posición del control.
Es importante no confundir los términos control MIDI y controlador MIDI (llamados igual en inglés). El controlador MIDI hace referencia a toda la pieza hardware que contiene el teclado y varios controles MIDI.

9.5.8. Velocity

El parámetro MIDI *velocity* hace referencia a la fuerza con la que se presiona una tecla. Para calcular esta fuerza, el controlador MIDI mide el tiempo que dura el desplazamiento de la tecla desde que empieza a ser presionada hasta que toca el fondo y traduce este tiempo a un valor numérico similar al valor de posición de un control MIDI.
La clase *Velocity* se encarga de gestionar este parámetro. Inicialmente se planteó su gestión de una manera más general utilizando la clase *MidiController*, pero el funcionamiento y flujo de procesamiento de audio de JUCE instaba a utilizar una clase separada.

9.5.9. Pitch wheel

La pitch wheel es un tipo especial de control MIDI. No se utiliza como modulador general sino como uno dedicado a modular la frecuencia del oscilador. Su valor de posición también es diferente al de un control MIDI genérico, proporcionando mayor precisión.
Los mensajes MIDI del movimiento de la pitch wheel capturados por el método de *SynthVoice pitchWheelMoved* modifican directamente la frecuencia objetivo del oscilador.

9.5.10. Notas MIDI

Cuando se presiona una tecla del controlador MIDI, este envía un mensaje MIDI con un valor indicativo de la nota presionada y su velocity. Estos valores llegan hasta el método de *SynthVoice startNote*. Desde este método, se actualiza la frecuencia objetivo del oscilador, se indica a los envelopes que inicien su fase attack y se actualiza el valor del objeto *velocity*.

De manera similar, cuando la tecla presionada se levanta el controlador MIDI envía un mesaje con un valor indicativo de la nota liberada. Este valor llega al método de *SynthVoice stopNote* donde se indica a los envelopes que entren en su fase release. Desde este método también se libera la voz para que esté disponible para procesar otras notas. Pero para poder liberar la voz una vez hayan terminado los envelopes su fase release, se utiliza un parámetro booleano *isTailOff* que mantiene la voz activa hasta entonces.

9.6. Valores constantes

Para evitar números mágicos y tener un uso homogéneo de valores constantes se han creado archivos contenedores de los valores constantes por paquete.

Capítulo 10

Pruebas

10.1. Aserciones

Se han usado aserciones utilizando las instrucciones *jassert* del framework JUCE para asegurar que el funcionamiento esperado se cumple.
El beneficio que proporcionan estas instrucciones *jassert* es que sólo son compiladas si hay un debugger asociado, lo que permite comprobar el correcto funcionamiento durante la implementación, pero evitando una carga de cómputo innecesaria durante la ejecución de la aplicación final.
Estas instrucciones *jassert* comprueban una sentencia lógica y detienen la ejecución si esta es falsa.

```
void NumericalParameter::setValue(float newValue)
{
    jassert(newValue <= max);
    jassert(newValue >= min);
    value.setValue(newValue);
}
```

Figura 10.1: Ejemplo de uso de instrucciones jassert

10.2. Pruebas

Se han realizado las siguientes pruebas para evaluar el comportamiento del sistema.

Tabla 10.1: Prueba P-1

Código	**P-1**
Prueba	Mover cada deslizador de la interfaz gráfica de usuario en todo su rango.
Resultado	Los parámetros y la interfaz se actualizan debidamente. Al alcanzar el máximo del control waveposSlider, la aplicación crashea.
Evaluación	Negativa para el control waveposSlider. Positiva para el resto.
Motivo del problema	El rango de posiciones de la wavetable se obtiene a partir del número de filas de esta. Se estaba asignando como valor máximo el número de filas.
Corrección	Restar 1 al número de filas al calcular el rango de posiciones de la wavetable.

Tabla 10.2: Prueba P-2

Código	**PA-2**
Prueba	Seleccionar valores límite de cada combo box.
Resultado	Los parámetros y las interfaz se actualizan debidamente.
Evaluación	Positiva.

Tabla 10.3: Prueba P-3

Código	**PA-3**
Prueba	Introducir textualmente valores límite para cada deslizador.
Resultado	Los parametros y la interfaz se actualizan debidamente.
Evaluación	Positiva.

Tabla 10.4: Prueba P-4

Código	**PA-4**
Prueba	Introducir textualmente valores fuera de su rango en cada deslizador.
Resultado	El parámetro se ajusta a su límite inferior cuando el valor introducido es menor que este y a su límite superior cuando el valor introducido es mayor que este. La interfaz se actualiza debidamente.
Evaluación	Positiva.

Tabla 10.5: Prueba P-5

Código	**PA-5**
Prueba	Hacer doble click sobre un parámetro para restablecer su valor por defecto.
Resultado	El parámetro se actualiza a su valor por defecto. La interfaz se actualiza debidamente.
Evaluación	Positiva.

Tabla 10.6: Prueba P-6

Código	**PA-6**
Prueba	Presionar simultaneamente un numero de teclas del controlador MIDI superior al numero de voces.
Resultado	Al superar el número de voces, la voz interpretando la última nota presionada cambia su nota a la nueva presionada.
Evaluación	Positiva.

Tabla 10.7: Prueba P-7

Código	**PA-7**
Prueba	Mover modulation wheel en todo su rango.
Resultado	El modulador se actualiza debidamente.
Evaluación	Positiva.

Tabla 10.8: Prueba P-8

Código	**PA-8**
Prueba	Mover pitch wheel en todo su rango.
Resultado	La frecuencia del oscilador de las voces se actualiza debidamente.
Evaluación	Positiva.

Tabla 10.9: Prueba P-9

Código	**PA-9**
Prueba	Se conecta un nuevo controlador MIDI. Se desconecta un controlador MIDI conectado antes de la ejecución del sistema.
Resultado	El sistema mantiene la lista de canales MIDI disponibles que generó al inicio de su ejecución.
Evaluación	Negativa
Motivo del problema	El sistema no tiene forma de detectar el evento de la conexión o desconexión de un controlador MIDI.
Corrección	Actualizar lista de canales MIDI cuando cambian las entradas MIDI disponibles.

Para solucionar este problema, se ha modificado la clase *MidiChannelAttachment* para que herede de la clase del framework JUCE *ChangeListener*. Ahora *MidiChannelAttachment* escucha al *AudioDeviceManager* del modelo y cuando un dispositivo MIDI es conectado o desconectado *MidiChannelAttachment* es notificado. Cuando esto ocurre, actualiza en el modelo y la vista la lista de canales MIDI.

Tabla 10.10: Prueba P-10

Código	**PA-10**
Prueba	Leer un preset con datos corruptos.
Resultado	El sistema tras comprobar los datos del preset, no altera su estado y notifica al usuario de que el contenido del preset es inválido o está corrupto.
Evaluación	Positiva

Tabla 10.11: Prueba P-11

Código	**PA-11**
Prueba	Leer un preset con una extensión de archivo incorrecta
Resultado	El sistema no altera su estado y notifica al usuario de que el preset que se ha intentado leer no es válido.
Evaluación	Positiva.

El caso descrito en la tabla 10.11 es muy poco común ya que a la hora de seleccionar un archivo para cargar como preset, el sistema sólo muestra archivos con la extensión correcta, pero se puede forzar introduciendo una ruta absoluta de un archivo.

Tabla 10.12: Prueba P-12

Código	**PA-12**
Prueba	Abortar lectura de preset antes de seleccionar un archivo
Resultado	El sistema no altera su estado y continúa su funcionamiento normal.
Evaluación	Positiva.

Tabla 10.13: Prueba P-13

Código	**PA-13**
Prueba	Leer un preset con una wavetable inexistente
Resultado	El sistema carga la configuración del preset con la wavetable por defecto.
Evaluación	Positiva.

Tabla 10.14: Prueba P-14

Código	**PA-14**
Prueba	Cambiar ajustes de audio a traves de la ventana de ajustes.
Resultado	El sistema cambia su estado para incorporar los nuevos ajustes. El sistema de audio ha cambiado sus parámetros.
Evaluación	Positiva.

Capítulo 11

Despliegue

Para desplegar la aplicación en sistemas Windows se ha empleado la herramienta Inno Setup [12], creando un script sencillo que genera un instalador que ubica la aplicación compilada en un directorio de instalación, facilitando la adición de la aplicación al menú de inicio y agregando esta a la lista de aplicaciones de Windows para poder ser gestionada como una aplicación estándar, ayudando, por ejemplo, su desinstalación.

```
; -- Softwaver.iss --
[Setup]
AppName=Softwaver
AppVersion=1.0
PrivilegesRequired=admin
DefaultDirName={commonpf}\Softwaver
DefaultGroupName=Softwaver
OutputBaseFilename=Softwaver-windows
Compression=lzma
SolidCompression=yes
ArchitecturesAllowed=x64compatible
ArchitecturesInstallIn64BitMode=x64compatible

[Dirs]
Name: "{userappdata}\Softwaver\Wavetables"; Flags: uninsalwaysuninstall
Name: "{userappdata}\Softwaver\Presets"; Flags: uninsalwaysuninstall

[Files]
Source: "Softwaver.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "Wavetables\*"; DestDir: "{userappdata}\Softwaver\Wavetables"; Flags: ignoreversion recursesubdirs
Source: "Presets\*"; DestDir: "{userappdata}\Softwaver\Presets"; Flags: ignoreversion recursesubdirs

[Icons]
Name: "{group}\Softwaver"; Filename: "{app}\Softwaver.exe"
Name: "{commondesktop}\Softwaver"; Filename: "{app}\Softwaver.exe"

[Run]
Filename: "{app}\Softwaver.exe"; Description: "Run application now"; Flags: nowait postinstall
```

Figura 11.1: Script para generar el instalador de la aplicación con Inno Setup

Además, el instalador ubica los archivos wavetable así como un conjunto de presets predeterminados en el directorio %appdata%. En caso de desinstalación de la aplicación, estos ficheros y directorios creados son eliminados automáticamente.

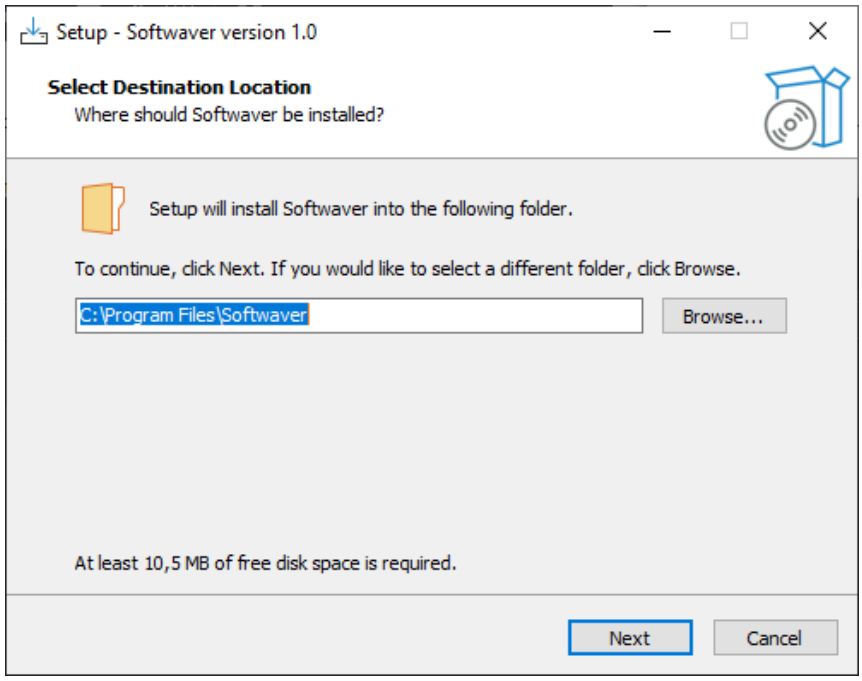


Figura 11.2: Instalador wizard de la aplicación

El resultado final es un instalador estilo *wizard* que guía de manera intuitiva al usuario durante la instalación de la aplicación.

Capítulo 12

Seguimiento del proyecto

La tabla 12.1 describe el desarrollo temporal del proyecto. Como se puede observar el proyecto ha finalizado notablemente antes de lo planificado. Una vez iniciado el proyecto, se recibió una oferta de un trabajo en junio, así que se decidió aumentar la carga de trabajo, trabajando durante fines de semana y realizando horas extra para adelantar la finalización de este proyecto.
Finalmente, aunque la fecha de finalización sea más temprana que la planificada, la cantidad de trabajo ha sido muy próxima a la estimada, variando levemente las duraciones de algunas fases con respecto a las estimadas.

Tabla 12.1: Seguimiento del proyecto

Fase	Inicio planificado	Inicio real	Final planificado	Final real
Estudio de factibilidad técnica	03/01/25	03/03/25	04/03/25	04/03/25
Estudio de factibilidad económica y temporal	05/03/25	05/03/25	07/03/25	07/03/25
Identificación de requisitos	10/03/25	10/03/25	11/03/25	10/03/25
Especificación de requisitos	12/03/25	11/03/25	14/03/25	13/03/25
Definición de casos de uso	17/03/25	14/03/25	18/03/25	19/03/25
Diagramas de casos de uso	19/03/25	20/03/25	26/03/25	26/03/25
Modelado del dominio	28/03/25	27/03/25	01/04/25	31/03/25
Diseño de la arquitectura general del sistema	02/04/25	01/04/25	07/04/25	04/04/25

Continúa en la siguiente página

Tabla 12.1 – Continúa de la página anterior

Fase	Inicio planificado	Inicio real	Final planificado	Final real
Diseño del núcleo de procesamiento de audio	08/04/25	07/04/25	11/04/25	10/04/25
Diseño de la interfaz gráfica de usuario	14/04/25	11/04/25	16/04/25	11/04/25
Mockups de la interfaz	17/04/25	14/04/25	18/04/25	14/04/25
Elección de tecnologías	21/04/25	15/04/25	22/04/25	15/04/25
Formación	23/04/25	16/04/25	28/04/25	18/04/25
Configuración del entorno	29/04/25	21/04/25	29/04/25	21/04/25
Implementación de la interfaz de usuario y modelo	30/04/25	21/04/25	13/05/25	30/04/25
Implementación del motor de audio	30/04/25	25/04/25	20/05/25	09/05/25
Implementación de persistencia	05/05/25	23/04/25	09/05/25	26/04/25
Integración	21/05/25	09/05/25	23/05/25	10/05/25
Pruebas	26/05/25	11/05/25	30/05/25	15/05/25
Despliegue	02/05/25	16/06/25	04/06/25	18/05/25
Entrega	05/06/25	19/05/25	11/06/25	26/05/25

Capítulo 13

Conclusiones

A lo largo del desarrollo de este Trabajo Fin de Grado, se ha llevado a cabo la implementación de un sistema de síntesis de audio con soporte para controladores MIDI, interfaz gráfica personalizada y almacenamiento de configuraciones mediante presets. Este proceso ha supuesto una oportunidad para aplicar de forma práctica los conocimientos adquiridos durante la formación universitaria, así como para adquirir nuevas habilidades técnicas y metodológicas.

Uno de los aprendizajes más significativos ha sido la capacidad para afrontar un proyecto completo desde su concepción hasta su despliegue. Esto ha incluido la identificación de requisitos, el análisis del sistema, el diseño arquitectónico, la implementación, la validación mediante pruebas y la documentación final. Cada fase ha requerido una planificación meticulosa y una adaptación constante a los imprevistos que surgieron en el camino.

Además, se han afianzado competencias clave como el uso de frameworks especializados (como JUCE para audio) y la estructuración modular del código para facilitar el mantenimiento y la escalabilidad del sistema. También se ha trabajado con herramientas de diseño y modelado UML para plasmar de forma visual y comprensible la estructura del sistema.

En cuanto a los retos superados, destaca la complejidad del procesamiento de audio en tiempo real, especialmente en lo que respecta a la optimización del rendimiento y la latencia. Este aspecto fue inicialmente una fuente de incertidumbre, pero se abordó mediante la mejora del manejo de buffers, la elección adecuada de bibliotecas eficientes y el ajuste fino de parámetros de configuración. Aunque finalmente el rendimiento del procesamiento de audio no ha terminado siendo tan competente como productos actuales del mercado, se ha desarrollado algo funcional que sienta las bases para optimizaciones posteriores.
Otro desafío importante fue la integración de la interfaz gráfica con el motor de audio, que requirió una cuidadosa sincronización de eventos y una arquitectura bien definida.

También resultó especialmente exigente la carga de trabajo, dado que el desarrollo fue llevado a cabo por una única persona, lo cual obligó a una gestión rigurosa del tiempo y a priorizar aquellas funcionalidades esenciales que aseguraran la entrega de un sistema robusto y funcional.

Finalmente, todos los objetivos descritos en el capítulo de Introducción han sido totalmente conseguidos.

En conclusión, este proyecto no solo ha dado como resultado una aplicación operativa, sino que también ha representado un proceso de crecimiento personal y profesional. La superación de los retos planteados y el aprendizaje derivado de ellos aportan una experiencia valiosa que sienta una base sólida para afrontar futuros desarrollos en el ámbito del software y la tecnología musical.

13.1. Líneas de trabajo futuras

Aunque la aplicación actual cumple con los objetivos propuestos, existe un amplio margen de mejora y expansión. A continuación, se detallan algunas posibles líneas de trabajo que permitirían incrementar la funcionalidad, el rendimiento y la versatilidad del sistema:

- **Adaptación a formato VST:** Una de las extensiones más relevantes sería portar la aplicación al formato de plugin VST, permitiendo su integración directa en estaciones de trabajo de audio digital (DAWs) como Ableton Live, FL Studio o Logic Pro. Esto aumentaría considerablemente su utilidad en entornos de producción musical profesional.
- **Animaciones en la wavetable:** La incorporación de visualizaciones dinámicas para las wavetables mejoraría significativamente la experiencia del usuario, proporcionando una representación visual clara y en tiempo real de la forma de onda seleccionada o generada.
- **Wavetables adaptativas por frecuencia:** Se propone implementar wavetables que utilicen múltiples *lookup tables*, con una cantidad variable de muestras por ciclo en función de la frecuencia fundamental. De esta forma, las formas de onda utilizadas en frecuencias graves contarían con mayor resolución, reduciendo el aliasing y enriqueciendo el contenido armónico en las zonas más bajas del espectro.
- **Oversampling con filtrado y downsampling:** Para mejorar la calidad del audio sintetizado y minimizar el aliasing, se sugiere implementar un sistema de *oversampling* seguido de un filtro paso bajo (LPF) y un posterior *downsampling*. Este enfoque permitiría una reproducción más fiel de las señales digitales, especialmente en entornos de síntesis compleja o cuando se emplean modulaciones rápidas.
- **Sistema de efectos:** Actualmente, el `AudioEngine` carece de módulos de procesamiento de efectos, lo cual limita el potencial creativo del sintetizador. Una línea de

mejora consistiría en la incorporación de efectos como *delay*, *reverb*, *chorus* o *distortion*, estructurados de forma modular y extensible.

Estas líneas de trabajo permitirían transformar la aplicación en una herramienta de síntesis aún más potente, flexible y profesional, adaptada a las necesidades de músicos, diseñadores sonoros y productores del entorno actual.

Bibliografía

[1] Aalto University. Acoustics and Audio Technology - Computer, Communication and Information Sciences, Master of Science (Technology), 2024. Último acceso: 21 de mayo de 2025.

[2] Ableton. Software Instruments: Wavetable, 2025. Último acceso: 11 de mayo de 2025.

[3] Arturia. Pigments: Polychrome Software Synthesizer, 2024. Último acceso: 21 de mayo de 2025.

[4] Change Vision, Inc. Astah: UML and Modeling Tool, 2024. Último acceso: 19 de marzo de 2025.

[5] John Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, 21:526–534, 1973.

[6] diagrams.net. diagrams.net (formerly draw.io): Diagramming and Flowchart Software, 2024. Último acceso: 21 de mayo de 2025.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] GitLab Inc. GitLab: The DevSecOps Platform, 2024. Último acceso: 19 de marzo de 2025.

[9] Glassdoor, Inc. Glassdoor: Información salarial y ofertas de empleo, 2024. Último acceso: 21 de mayo de 2025.

[10] IEEE Computer Society. *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)*. IEEE, 1998.

[11] ISO C++ Committee. C++ Standard Documentation, 2024. Último acceso: 19 de marzo de 2025.

[12] Jordan Russell, Martijn Laan. Inno Setup: Free Installer for Windows Programs, 2024. Último acceso: 14 de mayo de 2025.

[13] JUCE Developers. JUCE: The C++ Framework for Audio Applications, 2024. Último acceso: 19 de marzo de 2025.

[14] JUCE Developers. JUCE: Component Class Reference, 2025. Último acceso: 11 de mayo de 2025.

[15] JUCE Developers. JUCE: LadderFilter Class Reference, 2025. Último acceso: 11 de mayo de 2025.

[16] JUCE Developers. JUCE: LookAndFeel Class Reference, 2025. Último acceso: 11 de mayo de 2025.

[17] JUCE Developers. JUCE: Oscillator Class Reference, 2025. Último acceso: 11 de mayo de 2025.

[18] JUCE Developers. JUCE: Tutorials, 2025. Último acceso: 11 de mayo de 2025.

[19] JUCE Developers. Tutorial: Projucer Part 1: Getting started with the Projucer, 2025. Último acceso: 11 de mayo de 2025.

[20] Kilohearts AB. Phase Plant: Modular Synthesizer Plugin, 2024. Último acceso: 21 de mayo de 2025.

[21] Microsoft Corporation. Microsoft Excel: Spreadsheet Software, 2024. Último acceso: 19 de marzo de 2025.

[22] Microsoft Corporation. Microsoft Project: Project Management Software, 2024. Último acceso: 19 de marzo de 2025.

[23] Microsoft Corporation. Microsoft Visual Studio, 2024. Último acceso: 19 de marzo de 2025.

[24] Robert Moog. *Voltage-Controlled Electronic Music Modules*. Moog Music, 1964.

[25] Overleaf. Overleaf: LaTeX Collaborative Writing and Publishing Tool, 2024. Último acceso: 14 de mayo de 2025.

[26] Roger S. Pressman and Bruce R. Maxim. *Ingeniería del Software: Un enfoque práctico*. McGraw-Hill Education, 8 edition, 2020.

[27] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, 7th edition, 2021.

[28] Miller Puckette. *The Theory and Technique of Electronic Music*. World Scientific Publishing, 2007.

[29] Curtis Roads. *The Computer Music Tutorial*. MIT Press, 1996.

[30] Rust Programming Language. Rust Official Documentation, 2024. Último acceso: 19 de marzo de 2025.

[31] Ian Sommerville. *Software Engineering*. Pearson, 9th edition, 2011.

[32] Karl Wiegers and Joy Beatty. *Software Requirements*. Microsoft Press, 3rd edition, 2013.

[33] Xfer Records. Serum 2: Advanced Hybrid Synthesizer, 2024. Último acceso: 21 de mayo de 2025.

Apéndice A

Manuales

A.1. Manual de mantenimiento

Este manual está destinado a desarrolladores que deseen mantener, extender o adaptar la aplicación **Softwaver**. A continuación se describe la estructura general del sistema, los pasos necesarios para configurar el entorno de desarrollo, las dependencias clave y las pautas recomendadas para realizar modificaciones o ampliaciones del software.

A.1.1. Estructura del Proyecto

La aplicación sigue una arquitectura modular basada en el patrón *Modelo-Vista-Controlador (MVC)* con capa de servicios. El código se encuentra organizado en los siguientes paquetes principales:

- `audio`: Contiene los módulos de síntesis (oscilador, filtro, ruido, LFO, envelopes, etc.) y procesamiento de audio en tiempo real.
- `model`: Define los modelos de datos, parámetros y configuraciones del sistema.
- `controller`: Coordina el modelo y la interfaz gráfica de usuario (GUI) manteniendo una coherencia entre sus estados.
- `view`: Contiene la interfaz gráfica de usuario (GUI) con sus respectivos componentes.
- `file`: Gestiona el almacenamiento persistene y lectura de presets y wavetables.
- `common`: Incluye enumeraciones globales y funciones auxiliares reutilizables.

A.2.2 Entorno de Desarrollo

- **Lenguaje de programación:** C++
- **Framework principal:** JUCE
- **Entorno recomendado:** Visual Studio 2022 (Windows) con CMake y Projucer para generación del proyecto.
- **Control de versiones:** Git (repositorio alojado en GitLab)

A.1.2. Configuración del Entorno

1. Clonar el repositorio del proyecto desde GitLab.
2. Abrir el archivo `.jucer` con Projucer y generar los archivos de proyecto para Visual Studio.
3. Asegurarse de tener JUCE correctamente instalado y configurado en el sistema.
4. Abrir el proyecto en Visual Studio y compilar.

A.1.3. Recomendaciones para el Mantenimiento

- Las clases del paquete `audio` deben mantenerse ligeras y eficientes, dado que trabajan en tiempo real.
- Cualquier nuevo módulo de audio debe implementar una interfaz común para facilitar su integración y control desde la interfaz gráfica.
- Para extender la GUI, seguir el estilo y estructura definidos en las clases del paquete `view`, reutilizando los componentes personalizados existentes.
- Para añadir nuevos efectos, se recomienda crear un nuevo módulo orquestado por *AudioEngine* que procese la señal generada por *CustomSynth*.

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://gitlab.inf.uva.es/serramo/softwaver`.
- Descargar el instalador: drive.google.com