



TRABAJO FIN DE GRADO  
INGENIERÍA INFORMÁTICA

# SERVICIO SOFTWARE PARA INTERPRETACIÓN DE PARTITURAS PARA INSTRUMENTOS DE PERCUSIÓN

---

**Autor**

Sergio Samaniego Martínez

**Directores**

Manuel I. Capel Tuñón



Escuela Técnica Superior de Ingenierías Informática y  
de Telecomunicación

Granada, Junio de 2018









# SERVICIO SOFTWARE PARA INTERPRETACIÓN DE PARTITURAS PARA INSTRUMENTOS DE PERCUSIÓN

---

## **Autor**

Sergio Samaniego Martínez

## **Directores**

Manuel I. Capel Tuñón



## **Servicio software para Interpretación de partituras para instrumentos de percusión**

Sergio Samaniego Martínez

**Palabras clave:** Partitura, Notación, Trémolo, MIDI, Instrumento, VST, Calderón, Silencio, Nota, Batería

### **Resumen**

Este proyecto tiene como principal objetivo la creación de una herramienta software que permita a sus usuarios transformar una partitura, escrita en la notación del musicólogo Julio Omella para instrumentos de percusión, en una interpretación musical con calidad tímbrica y sonido natural de diferentes instrumentos del tipo mencionado: batería, timbal, tambor, etc. Para conseguirlo se ha definido una estructura de datos que almacena eficientemente todo el contenido de una partitura, posteriormente la aplicación desarrollada realiza una traducción automática al formato MIDI (Musical Instruments Digital Interface). En la etapa de postproceso se ha complementado la herramienta con la utilización de una biblioteca de sintetizadores de instrumentos de percusión para obtener calidad sonora real para cada uno de los diferentes tipos de instrumentos. El software una vez desarrollado se convertirá en un servicio Web de calidad y accesible a través de Internet.





## **Software service for performing musical partitures of percussion instruments**

Sergio Samaniego Martínez

**Keywords:** Music Sheet, Notation, Tremolo, MIDI, Instrument, VST, Silence, Note, Drums.

### **Abstract**

The fundamental objective of the project is to create a software tool that enables its users to transform a musical partiture for percussion instruments, written in the musicologist Julio Omella's notation, into a musical performance with a higher timbrical quality that captures the natural sound of different instruments: drum kit, cymbals, bass-drum, etc. A data structure was defined to efficiently store the content of a given partiture, and then an automatic translation to MIDI (Musical Instruments Digital Interface) is performed. In the post-processing stage, the developed tool was supplemented with a professional library of percussion synthesizers for obtaining sound quality for each instrument type. The software once developed becomes a quality Web service accessible through the Internet.



---

Yo, **Sergio Samaniego Martínez**, alumno de la titulación INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76628047-M, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Sergio Samaniego Martínez

Granada a 18 de Junio de 2018 .



---

D. **Manuel I. Capel Tuñón**, Profesor del Departamento  
Lenguajes y Sistemas Informáticos de la Universidad de  
Granada.

**Informan:**

**Que el presente trabajo, titulado Servicio software para Interpretación de  
partituras para instrumentos de percusión**, ha sido realizado bajo su supervisión  
por **Sergio Samaniego Martínez**, y autorizamos la defensa de dicho trabajo ante el  
tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a  
18 de Junio de 2018 .

**Los directores:**

**Manuel I. Capel Tuñón**



# Agradecimientos

Agradecimientos a Julio Omella por la notación musical que nos ha permitido realizar este trabajo. Con esta nueva notación, se podrán crear partituras para instrumentos de percusión sin tener unos extensos conocimientos musicales.





## Tabla de contenido

1. Introducción.....	19
1.1 Contexto .....	19
1.2 Objetivos.....	19
1.3 Estado del arte .....	20
1.3.1 GarageBand.....	20
1.3.2 GuitarPro.....	20
1.3.3 EZDrummer .....	21
2. Planificación.....	22
2.1 Formato MIDI .....	22
2.2 Estudio de la librería Kontakt5 y Cubase .....	22
2.3 Traducción de la Notación .....	23
2.4 Paso de la notación a MIDI .....	23
2.5 Creación de la Interfaz Gráfica .....	23
2.6 Aplicación del instrumento VST a MIDI .....	23
3. Estudio de los datos básicos necesarios para nuestro proyecto .....	24
3.1 Notación musical para instrumentos de percusión propuesta por Julio Omella.....	24
3.2 MIDI.....	34
3.2.1 Nodo de cabecera.....	34
3.2.2 Nodo de pista .....	34
3.2.3 Eventos MIDI .....	35
3.2.4 Meta-Eventos MIDI.....	35
3.2.5 Canal y Notas.....	36
4. Análisis.....	38
4.1 Requisitos Funcionales .....	38
4.2 Requisitos No Funcionales.....	38
4.3 Estructura de Datos .....	39
5. Diseño.....	40
5.1 Diagrama de clases MIDI y Notación.....	41
5.2 Diagrama de clases Interfaz Gráfica .....	43
6. Implementación .....	45
6.1 MIDI .....	45
6.1.1 Clase MIDIVec.....	45
6.1.2 Clase MIDITrack.....	45
6.1.3 Clase MIDIFile.....	46
6.2 Partitura.....	46
6.2.1 Clase Baqueta .....	46

6.2.2 Clase Nota .....	47
6.2.3 Enum TIPO_TREMOLO .....	47
6.2.4 Clase TREMOLO .....	47
6.2.5 Clase Partitura .....	50
6.3 Clase QtApplication1 .....	52
6.4 Clase Almacenar .....	54
6.5 Clase CalderonDialog .....	54
6.6 Clase NotaDialog .....	55
6.7 Clase SilencioDialog .....	55
6.8 Clase TremoloDialog .....	55
7. Instalación y Configuración .....	57
7.1 Cubase 8 y Kontakt 5 .....	57
8. Pruebas .....	61
9. Trabajo Futuro .....	69
10. Conclusiones .....	70
Índice de tablas .....	73
Índice de imágenes .....	73
Bibliografía .....	75

# 1. Introducción

## 1.1 Contexto

El trabajo trata de llevar a la práctica una nueva notación musical propuesta por el especialista en interpretación musical Julio Omella para instrumentos del tipo de percusión. Esta base de notación musical es exclusivamente de Julio Omella y él nos ha dado el permiso para poder utilizarla en este trabajo. De esta forma lo que intentamos conseguir es que una partitura escrita en esta notación musical pueda ser traducida al formato MIDI (Musical Instrument Digital Interface) y, posteriormente, utilizando una biblioteca de sintetizadores sea posible conseguir la interpretación de esta partitura con un ordenador consiguiendo un sonido de calidad tímbrica correspondiente, lo más parecida posible al sonido natural de diferentes instrumentos de percusión.

Este sistema u otro análogo no ha sido desarrollado hasta ahora porque la música, hasta hace muy poco tiempo, se concentró sobre todo en alturas (melodías y acordes).

En la actualidad, algo a desarrollar es el aspecto de duración e intensidad (ritmo), que no ha sido aún tratado en toda su potencialidad, por eso es tan importante que este sistema de notación pueda ser desarrollado por la informática, porque sería una herramienta novedosa que superaría las limitaciones de la notación actual permitiendo un manejo del ritmo de extensiones ilimitadas y de precisión absoluta.

## 1.2 Objetivos

El objetivo de este proyecto es el de crear un sistema en el que consigamos abstraer la notación propuesta por Julio Omella a una estructura de datos que sea eficiente, construir el archivo MIDI, siguiendo esta estructura creada, y por último añadir los sintetizadores VST que nos permitirán conseguir los sonidos más reales para el instrumento batería, pero puede ser extendido con facilidad a otros instrumentos de percusión tras instalar las bibliotecas de instrumentos adecuadas

Para poder llevar a cabo todas estas tareas, realizaremos los siguientes pasos:

- Analizar y comprender la notación de Julio Omella.
  - Ciertos símbolos, como pueden ser los “trémolos” no son un evento MIDI como tal, sino que son una abreviación de varios eventos MIDI. Por lo tanto, la traducción de estos símbolos será complicada y un único trémolo se corresponderá con varios eventos MIDI.
  - En la notación existen varias marcas que a la hora de interpretación no tienen efecto sonoro, simplemente son de interés informativo.
- Estudiar el estándar MIDI (Musical Instrument Digital Interface)
  - Tendremos que crear en C++ las clases que sean necesarias para poder trabajar con este estándar y crear de esta forma archivos que sean funcionalmente ejecutables con programas que permiten reproducir música en formato MIDI.
- Instalar un programa base que nos permita trabajar con diferentes librerías VST para poder reproducir los archivos MIDI creados.
  - Descargaremos librerías VST que nos permitan simular los sonidos del instrumento batería lo más fielmente posible a la realidad como, por ejemplo, utilizando la librería “Kontakt 5”.

Procederemos a conocer cómo funciona esta librería y cómo instalarla en un programa base que nos permita reproducir archivos MIDI como es el caso del programa “Cubase” de Steinberg para grabar, arreglar y editar música de instrumentos y también en formato MIDI.

### 1.3 Estado del arte

En esta sección veremos algunos proyectos existentes parecidos al nuestro. Presentaremos un par de programas que nos van a permitir crear y trabajar con partituras, reproducirlas y generar archivos de audio.

#### 1.3.1 GarageBand

GarageBand es una aplicación informática que va a permitir al usuario crear piezas de música. Esta aplicación está desarrollada por Apple y por lo tanto disponible en sus sistemas operativos iOS y MacOS.

La aplicación GarageBand no está destinada a músicos profesionales, pero ayuda a los principiantes a poder producir música de una forma sencilla. Esta aplicación presenta una amplia variedad de instrumentos con los que generar las diferentes muestras de audio.



*Ilustración 1: Captura de pantalla de GarageBand*

Esta es una aplicación muy potente que nos permitirá trabajar con archivos MIDI y crear partituras completas, con una gran variedad de instrumentación.

#### 1.3.2 GuitarPro

Guitar Pro es un editor de partituras orientado principalmente para guitarra y bajo, pero es capaz de admitir todos los instrumentos soportados por el formato MIDI. Es un software desarrollado por la empresa Arobass Music y disponible en Microsoft Windows, MacOS y GNU/Linux.

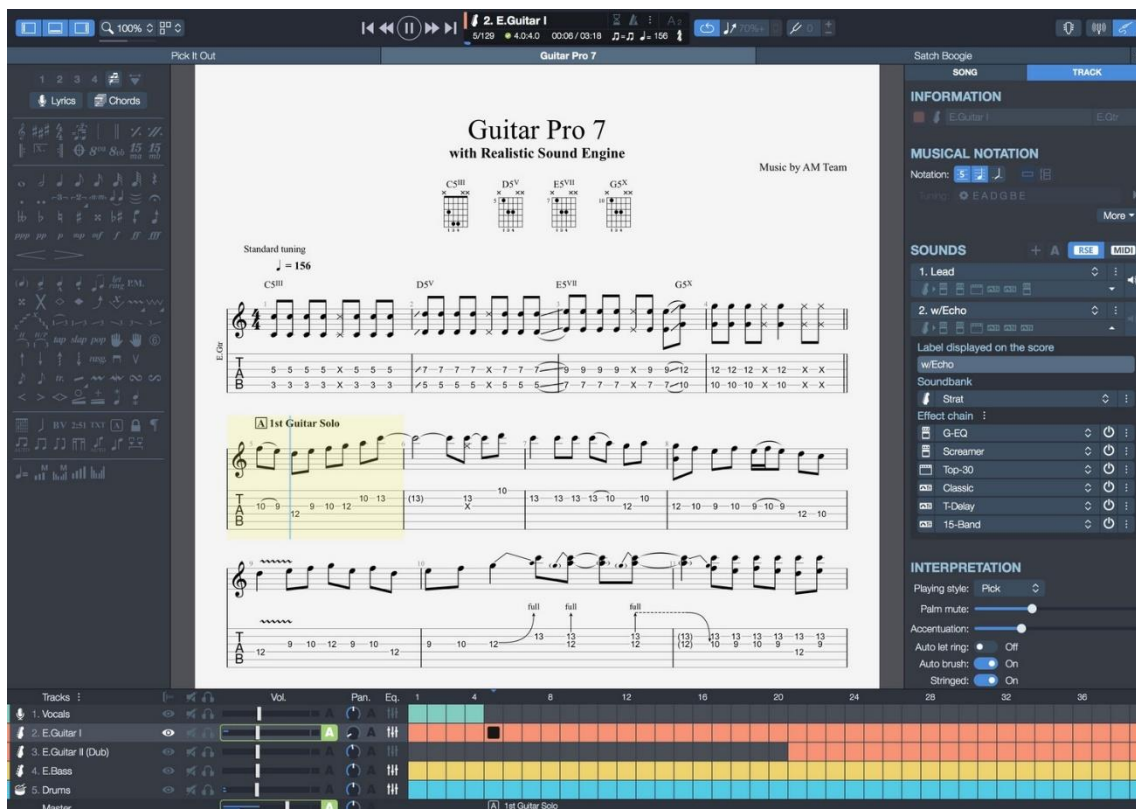


Ilustración 2: Captura de pantalla Guitar Pro

Guitar Pro nos permite crear partituras de batería además de guitarra y bajo. Esta aplicación podemos decir que se encuentra orientada hacia un público con conocimientos musicales, ya que necesitamos trabajar con pentagramas, como en una partitura profesional. Por lo tanto, para poder crear partituras de percusión en este programa, será necesario un mínimo de conocimiento musical.

### 1.3.3 EZDrummer

EZDrummer es un software para la creación de pistas de baterías. Este software ha sido desarrollado para poder eliminar cualquier limitación técnica que se pueda tener a la hora de creación de partituras para batería.

EZDrummer está basado en kits reales de batería y contiene además un mezclador que nos permite ajustar los volúmenes de cada parte de la batería por separado.

A diferencia de las dos herramientas anteriores, EZDrummer si está orientado para realizar partituras de batería, por lo que la creación de estas partituras es mucho más sencillo. GuitarPro no está orientado a este tipo de partituras y por lo tanto necesitamos conocimientos musicales para la generación de la partitura y GarageBand, trabaja directamente con MIDI, creamos las partituras MIDI desde un teclado MIDI. EZDrummer trabaja directamente con la batería como vemos en la siguiente imagen y la creación de las partituras es mucho más intuitiva, debido a que golpeamos directamente la batería como si se tratara de una real.



Ilustración 3: Captura de pantalla EZDrummer

## 2. Planificación

Este proyecto vamos a pasar a dividirlo en seis partes para poder afrontarlo poco a poco. Las partes vamos a describirlas en el orden de ejecución óptimo para poder realizar este trabajo de la mejor forma posible.

### 2.1 Formato MIDI

Vamos a comenzar por estudiar el formato MIDI y cómo trabaja desde dentro, conociendo cómo será posible generar los archivos MIDI. Como esta tarea no depende de las demás y además quizás sea la más desconocida, será la primera que abordaremos para tener conocimientos claros antes de comenzar con el resto del proyecto.

### 2.2 Estudio de la librería Kontakt5 y Cubase

Posteriormente, pasaríamos a conocer cómo funcionan las librerías VST que nos van a permitir simular el sonido de una batería real con nuestro archivo MIDI.

Existen multitud de librerías VST para cualquier tipo de sonido, por lo que tendremos que buscar información de Instrumentos VST para percusión que es lo que nosotros necesitaremos en este trabajo.

Por otra parte, estas librerías necesitan de un programa base, desde las que se puedan ejecutar. En mi caso, seleccioné trabajar con “Cubase” por haber trabajado ya con él y de esta forma me sería más sencillo conocer la integración de estas librerías sobre “Cubase”.

## 2.3 Traducción de la Notación

Una vez estudiado el funcionamiento de las librerías y del estándar MIDI, pasamos a comprender la notación de Julio Omella.

Gracias a conocimientos musicales previos, la comprensión de una notación como ésta, así como la traducción de la misma al formato MIDI no resulta algo complicado, ya que puedo conocer en todo momento qué se está representando en cada una de las partituras.

## 2.4 Paso de la notación a MIDI

Este paso será el que comprende la mayoría del trabajo, ya que una vez conocido cómo funciona el estándar MIDI y cómo representar la notación musical, debemos ponernos a trabajar en ello para conseguir ir generando sonidos en el formato MIDI.

Una vez que conseguimos generar los sonidos correspondientes (Notas, Trémolos, etc) veremos cómo trabajar para generar una partitura completa.

## 2.5 Creación de la Interfaz Gráfica

Finalmente, con la creación de la partitura desde línea de comandos, pasamos a crear la interfaz gráfica para que sea más sencillo e intuitivo realizar la creación de una partitura.

Para la creación de la interfaz gráfica utilizaremos Qt y por lo tanto, antes de comenzar con la creación de esta Interfaz, también será necesario estudiar cómo funciona Qt, así como integrarlo con Visual Studio.

## 2.6 Aplicación del instrumento VST a MIDI

El último paso será el de adaptar nuestro archivo MIDI, que hemos creado desde la Interfaz Gráfica desarrollada, para que el sonido sea interpretado con la calidad sonora de un instrumento VST.

Por consiguiente, con el archivo MIDI creado tendremos que aplicarle el instrumento VST y ver que todo se escucha tal y como queríamos, con sonidos mucho más naturales gracias a la aplicación de estas librerías VST.

A continuación veremos la organización de las tareas pendientes realizando una tabla con los tiempos dedicados a cada una de las tareas.

TAREA	FECHA DE INICIO	DURACIÓN (días)	FECHA DE FIN
Formato MIDI	01/12/2017	30	01/01/2018
Estudio de librería Kontakt 5 y Cubase	01/01/2018	15	18/01/2018
Traducción de la notación	15/01/2018	50	10/03/2018
Paso de la notación a MIDI	01/02/2018	70	10/05/2018
Creación de la Interfaz Gráfica	01/03/2018	50	20/05/2018
Aplicación del Instrumento VST a MIDI	15/04/2018	5	20/05/2018

Ilustración 4: Tabla organización de tareas

Con la tabla de las tareas, las fechas de inicio y de final de cada una, procedemos a elaborar el Diagrama de Gantt correspondiente, con el que podemos observar la planificación y los tiempos del proyecto de una forma gráfica.



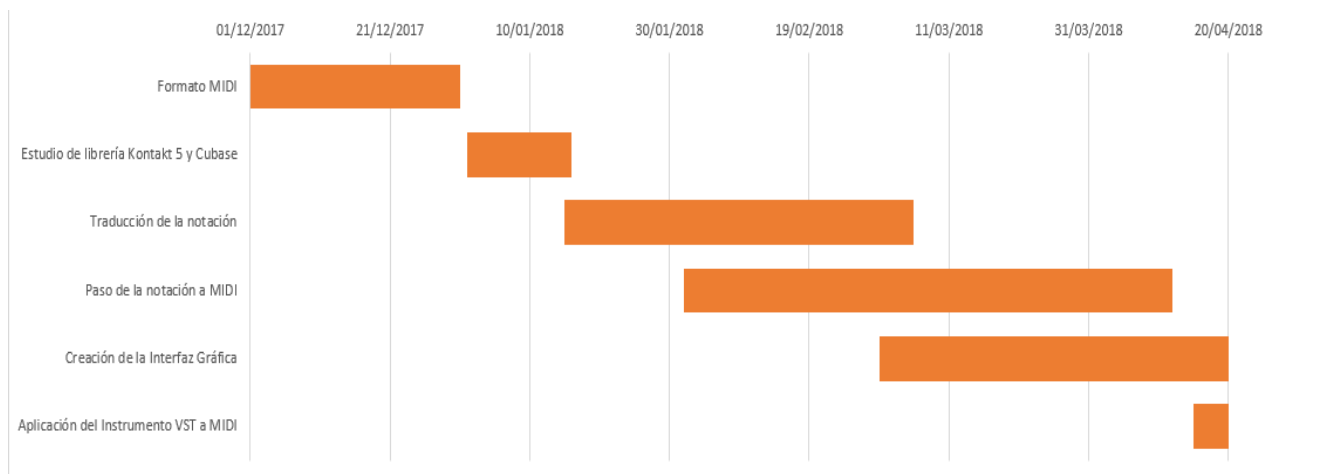


Ilustración 5: Diagrama de Gantt

## 3. Estudio de los datos básicos necesarios para nuestro proyecto

### 3.1 Notación musical para instrumentos de percusión propuesta por Julio Omella

#### Hexagrama

La notación se basa en un hexagrama de líneas negras dividido verticalmente por tres tipos de líneas de tres tonos diferentes de grises. Se lee de izquierda a derecha. En el eje de ordenadas se sitúa el factor intensidad (decibelios), y en el de abscisas el factor tiempo (segundos).

- Con las divisiones verticales y los tonos de grises se facilita tanto la lectura como la escritura.
- Las notas podrán ir situadas en cualquier lugar a lo largo del hexagrama, sin necesidad de coincidir con las líneas o espacios.
- Las líneas horizontales deben situarse por encima de las demás para facilitar la lectura horizontal.
- Medidas mínimas y tono aproximado de los colores:



Ilustración 6: Medidas mínimas



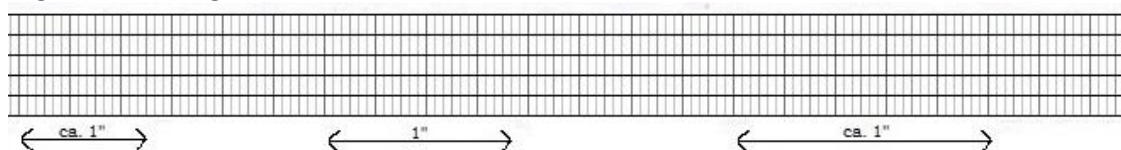
- Debe respetarse la proporción de las medidas.
- El programa contará con un zoom graduable.

### **Altura**

- Viene dada a consecuencia de la intensidad. A mayor intensidad, mayor será la altura del tono del sonido.
- Diferentes relaciones intensidad-altura por cada instrumento y en función del material de la baqueta
- Debe ser un aspecto muy cuidado para lograr la mayor naturalidad posible.

### **Duración**

- Tempo (velocidad de reproducción) indicado en unidad de segundo (1'') asociada a un segmento de longitud variable. 3

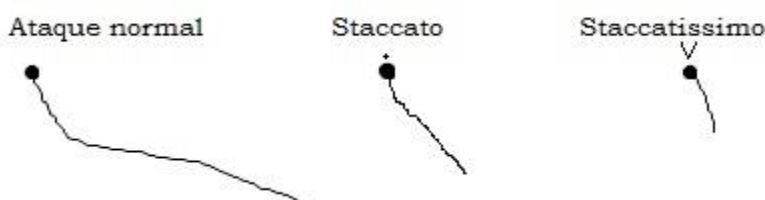


*Ilustración 7: Ejemplo de diferentes tempos*

- **ca.:** "circa", aproximadamente. Debe poderse seleccionar la opción de incluir ca. o no incluirla. El programa lo reproducirá exactamente a 1'' independientemente de si se indica ca. o no. Se trata de una indicación para el intérprete.
- La indicación de tempo podrá modificarse en cualquier momento a lo largo de la partitura.

### **Símbolos de duración:**

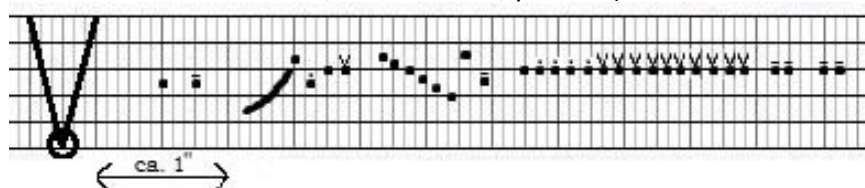
- **Staccato (.)**. Indicado con un pequeño punto encima de la nota. Reduce el tiempo de resonancia de la nota.
- **Staccatissimo (v)**. Indicado con una cuña encima de la nota. Reduce ampliamente el tiempo de resonancia de la nota.



*Ilustración 8: Ejemplo de staccato y staccatissimo*

- **Tenuto (—)**. Indicado con una raya encima de la nota. No afecta al sonido, es una indicación para el oyente o intérprete.

(Ejemplo de combinación de staccato, staccatissimo, y tenuto)



*Ilustración 9: Ejemplo de combinación de staccato, staccatissimo y tenuto*

• **Silencios:**

- **Silencio de negra:** Indica pausa breve

- **Cesura:** Indica pausa larga y podrá indicarse el tiempo de las pausas, con o sin **ca.**, o no indicarse. La indicación **ca.** no tendrá efecto en la reproducción del programa.

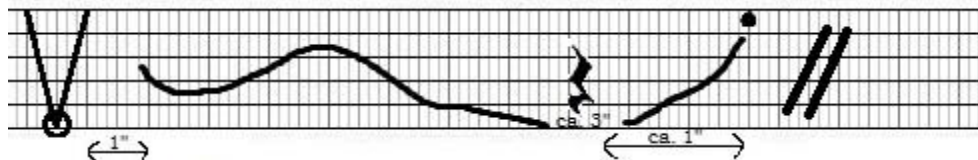


Ilustración 10: Ejemplo de silencios

(En este caso, el programa reproducirá 3 segundos exactos del silencio de negra, luego, el tiempo que se le haya indicado de la cesura, que no aparece escrito) 4

• **Calderón:** Indica que un fragmento debe ser repetido el tiempo que considere el intérprete. Se indicará el tiempo (en segundos) que es repetido el fragmento con **ca.** o sin ella, y al igual que en la cesura, podrá ser o no visible gráficamente en la partitura.

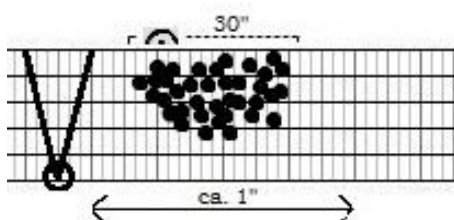


Ilustración 11: Ejemplo calderón

• **Sordina:** - Indica que debe aplacarse la resonancia del instrumento un 75%. Podrá ser graduable entre un 1% y un 95%.

- El primero indica que la sordina debe mantenerse a lo largo de la partitura hasta que se encuentre otra indicación que la anule, el segundo indica la utilización de sordina en un fragmento. Para eliminar la sordina, se indicará

- Se podrá realizar un paso gradual entre porcentajes o entre estos y la indicación “*senza sord.*”, tanto desde “” a un porcentaje concreto, como desde un porcentaje dado a una indicación “*senza sord.*”

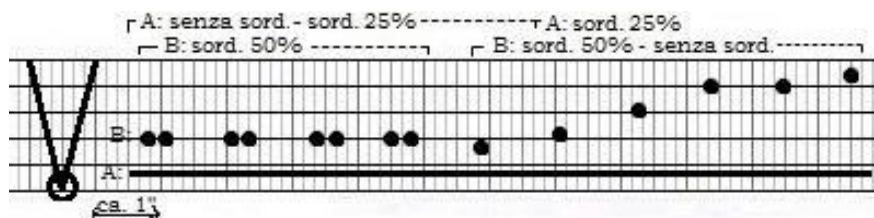


Ilustración 12: Ejemplo uso sordina

(En este caso, A pasará gradualmente de no tener sordina, a aplicarla restando un 25% de resonancia al instrumento. Luego de ello, mantendrá la sordina al 25%. B aplicará la sordina restando un 50% de resonancia al instrumento, y a partir de la 9ª nota, comenzará gradualmente a quitar la sordina hasta quitarla del todo en la 14ª nota)

• **Trémolo** (sucesión de ataques continuos ejecutados rápidamente). Toma forma de líneas, tal como se puede apreciar en la siguiente figura:

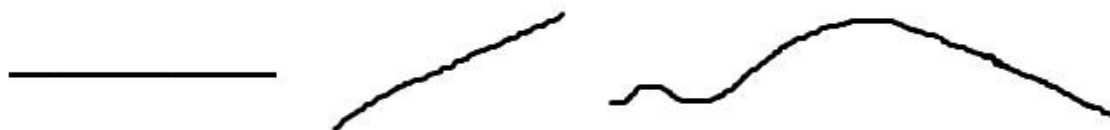


Ilustración 13: Ejemplo de trémolos

- La velocidad con la que se ejecutan los trémolos viene determinada por el símbolo. Cada línea horizontal representa un grado de velocidad, habiendo de uno a seis puntos, y por tanto, de una a seis líneas horizontales.

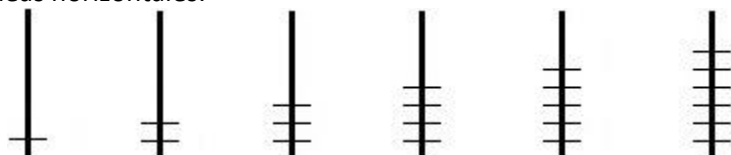


Ilustración 14: Símbolos de trémolo

Este símbolo podrá estar indicado: a) dentro de la partitura al comienzo de la línea de trémolo si no entorpece la visión de las demás voces, o b) fuera de ella en la parte superior indicando a la voz que pertenece. El tamaño del símbolo debe ser relativamente pequeño si se encuentra dentro de la partitura.

Al igual que la indicación para la sordina, el símbolo de velocidad de trémolo podrá estar indicado para un fragmento o para toda la partitura hasta que encuentre otra indicación que la anule.

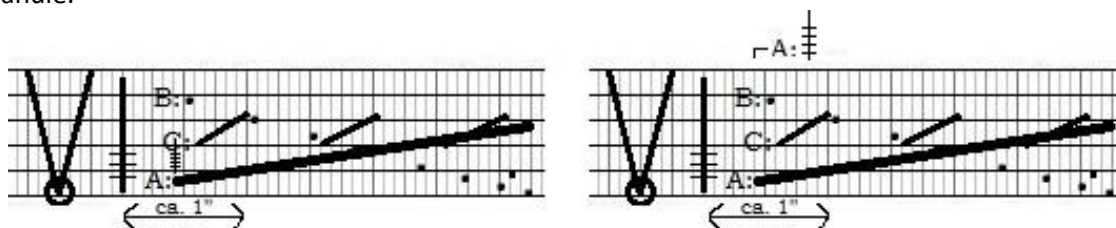


Ilustración 15: Ejemplo símbolo trémolo

(En el primer caso, el símbolo de trémolo que se indica con "A" está ubicado dentro de la partitura, y en el segundo caso, fuera de ella. Ambos tienen el mismo efecto, es decir, desde ese punto, "A" indica que se mantendrá la velocidad de trémolo a 4 puntos hasta que encuentre otra indicación que la anule)

Una misma indicación de velocidad de trémolo puede estar compartida por varias voces o por todas. En ese caso, se indicará en la parte superior de la partitura las voces a las que pertenece, o "tutti" si es para todos.

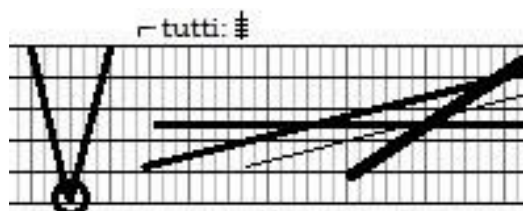


Ilustración 16: Ejemplo tutti

(En este caso, todas las voces ejecutarán el trémolo a 4 puntos sobre 6 hasta que se encuentre otra indicación que la anule)

Si no se especifica la velocidad de trémolo, esta será de 3 puntos sobre 6. 6

#### - Indicadores de ejecución gradual de trémolo.

En relación con el tiempo:



Ilustración 17: símbolos trémolo de tiempo

Este indicador se sitúa en la parte superior de la partitura haciendo referencia a una voz, a varias, o a todas.

En el fragmento comprendido entre ambos extremos del indicador, el trémolo debe ejecutarse a una velocidad de ataque gradual ascendente en el primer caso (**Ilustración 14** : Imagen Izquierda), y descendente en el segundo (**Ilustración 14**: Imagen derecha). El número de líneas horizontales determinará la velocidad de inicio y la de llegada.



Ilustración 18: Ejemplo símbolo de trémolo en el tiempo

(El trémolo se ejecutará a una velocidad ascendente de uno a cuatro puntos, y pasado este indicador, volverá a la velocidad anterior. En el caso de que no estuviese indicado anteriormente, o fuese el primer trémolo de esa voz, la velocidad del trémolo continuaría a tres puntos por ser la velocidad estándar)

#### En relación con la intensidad:



Ilustración 19: Símbolos trémolo de intensidad

Este indicador se sitúa en la parte superior de la partitura haciendo referencia a una voz, a varias, o a todas.

El primer indicador (izquierda) determina que a mayor intensidad, mayor velocidad de ejecución del trémolo. El segundo indicador (derecha) determina que a mayor intensidad, menor velocidad de ejecución de trémolo.

Podrá estar indicado para un fragmento o para toda la partitura hasta que se encuentre otra indicación que la anule.

El número de líneas horizontales determina la velocidad de inicio y la de llegada.

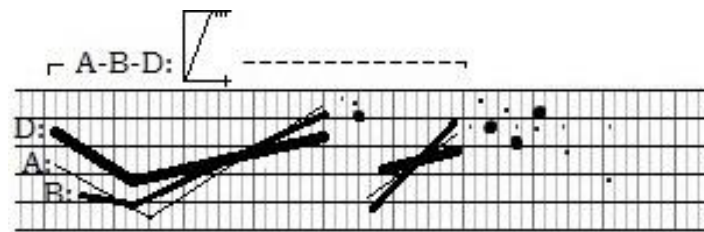


Ilustración 20: Ejemplo símbolo trémolo de intensidad

- Claves de trémolo.

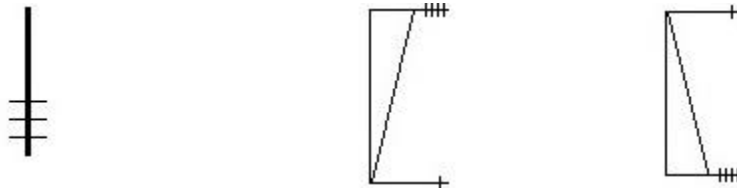


Ilustración 21: Claves trémolo

Ocupan todo el hexagrama y se indican en cualquier momento afectando a todas las voces y a toda la partitura hasta que se encuentre otra indicación que la anule.

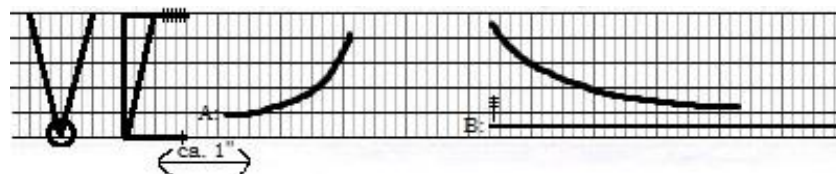


Ilustración 22: Ejemplo claves de trémolo

(En este caso, la clave no afecta a B, puesto que lleva su propia indicación de velocidad de trémolo)

- Habrá, por tanto, tres tipos de indicadores de velocidad de trémolo, uno fijo, y dos graduales; y dos tipos de claves:

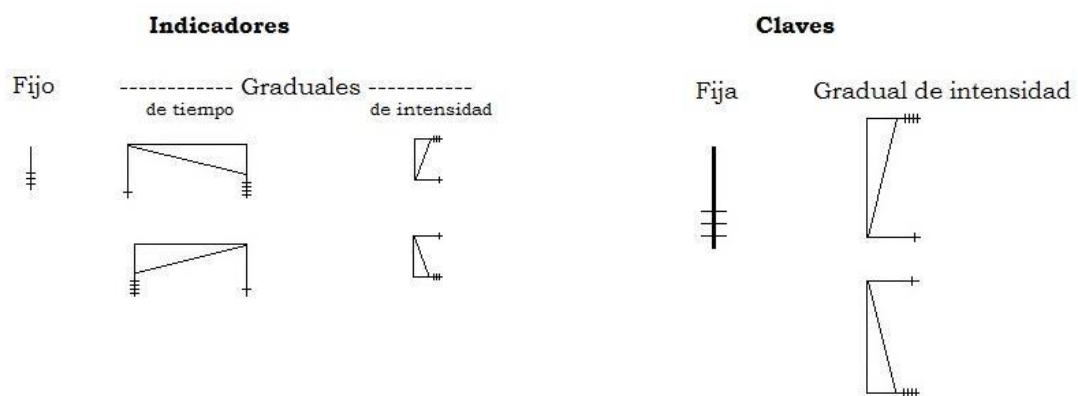
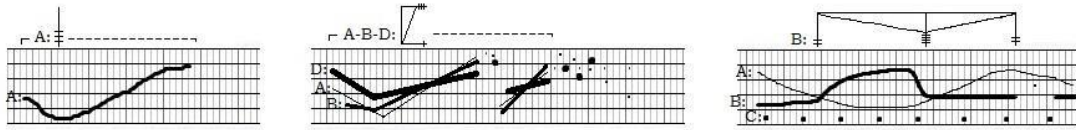


Ilustración 23: Resumen símbolos de trémolo

Se aplican en fragmentos, o en toda la partitura hasta que se indique otra cosa:

### Aplicados en fragmentos



### Aplicados de forma continua

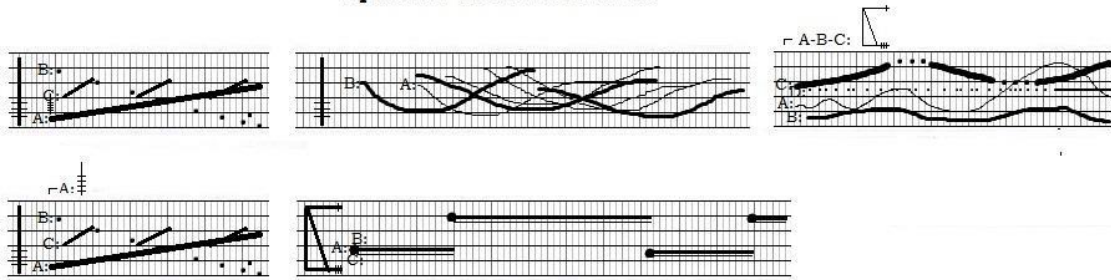


Ilustración 24: Ejemplos de trémolos

### Intensidad

- Se reparten 21 puntos de intensidad de la siguiente forma:

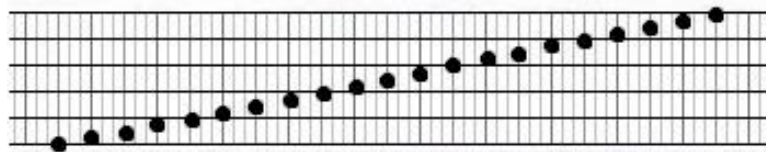


Ilustración 25: Puntos de intensidad

- Ocasionalmente podrá disponerse de 41 puntos de intensidad seleccionando una herramienta de zoom.

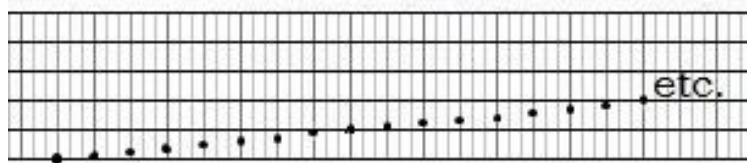


Ilustración 26: Zoom de puntos de intensidad

- Para trémolos, no habrá saltos bruscos de intensidad, sino que ésta debe variar de forma gradual.

- Claves de intensidad:

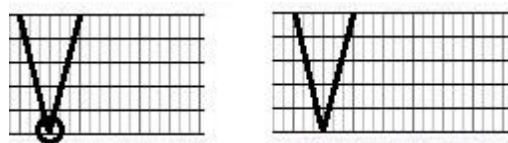


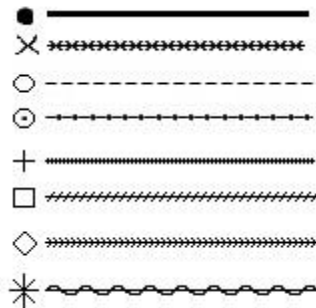
Ilustración 27: Claves de intensidad

- Se sitúan a elección al comienzo de la partitura.
- Ninguna de las dos tendrá efecto sonoro sobre el programa.

### **Timbre**

- Se tratará de lograr la mayor calidad sonora posible.
- Resonancia general graduable.
- Los instrumentos deberán tener su propia resonancia.
- Las voces se nombrarán por medio de letras asignadas (A, B, C, D... A2, B2, C2, D2... ad infinitum) al comienzo de su aparición. Las letras podrán aparecer representadas o no en la partitura.
- Las voces vendrán representadas mediante símbolos y colores diferentes que podrán escogerse y combinarse a voluntad. Para el caso del primer símbolo abajo expuesto, habrá diferentes grosores disponibles.

Símbolos aproximados:



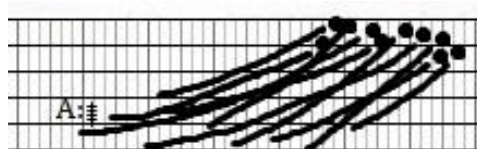
*Ilustración 28: Símbolos de voces*

Gama de colores aproximados:



*Ilustración 29: Colores para voces*

- Una sola voz podrá desdoblarse en varias voces que comparten una misma letra, instrumento, y velocidad de trémolo.



*Ilustración 30: Desdoblamiento de una voz*

### **· Nodos y vientres**

- Dependiendo del lugar en el que se golpee un tambor, este emitirá mayor o menor cantidad de armónicos. Si el tambor es tocado en el centro, los armónicos disminuyen porque es un punto nodal, mientras que, si el tambor es tocado más al borde, emitirá mayor cantidad de armónicos por ser un punto de vientre.
- Se dividen en seis puntos; tres para vientre y tres para nodo. Comenzando del borde al centro del tambor, seguiría el siguiente orden: V3, V2, V1, N1, N2, N3.



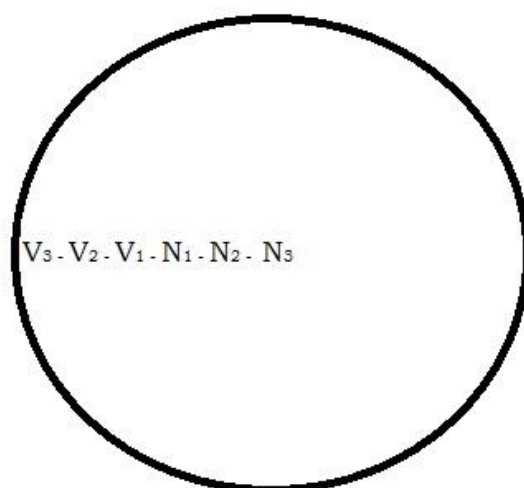


Ilustración 31: Vientres y nodos

- Si en la partitura no se especifica, todos los instrumentos sonarán en V1 como valor estándar.
- Para cambiar el lugar de golpeo del instrumento, se indicará de manera similar a los indicadores de trémolo. Asimismo, se dispondrá de dos claves diferentes de tímbrica nodo-vientre.

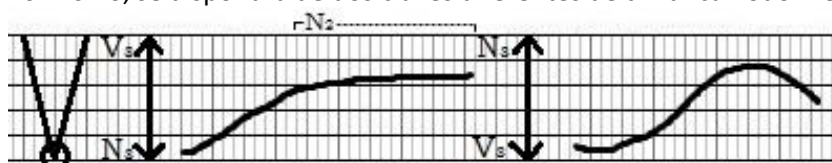


Ilustración 32: Ejemplo de uso de nodos y vientres

#### • Baquetas

- Otro recurso para cambiar la tímbrica es la utilización de diferentes materiales para las baquetas. Se dispondrán los siguientes materiales: madera, fieltro, corcho y goma; cada uno con las siguientes durezas: ligera, media, dura.
- La notación para baquetas es similar a los indicadores de trémolo o de nodos y vientres. La abreviación utilizada será "baq."
- No habrá claves para baquetas.
- Si no hay indicación, la baqueta utilizada será de madera. 11

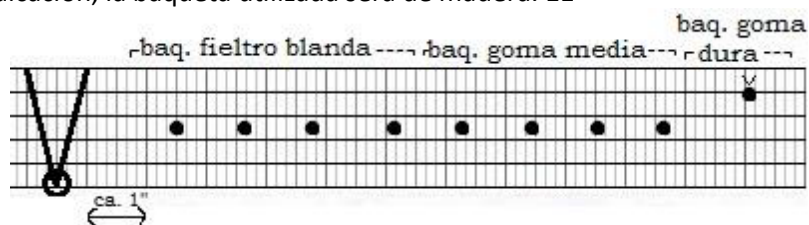


Ilustración 33: Ejemplo de uso de baquetas

#### • Instrumentos

Se dispondrá de los siguientes instrumentos:

Agogô

Agong

Alarma de diferentes timbres y alturas



Bongo agudo, medio, y grave  
 Cabasa  
 Caja muy aguda, aguda, media, y grave (snare drum)  
 Cañón  
 Caja china muy aguda, aguda, media, grave, y muy grave (woodblocks)  
 Campana muy aguda, aguda, media, grave, y muy grave  
 Cencerro agudo, medio y grave  
 Claxon  
 Clave Conga aguda, media, y grave  
 Cuenco tibetano agudo, medio, y grave  
 Darbuka aguda, medio, y grave  
 Djembe agudo, medio y grave  
 Gong agudo, medio y grave  
 Maraca Palmas Plato muy agudo, agudo, medio, grave, y muy grave (cymbal)  
 Silbato agudo, medio y grave Sirena de diferentes timbres y alturas  
 Timbal de orquesta muy agudo, agudo, medio, grave, y muy grave  
 Tom agudo, medio, grave, y muy grave  
 Triángulo  
 Viento

- Para instrumentos que puedan realizar sonidos continuos como la sirena o el viento, se dará la posibilidad de añadir una séptima línea horizontal de velocidad de trémolo para indicar que el sonido es continuo.

### Aspectos generales

· Al comienzo de la partitura, antes de establecer la clave de intensidad, podrá incluirse información de las diferentes voces en cuanto a letra que le corresponde, instrumento, grafía con la que se notarán notas sueltas y trémolos, velocidad de trémolo, sordina, tímbrica de nodo-vientre, y baquetas. La información asociada a las voces en el comienzo tendrá prioridad ante las claves generales del principio.

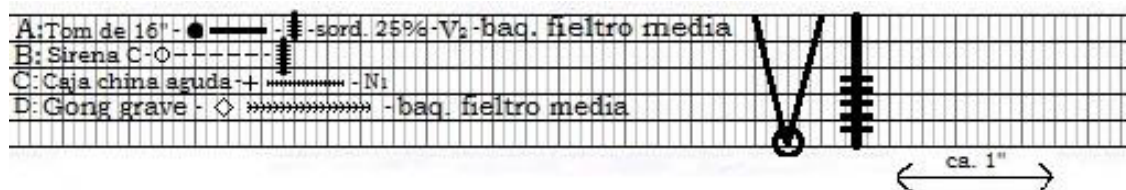


Ilustración 34: Información inicial de partitura

· Podrá seleccionarse la opción de contar con una guía visual que siga el transcurso de la reproducción de la partitura. Esta guía será una leve línea vertical que se sitúe desde la primera línea del hexagrama hasta la sexta.

· El espacio entre hexagramas debe ser suficientemente amplio para albergar en un momento determinado múltiples indicaciones de intensidad y tímbrica, aunque no siempre será necesario. Por tanto, el espacio entre hexagramas será ampliable y reducible en función de la información que albergue.

· En el caso de coincidir notas con claves o símbolos situados en el interior del hexagrama, se situarán por encima de estos obteniendo así las notas prioridad de visualización. Cuando dos o más voces coincidan en un punto concreto, las figuras asociadas a las voces quedarán

superpuestas. En caso de ilegibilidad, podrán seleccionarse dos hexagramas conjuntamente formando un sistema de hexagramas que funcionarán como uno solo, ampliando así el espacio disponible de escritura.

## 3.2 MIDI

El estándar MIDI (Musical Instrument Digital Interface) se trata de un estándar tecnológico que describe un protocolo, una interfaz digital y distintos conectores que permiten que varios instrumentos musicales electrónicos, computadores u otros dispositivos relacionados puedan conectarse y comunicarse entre sí.

Este formato se forma como una lista de nodos enlazada que contiene cada uno de los eventos clasificados por pistas.

Todos estos valores están en un formato big-endian, por lo que debemos tener esto en cuenta para trabajar con ellos.

### 3.2.1 Nodo de cabecera

Un archivo MIDI siempre va a tener como primer nodo el denominado nodo de cabecera, que contendrá la siguiente información:

Longitud (Bytes)	Descripción	Valor
4	Firma	"Mthd"
4	Tamaño de Cabecera	6
2	Formato	0-2
2	Número de pistas	1-65535
2	División del tiempo	

Tabla 1: Contenido Nodo Cabecera MIDI

Las variables nos darán la información siguiente:

- **Formato:** Esta variable nos va a indicar el formato del archivo MIDI que puede ser:
  - **0.** Una sola pista
  - **1.** Varias pistas, síncronas
  - **2.** Varias pistas, asíncronas
- **Número de pistas:** Esta variable nos indicará el número de pistas del archivo MIDI.
- **División de tiempo:** Esta variable indicará los ticks de reloj que durará una nota.

### 3.2.2 Nodo de pista

Una pista, a su vez constará de nuevo de una cabecera además de la lista de eventos que indicará la actividad de la pista, terminando con el meta-evento MIDI "fin de pista".

Longitud (bytes)	Descripción	Valor
4	Firma	"MTrk"

4	Tamaño	0-65535
---	--------	---------

Tabla 2: Contenido de la Cabecera Nodo Pista MIDI

### 3.2.3 Eventos MIDI

Cada evento MIDI tiene una distribución como la que vemos a continuación:

Longitud (bytes)	Descripción
Variable	Retraso
1	Tipo de evento y canal
1	Parámetro 1
1	Parámetro 2

Tabla 3: Contenido Nodo Evento MIDI

- **Retraso:** Esta variable nos va a indicar el número de ticks de reloj que separan el evento actual del evento anterior.
- **Tipo de evento y canal:** Estas variables se van a repartir en el byte de forma que los cuatro bits más significativos corresponderán al evento y los cuatro bits menos significativos al canal MIDI.

A continuación, veremos los tipos de eventos existentes:

Tipos de Evento			
Valor	Nombre	Parámetro1	Parámetro2
0x80	NOTE-OFF	Nota	Velocidad
0x90	NOTE-ON	Nota	Velocidad
0xA0	NOTE-AFTERTOUCH	Nota	Velocidad
0xB0	CONTROLLER	Controlador	Valor
0xC0	PROGRAM-CHANGE	Programa	
0xD0	CHANNEL-AFTERTOUCH	Velocidad	
0xE0	PITCHBEND	Valor	
0xF0	SYSTEM-EXCLUSIVE		
0xFF	Meta-evento		

Tabla 4: Tipos de Evento MIDI

### 3.2.4 Meta-Eventos MIDI

Este tipo de eventos solo sirven para extender la semántica de los eventos que se han visto anteriormente. Los meta-eventos tienen la siguiente distribución:

Longitud(bytes)	Descripción
Variable	Retraso
1	0xFF

1	Tipo de meta-evento
Variable	(Longitud) del argumento
(Longitud)	Valor del argumento

Tabla 5: Contenido del Nodo Meta-evento MIDI

Y los distintos tipos de meta eventos son los siguientes:

Tipos de meta-evento	
Valor	Nombre
0x00	Número de secuencia
0x01	Texto
0x02	Noticia de copyright
0x03	Nombre de la secuencia
0x04	Nombre del instrumento
0x05	Letra de la canción
0x06	Marca
0x07	Punto de corte
0x08	Nombre del programa
0x09	Nombre del dispositivo
0x20	Canal por defecto
0x21	Puerto por defecto
0x2F	Fin de pista
0x51	Tempo
0x54	Desplazamiento temporal
0x58	Indicación de compás
0x59	Indicación de tonalidad
0x7F	Evento específico para el secuenciador

Tabla 6: Tipos de metaevento

### 3.2.5 Canal y Notas

MIDI tiene reservado su canal número 10 para la percusión, donde el valor nota enviado a “NOTE-ON” será diferente del que se envía en todos los demás canales.

Para el resto de canales, el parámetro de Nota representa la frecuencia de la misma, teniendo un rango desde 0 a 127. De forma que, si por ejemplo nosotros indicamos como nota 60, estamos indicando que queremos un “Do” y cada unidad que avancemos aumentamos o disminuimos medio tono, siendo 61 “Do#”, o 59 “Si”.

En el caso del canal número 10, este parámetro nos indicará el instrumento que deseamos tocar, estando cada uno identificado por un único identificador numérico que veremos a continuación. En este canal cada nota estará asignada a un instrumento distinto que vemos a continuación:

Valor	Sonido	Valor	Sonido	Valor	Sonido
35	Bombo2	36	Bombo1	37	Aro de caja
38	Caja1	39	Palmas	40	Caja2
41	Base2	42	Hit-Hat Cerrado	43	Base1
44	Pedal Hithat	45	Tom Medio2	46	Hit-hat Abierto
47	Tom Medio1	48	Tom Agudo2	49	Crash

50	Tom agudo 1	51	Ride	52	China
53	Ride (Campana)	54	Pandereta	55	Splash
56	Cencerro	57	Crash2	58	VibraSlap
59	Ride2	60	Bongo Agudo	61	Bongo Grave
62	Conga Aguda (Mute)	63	Conga Aguda	64	Conga Grave
65	Timbal agudo	66	Timbal Grave	67	Agogo Agudo
68	Agogo grave	69	Cabasa	70	Maracas
71	Silvido corto	72	Silvido largo	73	Güiro corto
74	Güiro largo	75	Claves	76	Caja china aguda
77	Caja china grave	78	Cúica (Mute)	79	Cúica
80	Triángulo (Mute)	81	Triángulo		

Tabla 7: Valores MIDI de Instrumentos de percusión

Vamos a mostrar una imagen, de la estructura de una batería típica, con cada uno de los instrumentos etiquetados, para que así puedan conocer cada algunos de los instrumentos mencionados en la tabla anterior.



Ilustración 35: Estructura de batería

# 4. Análisis

## 4.1 Requisitos Funcionales

- El sistema abrirá una ventana cuando el usuario pulse sobre el botón de Nota, Trémolo, Calderón o Silencio.
- Al pulsar el botón de Nota el sistema pedirá al usuario la intensidad y el nodo de la nota.
- Al pulsar el botón de Trémolo el sistema pedirá al usuario el tipo de trémolo, la duración y las intensidades y velocidades del mismo.
- Al pulsar el botón de Silencio el sistema pedirá al usuario la duración del silencio.
- Al pulsar el botón de Calderón el sistema pedirá al usuario el fragmento de partitura a repetir y el número de repeticiones.
- El sistema representará gráficamente la partitura.
- El sistema permitirá la variación del tempo de la partitura.
- Al pulsar sobre el botón “Añadir” el sistema almacenará la acción del usuario.
- Al pulsar sobre el botón “Deshacer” el sistema eliminará la última acción realizada por el usuario.
- Al pulsar el botón de reproducir, el sistema pasará a la creación del archivo MIDI.
- Al pulsar el botón de “Reproducir” el sistema nos mostrará un Explorador de Archivos para almacenar el archivo creado.

## 4.2 Requisitos No Funcionales

- Podrá incluirse una pista de audio para trabajar sobre ella.
- El resultado podrá ser exportable en formatos de audio y .pdf.
- Fondo beige para atenuar el brillo de la pantalla. · Zoom graduable. · Posibilidad de incluir textos.
- Ejecutable en Windows, y a ser posible, en Linux y Mac.
- El sistema controlará los valores introducidos por el usuario, no permitiendo que

introduzca valores no deseados.

- Incorporará el diseño de la partitura de forma gráfica.
- Interfaz sencilla de entender.
- Podrá ejecutar el archivo MIDI con reproductores externos.

### 4.3 Estructura de Datos

En esta sección vamos a ver cómo podemos transformar la notación de Julio Omella en una estructura de datos eficiente.

Como podemos imaginar, una clase principal será la clase Nota, esta clase será la encargada de representar el sonido de un golpeo de la baqueta sobre cualquier instrumento.

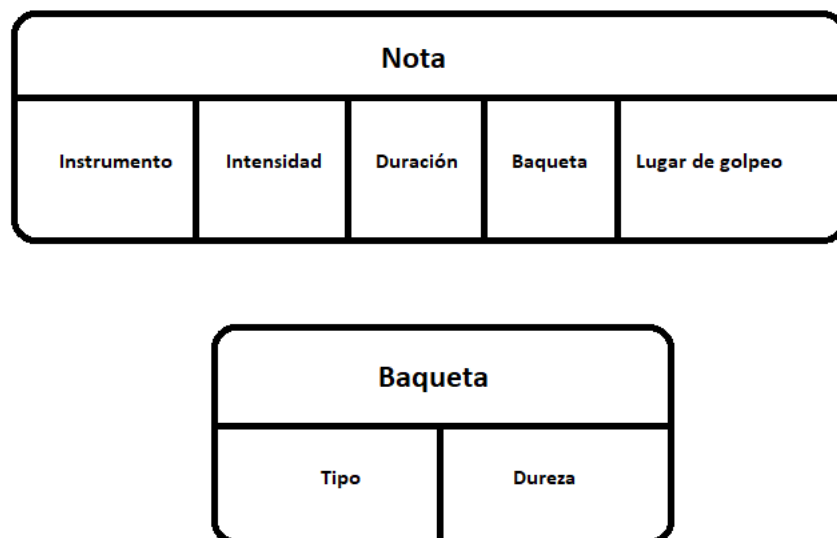
Procedamos a analizar los componentes principales de la clase Nota, y estos pueden ser:

- Instrumento sobre el que vamos a dar la nota.
- Duración de la nota.
- Intensidad.
- Lugar en el que golpearemos el instrumento.
- Baqueta con la que golpearemos el instrumento.

Podemos decir, sobre una primera pasada que estos van a ser los componentes principales de esta clase.

Otra clase, será la clase Baqueta, la cual nos permitirá cambiar entre los diferentes tipos de baqueta que puedan existir, cambiando las durezas de la misma y el material.

Por lo tanto, tendremos una clase Nota, la cual a su vez tendrá un atributo de la clase Baqueta, que nos dará diferentes tipos de sonido.

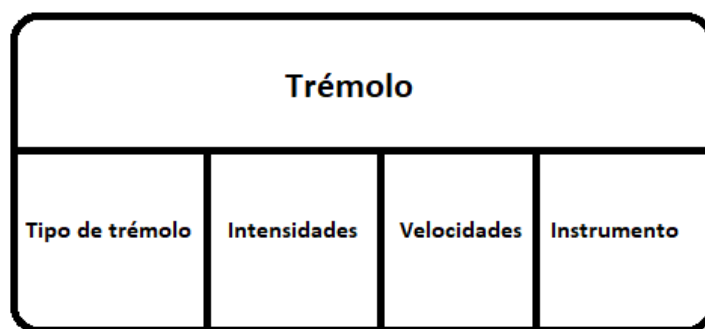


*Ilustración 36: Atributos principales de las clases Nota y Baqueta*

Por otra parte, estará la clase Partitura, una partitura como sabemos está compuesta por varias notas, además de las diferentes anotaciones que se puedan hacer. De esta forma podemos ver que ya tenemos tres clases diferenciadas.

Ahora nos queda trabajar con los Trémolos. Un trémolo es un tipo de evento diferente, no es una simple Nota, entonces, podemos tener diferentes tipos de Tremolo, que podremos almacenarlos en un Enumerado y por otra parte la clase Tremolo.

Esta clase estará compuesta por un tipo de Trémolo, y por una serie de atributos que nos identificarán el trémolo, como un vector de intensidades, velocidades de inicio y final y el instrumento que deseamos tocar.



*Ilustración 37: Atributos principales de la clase Trémolo*

Con esto, podemos decir que tenemos las clases principales para comenzar a realizar la representación de la notación de Julio Omella en nuestro programa.

## 5. Diseño

El planteamiento de nuestro sistema es el siguiente:



*Ilustración 38: Planteamiento del sistema*

El lenguaje que se ha elegido para la traducción de la notación es C++ debido a que los plugins de Cubase y VST también se encuentran programados en este lenguaje.

A continuación, veremos los diagramas de clases de los diferentes archivos creados para el proyecto.



## 5.1 Diagrama de clases MIDI y Notación

### Traducción de la notación de Julio Omella

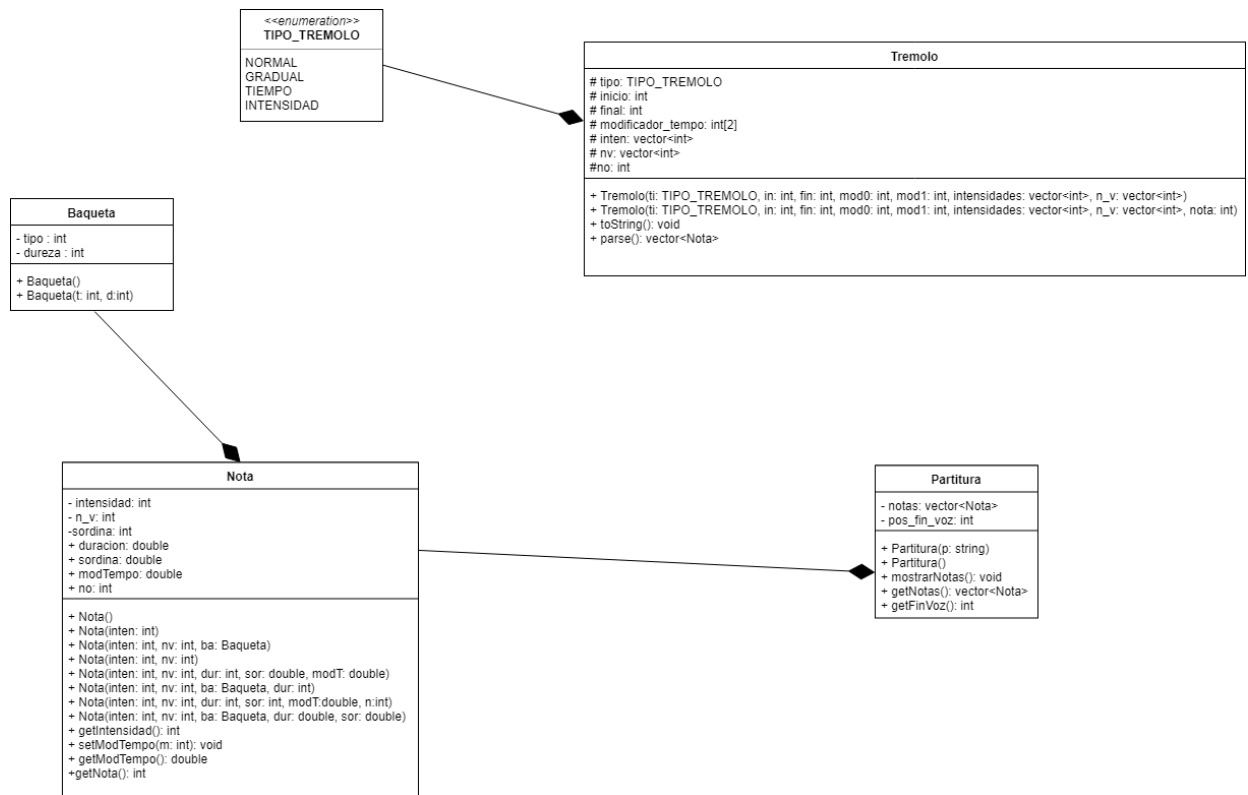


Ilustración 39: Diagrama de clases notación

Este diagrama de clases representa la estructuración de las clases creadas tras realizar la traducción de la notación de Julio Omella.

Como ya describimos anteriormente, en la estructura de datos que podíamos crear de esta notación, vemos que hemos creado una clase **Baqueta**, que presenta una relación de composición con la clase **Nota**, como ya intuíamos, y a su vez vemos cómo la clase **Partitura** tiene una relación de composición con la clase **Nota**.

Esto se debe a que una partitura, estará compuesta por varias notas.

A su vez, tendremos la clase **trémolo**, que sabemos que es un evento especial en la creación de nuestras partituras, la cual tendrá una relación de composición con el enumerado que nos indicará el tipo de trémolo en cada caso.

Esta clase, nos va a servir para trabajar de una forma más sencilla con este tipo de eventos, pero que al fin y al cabo, serán una consecución rápida de golpes sobre el instrumento, es decir, una serie de **Notas**.

## Diagrama de clase del diseño para conversión a formato MIDI

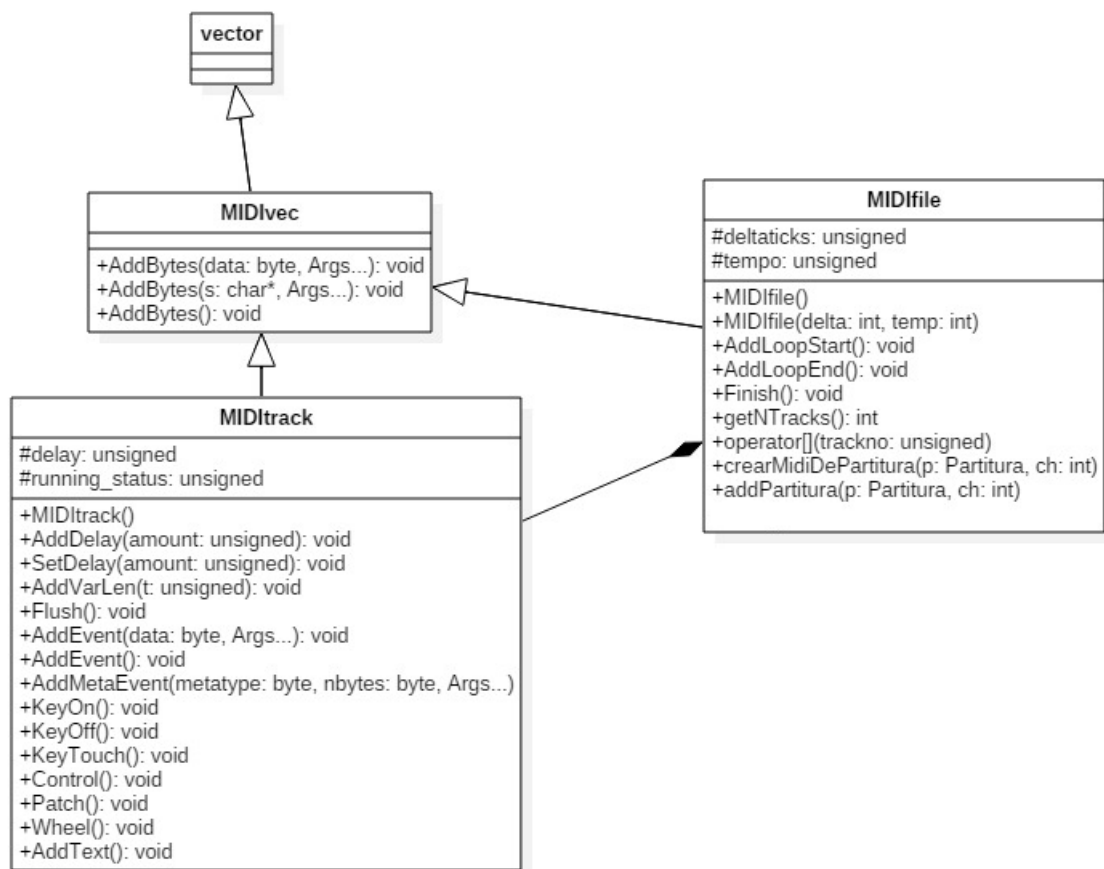


Ilustración 40: Diagrama de clases MIDI

Este diagrama de clases nos va a permitir entender cómo se relacionan las diferentes clases existentes y utilizadas para formar nuestro archivo MIDI.

Vemos una primera clase padre, que la mostraremos para ver de dónde hereda la primera clase que vamos a utilizar, que será la clase **"MIDIVec"**. Esta clase heredarán de la clase **Vector** y será la encargada de ir añadiendo los bytes necesarios para la creación de nuestro archivo MIDI. De esta clase, heredarán otras dos clases que nosotros utilizaremos, estas dos clases son **"MIDIFile"** y **"MIDITrack"**.

La clase **MIDITrack** es la que se va a encargar de codificar los eventos MIDI y crear nuestra pista. Por lo tanto, será la que nos permitirá ir ordenando todos los eventos MIDI que necesitemos para crear nuestra pista de audio.

Por otro lado, tendremos la clase **MIDIFile**, esta clase será la que nos va a permitir crear nuestro archivo MIDI.

Un archivo MIDI, está compuesto por una o varias pistas MIDI, que serán las que contendrán los sonidos y eventos del archivo, por eso existe esa relación de composición entre estas dos clases. Por lo tanto, **MIDIFile** será la clase que nos permitirá ordenar las pistas que disponemos, y estructurarlas de tal forma que puedan ser interpretadas posteriormente por los reproductores de archivos MIDI.

Estas dos partes se comunicarán a través de método que dispone la clase MIDIFile, **“crearMIDIdePartitura()”** en el que podemos ver que como parámetro debemos pasarle un objeto de la clase Partitura y a partir de esta, crearemos el archivo MIDI correspondiente.

Las siguientes clases están creadas para la Interfaz Gráfica.



Existe una clase principal, de la que van a heredar una gran parte de las clases presentes en nuestro proyecto. Esta clase es la clase “**QDialog**” la cual nos permitirá mostrar cuadros de diálogo. Todas las clases que heredan de esta clase son aquellas que nos mostrarán cuadros de diálogo para pedirle al usuario que introduzca los datos necesarios para la acción que desea, es decir, serán aquellas clases que al pulsar sobre el botón de Nota, Calderón, Silencio o Trémolo nos abrirán un cuadro de diálogo para pedirle al usuario que nos introduzca la intensidad, duración, velocidades, etc. Dependiendo de los datos necesarios para la acción a realizar.

Por otro lado, tendremos la clase **Almacenar**, que hereda de la clase **QWidget** y la clase principal **QtGuiApplication1** que herederá de **QMainWindow**.

Esto se debe a que será la clase principal, aquella que se ejecutará al abrir la aplicación y que contiene todo el grueso de la interfaz.

Al ser la aplicación principal, vemos que se relaciona con el resto de clases, debido a que en esta clase dispondremos de un objeto para cada clase, los cuales iremos llamando cuando el usuario pulse el botón correspondiente.

Para entender mejor el funcionamiento y la relación existente entre las diferentes clases de nuestro proyecto, vamos a realizar un diagrama de secuencia de cómo sería la creación de un archivo MIDI con únicamente una nota en él.

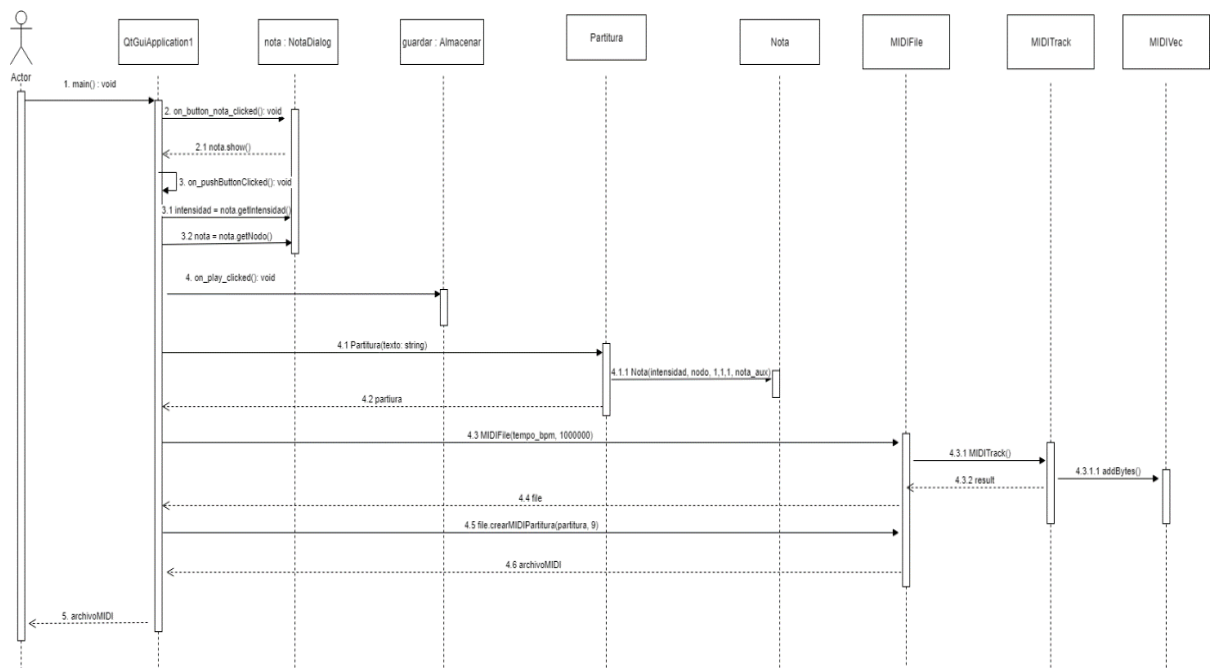


Ilustración 42: Diagrama de secuencia

## 6. Implementación

En este capítulo, veremos cómo se ha implementado todas las tareas descritas anteriormente, tanto la transposición de la notación de Julio, como del estándar MIDI y la Interfaz Gráfica.

### 6.1 MIDI

#### 6.1.1 Clase MIDIVec

Esta clase actúa de wrapper sobre `std::vector<byte>`, permitiéndonos que sea posible realizar push-backs de múltiples bytes en una sola llamada.

- **Class MIDIVec : public std::vector<byte>**
  - **Void addBytes(...):** Gracias a esta plantilla y a la funcionalidad de “template <typename ...Args>” podremos realizar en la llamada a “addBytes” con distintos argumentos lo que explicamos anteriormente de la clase MIDIVec, es decir, podremos añadir con una sola llamada los bytes que necesitemos sin problema.

#### 6.1.2 Clase MIDITrack

Esta clase será la encargada de codificar los eventos en un track de MIDI.

Los distintos métodos y atributos que posee esta clase son:

- **unsigned delay, running status:** Variable delay, que nosotros llamamos en las tablas de la sección 3, valor retraso.
- **void addDelay(unsigned amount):** Podremos sumar una cantidad al delay actual.
- **void setDelay(unsigned amount):** Método para establecer el valor de la variable delay.
- **unsigned getDelay():** Método getter que nos permitirá obtener el valor de la variable delay.
- **void addVarLen(unsigned t):** Método que prepara la variable delay para su uso en el nodo.
- **void flush():** Este método introduce el valor delay en el nodo.
- **void addEvent(...):** Este método utiliza la misma teoría que el método addBytes() de MIDIVec, pero en este caso para añadir nodos eventos en el track MIDI.
- **void addMetaEvent(...):** Método que utiliza la misma teoría que el método addBytes() de MIDIVec, pero en este caso para añadir nodos meta-eventos en el track MIDI.
- **Eventos y meta-eventos que poseen su propia función para ser añadidos de forma más intuitiva:**
  - `keyOn(int ch, int n, int p)`
  - `keyOff(int ch, int n, int p)`

- keyTouch(int ch, int n, int p)
- control(int ch, int c, int v)
- patch(int ch, int patchno)
- Wheel(int ch, unsigned value)
- addText(int texttype, const char\* text)

### 6.1.3 Clase MIDIFile

Esta clase encapsulará los métodos necesarios para poder crear un archivo MIDI.

- **std::vector<MIDITrack> tracks:** En este vector almacenaremos las distintas Tracks que posea el archivo MIDI. Debemos tener en cuenta que el formato de nuestro archivo es 1, y por lo tanto, las pistas serán síncronas.
- **unsigned deltaTicks, tempo:** Esto son variables requeridas para la inicialización del nodo cabecera MIDI, nos permitirán indicar el tempo de la canción.
- **MIDIFile():** Constructor por defecto con valores predeterminados.
- **MIDIFile(int delta, int tempo):** Constructor con el que determinamos deltaticks y el tempo de la partitura.
- **void addLoopStart():** Esta función llama a “addText()” para introducir la etiqueta “LoppStart”.
- **void addLoopEnd():** Función que llama a “addText()” para introducir la etiqueta “LoopEnd”.
- **MIDITrack& operator[] (unsigned trackno):** Vamos a definir el operador corchete, para poder trabajar de forma más cómoda con las tracks, de forma que cuando accedamos a una posición, si este track no existe, crearlo.
- **Finish():** Función que se llama para dar el último toque para crear el archivo MIDI. Se encargará de introducir el nodo cabecera MIDI que deben tener todos los archivos MIDI, y posteriormente cada uno de los tracks creados.
- **GetNTracks():** Devuelve el número de tracks actuales.
- **crearMidiDePartitura(Partitura p, int ch):** Este método se encargará de crear un archivo MIDI a partir de una partitura y un canal pasado.
- **addPartitura(Partitura p, int ch):** Con este método añadimos la partitura pasada al vector de tracks.

## 6.2 Partitura

### 6.2.1 Clase Baqueta

Clase que nos va a permitir definir una baqueta, de acuerdo a lo que vimos en el análisis de la misma.

- **Int tipo, int dureza:** Estas son las dos variables deseadas en la baqueta.
- **Baqueta():** Constructor con valores por defecto
- **Baqueta(int t, int d):** Constructor que nos va a permitir declarar una baqueta según los valores pasados.

### 6.2.2 Clase Nota

Esta clase nos va a permitir crear las diferentes notas según las indicaciones que deseemos, de intensidad, baqueta y nodo (vientre).

- **Int intensidad, n\_v, no:** Estas variables, nos van a permitir determinar, por orden, la intensidad de la nota (irá de 0 a 127), el nodo o el vientre de la nota [0-5], siendo por defecto el valor 2, que correspondería a V1 en la imagen que vimos de la notación de Julio Omella, y por último el identificador de la nota, que nos indicará el instrumento a tocar.
- **Baqueta b:** Variable que nos dirá la baqueta usada para ejecutar dicha nota.
- **double duracion, sordina, modTempo:** Variables que nos permiten añadir información adicional a la nota.
- **Constructores de Nota(...):** Tendremos varios constructores para los distintos valores por defecto o bien los pasados por argumento.
- **int getIntensidad():** Método getter de la variable intensidad.
- **void setModTempo(int t):** Método para definir la variable modTempo.
- **int getModTempo():** Método getter de la variable modTempo.
- **int getNota():** Getter del identificador nota, en concreto, la variable “no”.

### 6.2.3 Enum TIPO\_TREMOLO

Este enumerado tendrá los distintos tipos de trémolo que pueden ser:

- NORMAL
- GRADUAL
- GRADUALTIEMPO
- GRADUALINTENSIDAD

### 6.2.4 Clase TREMOLO

Esta clase nos va a permitir abstraer la definición de trémolo que nos ofrece Julio, que como ya comentamos no se corresponde con un único evento MIDI. Gracias a esta clase podremos hacer referencia a un simple trémolo en singular, sin necesidad de tener que trabajar siempre con todos los eventos que lo incluyan, haciendo este trabajo dentro de la clase.

- **TIPO\_TREMOLO t:** Variable que nos va a determinar el tipo de trémolo con el que estamos trabajando.
- **int inicio, final, no :** Las variables “inicio” y “final” nos van a permitir determinar la longitud del trémolo, mientras que la variable “no” nos indicará el identificador del instrumento MIDI sobre el que se realizará el trémolo.
- **int[2] modificador\_tempo:** Para los tipos de tremolo NORMAL y GRADUALINTENSIDAD, ambos valores serán iguales, debido a que en ellos no se modificará la velocidad con la que se realiza el trémolo.  
Para el tipo de trémolo GRADUALINTENSIDAD, ambos valores nos indicarán la velocidad de l trémolo al inicio y al final, de forma que el trémolo variará la velocidad de los golpes de forma gradual desde el inicio al final.
- **vector<int> inten, nv:** Estas variables nos indicarán los valores de intensidad y de nodo y vientre con las que se realizarán las notas del trémolo.
- **Constructores Tremolo(...):** Dispondremos de diferentes constructores para inicializar las variables anteriormente comentadas.
- **void toString():** Función que nos permite sacar por consola toda la información del trémolo.
- **Vector<Nota> parse():** Esta es la función más importante de esta clase. Esta función se encargará de transformar la clase trémolo con todas sus variables en un vector compuesto por elementos de la clase “Nota”.

Vamos a ver cómo funciona esta transformación para los distintos tipos de Tremolos.

### TREMOLO NORMAL

```
for (int i = 0; i < final; i++) {
    for (int j = 0; j < modificador_tempo[0]; j++) { // TODO: modificar tempo, hacer media de notas
        double modif = modificador_tempo[0];

        int intensidadFinal;
        if (inten[i] < inten[i + 1]) {
            intensidadFinal = inten[i] + (j * ((inten[i + 1] - inten[i]) / modif));
        }
        else intensidadFinal = inten[i] - (i * ((inten[i] - inten[i + 1]) / modif));
        Nota aux(intensidadFinal, nv[i], 1, 1, modif, no);

        //Nota aux(inten[i], nv[i], 1, 1, modif, no);
        res.push_back(aux);
    }
}
```

*Ilustración 43: Ejemplo de transformación de Tremolo NORMAL*

Como podemos ver, para el tipo de trémolo NORMAL, esta transformación es sencilla, ya que lo que haremos será con un bucle controlar que el trémolo dura el tiempo que se le ha indicado, y con un segundo bucle en cada tiempo introducir un número de notas que dependerá de la velocidad con la que se vaya a realizar este trémolo.



## TREMOLO GRADUAL-TIEMPO

```

if (modificador_tempo[0] < modificador_tempo[1]) {
    float tamBloque = (inicio*1.0) / (modificador_tempo[1] - modificador_tempo[0]);
    for (int i = modificador_tempo[0]; i < modificador_tempo[1]-1; i++) {
        double v = i;
        double start = 1 / v;
        double t = ((tamBloque - start)*(v + 0.5)) + 1;
        double auxx = ((1 / v) - (1 / (v + 1))) / t;
        for (int j = 0; j < t; j++) {
            Nota aux(inten[i], nv[i], 1, 1, v, no);
            res.push_back(aux);
            start -= auxx;
            v = 1 / start;
        }
    }
}
else {
    float tamBloque = (inicio*1.0) / (modificador_tempo[0] - modificador_tempo[1]);

    for (int i = modificador_tempo[0]-1; i > modificador_tempo[1]; i--) {
        double v = i;
        double start = 1 / v;
        double t = ((tamBloque - start)*(v - 0.5));
        double auxx = ((1 / (v - 1)) - (1 / (v))) / t;
        for (int j = 0; j < t; j++) {
            Nota aux(inten[i], nv[i], 1, 1, v, no);
            res.push_back(aux);
            start += auxx;
            v = 1 / start;
        }
    }
}
}

```

Ilustración 44: Ejemplo de código de transformación de trémolo GRADUAL-TIEMPO

El trémolo gradual de tiempo nos va a permitir modificar gradualmente la velocidad con la que se realizan los golpes en cada tiempo. Para verlo gráficamente, se va a mostrar un ejemplo de partitura MIDI, en el que se realiza este tipo de trémolo, para que se pueda observar la variación de golpes en cada tiempo.



Ilustración 45: Vista de trémolo GRADUAL-TIEMPO en Cubase

Como se observa fácilmente, vemos que la velocidad de los golpes va aumentando al ir disminuyendo la separación entre los golpes.

Debido a que la velocidad puede aumentar o disminuir, por eso debemos diferenciar primero con una comprobación si lo que queremos es aumentar o disminuir la velocidad.

Hecho esto, obtendremos el tamaño del bloque, que dependerá de la variación de velocidad que queramos dar.

Conocido esto, al igual que con el trémolo NORMAL, con dos bucles controlaremos el número de notas a introducir en cada tiempo, así como que no se sobrepase la longitud del trémolo.

## TREMOLO GRADUAL-INTENSIDAD

```
double tamBloque = inicio/3;

for (int i = 0; i < nv.size(); i++) {
    double v = modificador_tempo[0];
    double start = 1 / v;
    double t = ((tamBloque - start) * (v + 0.5)) + 1;
    double auxx = ((1 / v) - (1 / (v + 1))) / t;
    for (int j = 0; j < t; j++) {
        Nota aux(inten[i], nv[i], 1, 1, v, no);
        res.push_back(aux);
        start -= auxx;
        v = 1 / start;
    }
}
```

Ilustración 46: Ejemplo de código de Trémolo GRADUAL-INTENSIDAD

En este tipo de trémolo vamos a variar la intensidad de los golpes de una forma gradual, que podemos ver gráficamente en la siguiente imagen, extraída de una partitura MIDI representada con el programa Cubase.

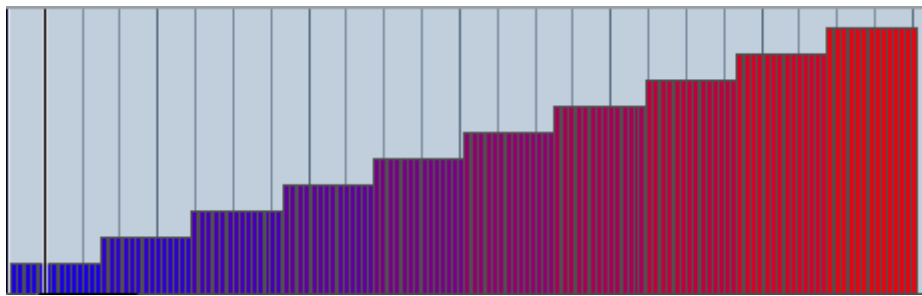


Ilustración 47: Vista de la variación de intensidad de un Trémolo GRADUAL-INTENSIDAD en Cubase

El trémolo Gradual de Intensidad se trabaja prácticamente igual que el trémolo Gradual de Tiempo, con la única diferencia de que aquí el tamaño del bloque, así como la variable “v” que nos servirá para indicar la separación entre las notas, serán fijos, ya que el tiempo no varía y lo único que varía es la intensidad de las notas, que al estar ya almacenada en el vector “inten<>” no debemos hacer más cálculos, simplemente obtener el valor almacenado.

### 6.2.5 Clase Partitura

La clase partitura es la encargada de transformar la notación introducida por el usuario en el formato necesario para que el programa pueda interpretar qué es lo que necesita el usuario para poder transformarlo posteriormente en el archivo MIDI.

- **Vector<Nota> notas:** Variable que se encargará de tener almacenadas cada una de las notas que se introduzcan en la partitura.
- **int pos\_fin\_voz:** Variable que nos permitirá identificar el final de un track, en el caso de que el MIDI disponga de varios tracks.
- **Partitura():** Constructor de la clase partitura que nos va a permitir crear una partitura

desde la consola, de forma interactiva. Se nos irán realizando preguntas una a una para conocer qué es lo que queremos introducir en cada caso. Este constructor no se utiliza en la aplicación final.

- **Partitura(string p):** Este constructor recibirá un string con todo lo que se debe introducir en el vector de notas, es decir, los trémolos, notas, calderones y silencios que existan en ese track, así como las características de cada evento.

```
if (str3.find("Nota") != -1) {
    int intensidad = 0;
    int nodo = 0;
    int nota_aux = 0;
    int paso = 1;
    pos = p_aux.find("-");
    string spe = str3.substr(pos + 1);
    string aux = "";
    for (int i = 0; i < spe.size(); i++) { // En este bucle nos quedamos únicamente con las características de la nota
        if (spe.at(i) != ' ')
            aux += spe.at(i);
        else {
            switch (paso) {
                case 1:
                    paso = 2;
                    intensidad = stoi(aux); // Intensidad de la nota
                    break;
                case 2:
                    paso = 3;
                    nodo = stoi(aux); // Nodo
                    break;
            }
            aux = "";
        }
    }
    nota_aux = stoi(aux); // Nota a reproducir
    Nota n_aux(intensidad, nodo, 1, 1, 1, nota_aux);
    notas.push_back(n_aux);
}
```

*Ilustración 48: Parte del constructor Partitura(String p)*

En este ejemplo podemos ver por ejemplo cómo se transforma el string recibido a algo que pueda ser interpretado como un objeto de la clase Nota. El string que recibe será algo como lo que vemos a continuación:

```
Nota-127 2 38
Tremolo-Intensidad 10 3 3 1 6 35
Calderon-4 3
Silencio
```

*Ilustración 49: Texto partitura*

Como vemos lo que se hace (nos centramos en el caso de Nota) es indicar qué acción queremos realizar y seguidamente las características de la acción.

De esta forma, lo que haremos será ir recorriendo el string quedándonos con cada una de las características indicadas, para posteriormente pasarlas al constructor de la clase Nota, o de la acción que se indique y añadir esto al vector de notas.

- **void mostrarNotas():** método que se encargará de imprimírnos por pantalla todo el vector de notas.
- **vector<Nota> getNotas():** método que nos va a devolver el vector de notas.
- **int getFinVoz():** método que nos devuelve el valor de la variable “pos\_fin\_voz”.

## 6.3 Clase QApplication1

Esta clase es la clase encargada de lanzar la interfaz gráfica y comunicarse con la clase Partitura, para poder crear nuestro archivo MIDI.

- **QTableWidget \*\*tab\_voz:** Widget de la interfaz gráfica que nos permitirá representar la partitura conforme se vaya creando.
- **string \*voces:** array que almacenará las diferentes voces creadas.
- **int voces\_size:** índice que nos dirá el número de voces totales.
- **string frase:** Esta variable será la que iremos guardando en el array de voces, que nos irá indicando en cada tiempo de la partitura qué se debe ejecutar.
- **int numero\_compases:** Variable que indicará el número de compases que hay en cada voz.
- **int botón\_pulsado:** variable que nos permitirá conocer qué botón de acción se ha pulsado (Nota, Tremolo, Calderón, Silencio).
- **Ui::QtGuiApplication1Class ui:**
- **NotaDialog \*nota:** variable de la clase NotaDialog que nos permitirá mostrar una segunda ventana cuando queramos introducir una nota.
- **TremoloDialog \*tremolo:** variable de la clase TremoloDialog que nos permitirá mostrar una segunda ventana cuando queramos introducir un tremolo.
- **SilencioDialog \*silencio:** variable de la clase SilencioDialog que nos permitirá mostrar una segunda ventana cuando queramos introducir un silencio.
- **CalderonDialog \*calderon:** variable de la clase CalderonDialog que nos permitirá mostrar una segunda ventana cuando queramos introducir un calderon.
- **Almacenar \*guardar:** variable de la clase Almacenar que nos permitirá mostrar un explorador de archivos para almacenar nuestro archivo MIDI donde deseemos.
- **void on\_pushButton\_clicked():** método encargado de almacenar la acción que ha indicado el usuario que quiere realizar, así como representarla.

Veamos un ejemplo de cómo funciona este método.

Primero comprobaremos qué botón se ha pulsado, viendo el indicador "botón\_pulsado".

Una vez comprobado, leeremos los valores que ha introducido el usuario, de la siguiente forma.

```

frase += "Nota-";
int intensidad = nota->getIntensidad();
frase += to_string(intensidad) + " ";

int nodo = nota->getNodo();
frase += to_string(nodo) + " ";

instrumento = (ui.comboBox->currentText()).toUtf8().constData();

```

*Ilustración 50: Obtener valores introducidos por el usuario*

Como vemos, para el caso de la nota, comenzaremos a construir el String que almacenaremos posteriormente, leyendo los valores de intensidad, nodo e instrumento que ha indicado el usuario.

Seguidamente, para representar la nota, comprobaremos la intensidad introducida, para representarla en una fila u otra de nuestra partitura.

```

if (intensidad <= 22) {
    fila = 5;
}
else if (intensidad > 22 && intensidad <= 44) {
    fila = 4;
}
else if (intensidad > 44 && intensidad <= 66) {
    fila = 3;
}
else if (intensidad > 66 && intensidad <= 88) {
    fila = 2;
}
else if (intensidad > 88 && intensidad <= 110) {
    fila = 1;
}
else if (intensidad >= 110) {
    fila = 0;
}
}

```

*Ilustración 51: Ejemplo de separación de fiñas en el hexagrama*

Finalmente, conocidos los valores de la nota y con el string creado, lo almacenamos en el array de voces y representamos en el “hexagrama” nuestra acción.

```

int f_aux = ui.tabWidget->currentIndex();

string voz_aux = voces[f_aux];
voz_aux += frase;

voces[f_aux] = voz_aux;

QTableWidget *tab_select = tab_voz[f_aux];
QString ins_tab = QString::fromStdString(instrumento);

tab_select->insertColumn(tab_select->columnCount());
QTableWidgetItem* item = new QTableWidgetItem;
item->setIcon(*nota_icon);
item->setText(ins_tab);
tab_select->setItem(fila, (tab_select->columnCount()-1), item);

```

*Ilustración 52: Representación de nota*

- **void on\_button\_nota\_clicked():** método para mostrar un cuadro de diálogo al crear una Nota.
- **void on\_button\_tremolo\_clicked():** método para mostrar un cuadro de diálogo al crear un Tremolo.
- **void on\_button\_silencio\_clicked():** método para mostrar un cuadro de diálogo al crear un Silencio.
- **void on\_button\_calderon\_clicked():** método para mostrar un cuadro de diálogo al crear un Calderón.
- **void on\_nueva\_voz\_clicked():** método que nos permitirá crear una nueva pestaña para una nueva voz en nuestra partitura.
- **void on\_play\_clicked():** método que nos permite guardar y crear nuestro archivo MIDI.

## 6.4 Clase Almacenar

Clase que nos permite mostrar un explorador de archivos para que el usuario pueda almacenar el archivo MIDI en el directorio que elija.

- **Ui::Almacenar ui:**
- **string ruta:** variable que almacena la ruta que ha elegido el usuario.
- **bool guardado:** variable que nos indica si se ha guardado el archivo ya, o no.
- **Almacenar(QWidget \*parent):** Constructor de la clase Almacenar.
- **string getPath():** getter de la variable “ruta”.
- **~Almacenar():** Destructor de la clase.

## 6.5 Clase CalderonDialog

Con esta clase podremos mostrar un cuadro de diálogo, con el que el usuario podrá introducir los valores necesarios para crear el calderón, como son el fragmento de la partitura a repetir, como la duración del calderón.

- **Ui::CalderonDialog ui:**
- **int fragmento:** variable que nos indicará el fragmento de partitura anterior que deseamos reproducir en el calderón.
- **int longitud:** variable que indica el número de veces que se repetirá el calderón.

- **CalderonDialog(QWidget \*parent):** Constructor de la clase CalderonDialog.
- **int getFragmento():** getter de la variable “fragmento”.
- **int getLongitud():** getter de la variable “longitud”.
- **void setMax(int maxi):** método que pondrá un límite máximo para que no se puedan coger fragmentos con más notas de las que existen en la partitura.
- **~CalderonDialog():** Destructor de la clase.

## 6.6 Clase NotaDialog

La clase NotaDialog será la encargada de mostrarnos un cuadro de diálogo cuando el usuario decida introducir una nota nueva en la partitura.

- **Ui::NotaDialog ui:**
- **int intensidad :** Atributo de la clase que nos almacenará la intensidad que tendrá la nota.
- **int nodo :** Atributo de la clase que nos permitirá almacenar el lugar de golpeo de la baqueta en el instrumento.
- **NotaDialog(QWidget \*parent):** Constructor de la clase NotaDialog.
- **int getIntensidad():** Método para obtener el valor de la variable “intensidad”.
- **int getNodo():** Método para obtener el valor de la variable “nodo”.
- **~NotaDialog():** Destructor de la clase.

## 6.7 Clase SilencioDialog

La clase SilencioDialog será la clase que nos mostrará el cuadro de diálogo con el que el usuario podrá indicar la duración que necesita para el silencio.

- **Ui::SilencioDialog ui:**
- **int longitud:** Variable para almacenar la longitud deseada para el silencio.
- **SilencioDialog(QWidget \*parent):** Constructor de la clase SilencioDialog.
- **int getLongitud():** Método por el cual obtendremos el valor de la variable “longitud”.
- **~SilencioDialog():** Destructor de la clase.

## 6.8 Clase TremoloDialog

Con la clase TremoloDialog podremos crear un cuadro de diálogo con el que obtener todos los

valores necesarios por parte del usuario para crear el Trémolo necesario.

- **Ui::TremoloDialog ui:**
- **string tipo\_tremolo:** Variable donde almacenaremos el tipo de trémolo que desea el usuario.
- **int longitud:** Variable para almacenar la longitud del trémolo.
- **int intensidad\_ini, intensidad\_fin:** Variables para almacenar la intensidad de inicio y final del trémolo.
- **int velocidad\_ini, velocidad\_fin:** Variables para almacenar la velocidad de inicio y final del trémolo.
- **TremoloDialog(QWidget \*parent):** Constructor de la clase TremoloDialog.
- **int getLongitud():** Método para obtener el valor de la longitud del trémolo.
- **int getVelocidadIni(), int getVelocidadFin():** Métodos para obtener las velocidades de inicio y final del trémolo.
- **int getIntensidadIni(), int getIntensidadFin():** Métodos para obtener las intensidades de inicio y final del trémolo.
- **string getTipoTremolo():** Método para obtener el tipo de trémolo.
- **~TremoloDialog():** Destructor de la clase.
- **on\_comboBox\_currentIndexChanged():** Este método nos servirá para obtener cuándo el usuario cambia el tipo de trémolo que desea para habilitar y deshabilitar las diferentes opciones de trémolo, ya que, por ejemplo, para el trémolo de intensidad no habrá variación de velocidad.  
Esto podemos verlo en la siguiente imagen, en la que vemos que dependiendo del tipo de tremolo, habilitamos y deshabilitamos las entradas del usuario correspondientes.

```
void TremoloDialog::on_comboBox_currentIndexChanged() {
    string tipo = (ui.comboBox->currentText()).toUtf8().constData();

    if (tipo == "Tremolo Intensidad") {
        ui.spinBox_3->setEnabled(false); // Deshabilitamos velocidad fin
        ui.spinBox_4->setEnabled(true); // Habilitamos intensidad inicio
        ui.spinBox_5->setEnabled(true); // Habilitamos intensidad fin
    }
    else if (tipo == "Tremolo Velocidad") {
        ui.spinBox_3->setEnabled(true); // Habilitamos velocidad fin
        ui.spinBox_5->setEnabled(false); // Deshabilitamos intensidad fin
    }
    else {
        ui.spinBox_3->setEnabled(false); //Deshabilitamos velocidad fin
        ui.spinBox_5->setEnabled(false); // Deshabilitamos intensidad fin
    }
}
```

*Ilustración 53: Cambiar entradas de usuario según el tipo de trémolo*



Indicar además que en cada clase, se tiene un método correspondiente para cada botón que exista en ese cuadro de diálogo, que será el encargado de almacenar los datos correspondientes que ha introducido el usuario y cerrar el cuadro de diálogo.

## 7. Instalación y Configuración

En esta sección veremos cómo instalar y configurar los programas externos necesarios para poder pasar a realizar las pruebas de nuestra aplicación.

### 7.1 Cubase 8 y Kontakt 5

Cubase es una serie de aplicaciones informáticas capacitadas para editar audio digital, MIDI y secuenciadores de música, creadas originalmente por Steinberg.

Kontakt es el sampler, es decir, un instrumento musical virtual que utiliza grabaciones (“samples”) cargadas por el usuario para ser reproducidas mediante un dispositivo para interpretar o componer música. Kontakt es el más potente sampler que existe actualmente. Posee un sofisticado motor sonoro, una amplia variedad de efectos, así como potentes opciones de modulación que nos permitirán sacar un sonido lo más realista posible.

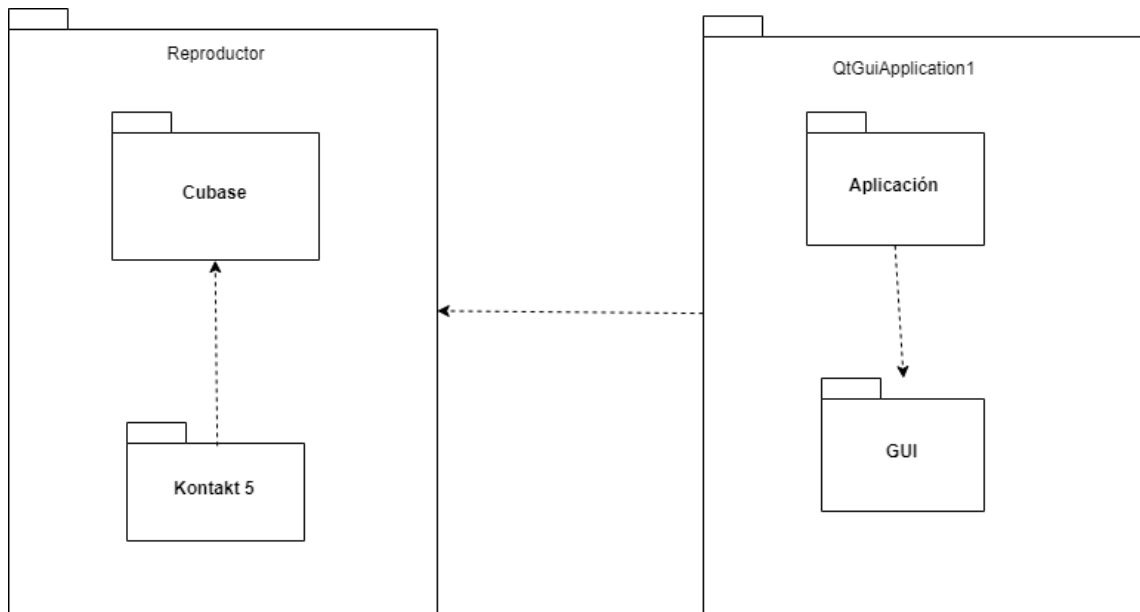
Ambos programas, anteriormente mencionados, son programas de pago, por lo que nosotros utilizaremos una Demo que nos permita probar nuestras creaciones de archivos MIDI.

Kontakt tiene una aplicación gratuita, que es “Kontakt 5 Player” que nos permitirá tocar diferentes instrumentos, en nuestro caso instrumentos de percusión.

Lo primero que haremos será descargar Cubase ya que Kontakt debemos integrarlo dentro de él. Al descargar Kontakt obtendremos tres archivos con extensión “.dll” que tendremos que integrar dentro de la carpeta de instalación de Cubase, concretamente en la carpeta “VSTPlugins”. Con esto integrado ya podremos correr nuestros archivos MIDI dentro de Cubase con un sonido más real.

Esta parte consistirá en la instalación y configuración de Cubase y Kontakt 5, para poder reproducir nuestra partitura MIDI creada. Lo que tendremos que hacer será descargar Kontakt 5 y Cubase desde su página oficial. Kontakt 5 es Free Software, por lo que únicamente tendremos que registrarnos en la página para descargarlo mientras que de Cubase necesitaremos usar la versión de evaluación o introducir un código de licencia en el caso de que lo tengamos.

Teniendo descritas tanto la parte que usaremos de reproductor como la parte de nuestra aplicación, vamos a ver más gráficamente las dependencias entre las distintas partes que vamos a utilizar en este proyecto.



*Ilustración 54: Diagrama de paquetes*

Con este diagrama podemos ver de una forma esquemática las diferentes partes del proyecto, tendremos un primer paquete que será el que veremos como reproductor que incluirá a Cubase y a Kontakt 5, sabiendo que entre ambos existe una dependencia, ya que será necesario incluir Kontakt 5 en Cubase como veremos a continuación.

Por otra parte tendremos nuestra aplicación, que estará formada por dos partes, que serán la interfaz gráfica y la funcionalidad del programa, sabiendo que dentro de la interfaz tendremos que incluir el programa para darle la funcionalidad que deseamos.

Finalmente, entre estos dos paquetes globales, existirá una dependencia, ya que para poder reproducir un archivo en Cubase, será necesario primero crearlo y por lo tanto, necesitaremos primero tener el archivo generado por “QtGuiApplication1” para ejecutarlo en el “Reproductor”.

Instalado tanto Cubase como Kontakt, vamos a ir a la ubicación en la que se ha instalado Kontakt, aquí tendremos las bibliotecas de enlace dinámico (.dll) que serán las que necesitemos para integrar Kontakt 5 en Cubase. Estos archivos se encontrarán dentro de la carpeta “VSTPlugins 32 bit” como se muestra en la imagen.




Este equipo > Disco local (C:) > Archivos de programa > Native Instruments > VSTPlugins 32 bit				
Nombre	Fecha de modifica...	Tipo	Tamaño	
 Kontakt 5 8out.dll	13/09/2017 12:53	Extensión de la apl...	42 KB	
 Kontakt 5 16out.dll	13/09/2017 12:53	Extensión de la apl...	42 KB	
 Kontakt 5.dll	13/09/2017 12:53	Extensión de la apl...	59.264 KB	

Ilustración 55: Ubicacion DLLs Kontakt 5

Kontakt además puede ejecutarse él solo, es decir, no necesita ningún otro programa para poder abrirse, aunque con esta funcionalidad no podemos cargar archivos MIDI, que es lo que nosotros necesitamos, por eso es necesario Cubase.

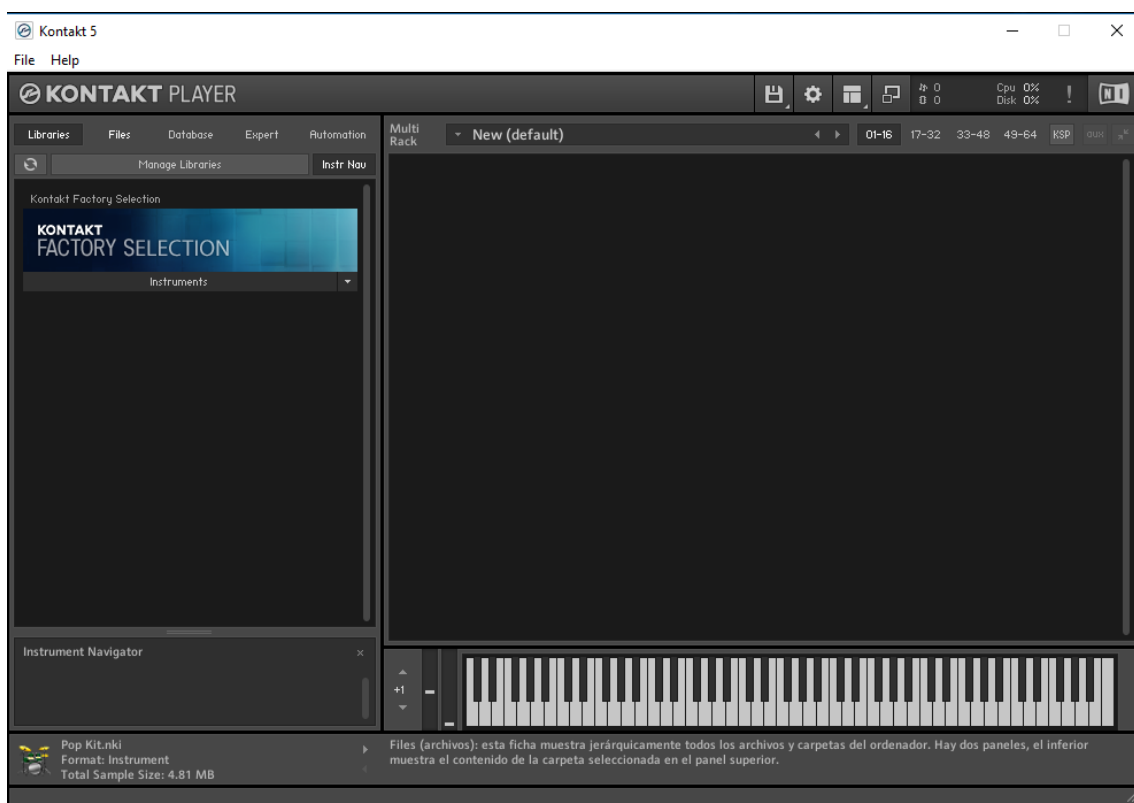


Ilustración 56: Pantalla principal Kontakt 5

Por otro lado, ahora tendremos que ir a la ubicación de la instalación de Cubase para agregar los DLLs de Kontakt e integrar este instrumento VST en Cubase.

Este equipo > Disco local (C:) > Archivos de programa > Steinberg > Cubase LE AI Elements 8 > VSTPlugins				
Nombre	Fecha de modifica...	Tipo	Tamaño	
Distortion	14/05/2025 9:12	Carpeta de archivos		
Filter	14/05/2025 9:12	Carpeta de archivos		
Modulation	14/05/2025 9:12	Carpeta de archivos		
Crystal.dll	05/04/2016 14:00	Extensión de la apl...	11.265 KB	
Kontakt 5 8out.dll	13/09/2017 12:57	Extensión de la apl...	50 KB	
Kontakt 5 16out.dll	13/09/2017 12:57	Extensión de la apl...	50 KB	
Kontakt 5.dll	13/09/2017 12:58	Extensión de la apl...	65.038 KB	
NewProject.dll	14/05/2018 16:46	Extensión de la apl...	2.688 KB	
Tunefish4.dll	22/07/2017 14:27	Extensión de la apl...	4.622 KB	

Ilustración 57: Ubicación VSTPlugins de Cubase

Dentro de la carpeta de instalación de Cubase, tendremos una carpeta llamada “VSTPlugins” que será en la que tendremos que introducir nuestros archivos de Kontakt para integrarlo. Ahora si abrimos Cubase y buscamos los instrumentos VST que dispone, vemos que nos aparece Kontakt 5 y por lo tanto quiere decir que la integración ha sido correcta.

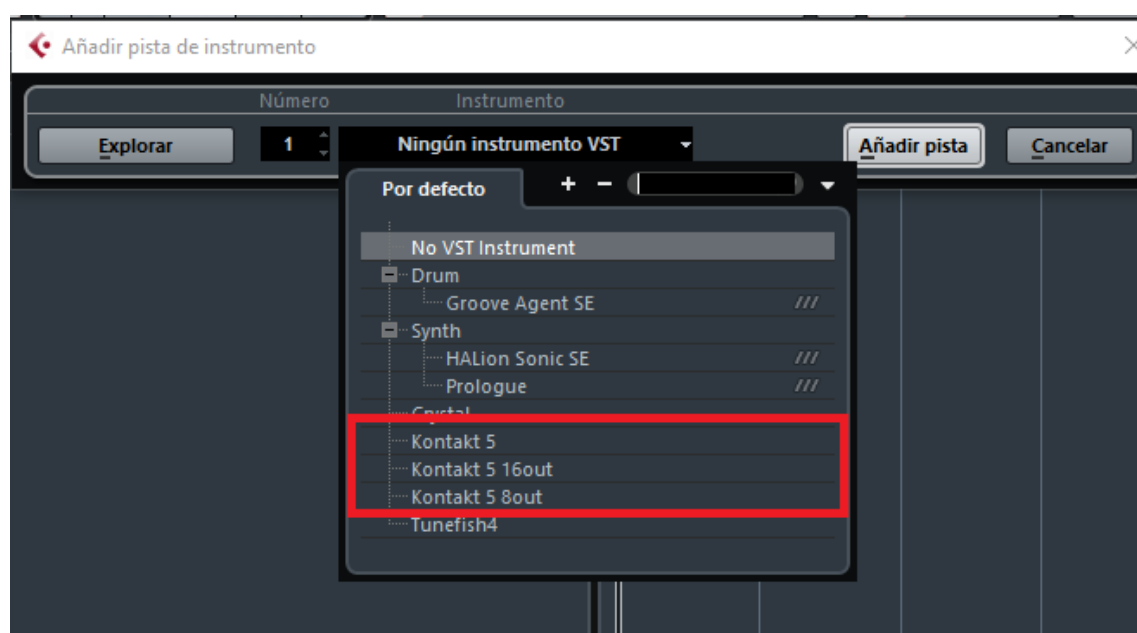


Ilustración 58: Kontakt 5 como VST Instrument en Cubase

## 8. Pruebas

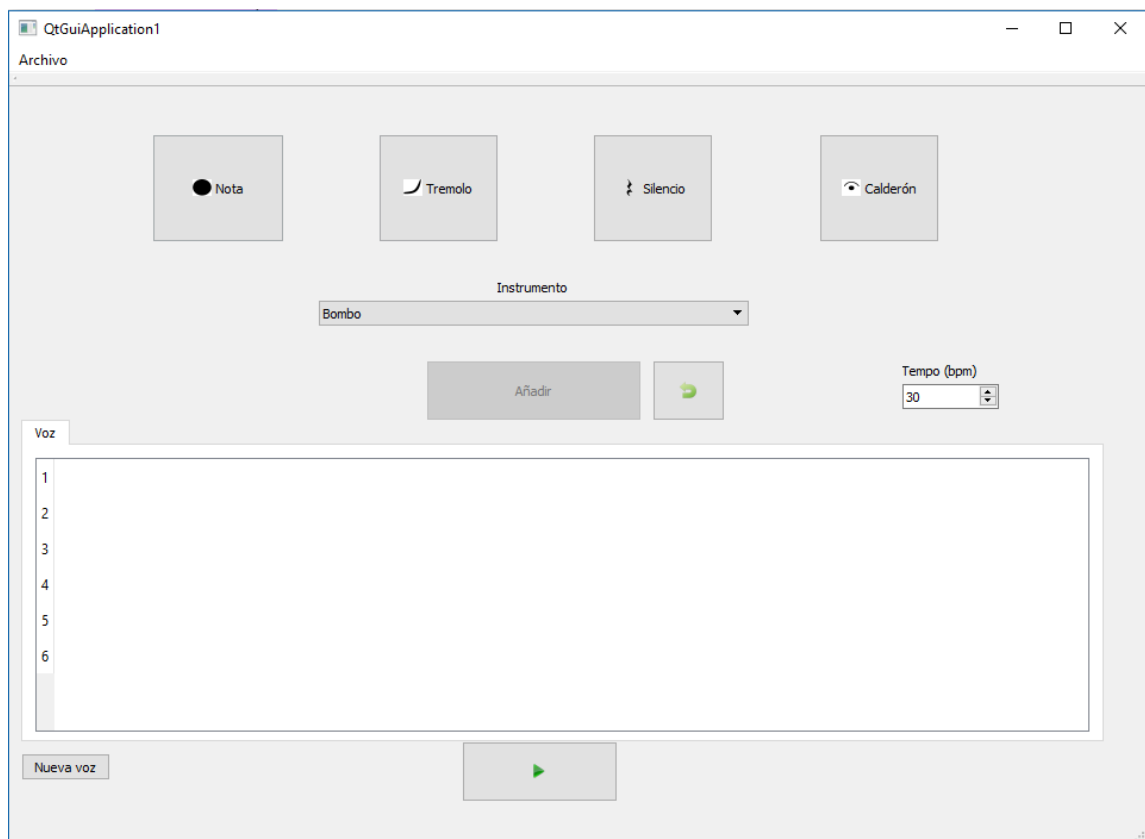
Ahora ya disponemos de todo lo necesario para realizar una prueba y comprobar que todo funciona como deseamos.

En esta prueba vamos a buscar una canción famosa, y generaremos el ritmo de batería con nuestro programa. En este caso vamos a escoger una canción de AC/DC, concretamente “You shook me all night long”. Crearemos un archivo MIDI que posteriormente abriremos con Cubase y aplicaremos el instrumento VST Kontakt para aplicarle las distintas sonoridades a nuestro instrumento, de forma que podamos modificar los sonidos a nuestro gusto.

Comenzaremos con nuestro programa principal, creando el ritmo de batería que consistirá en:

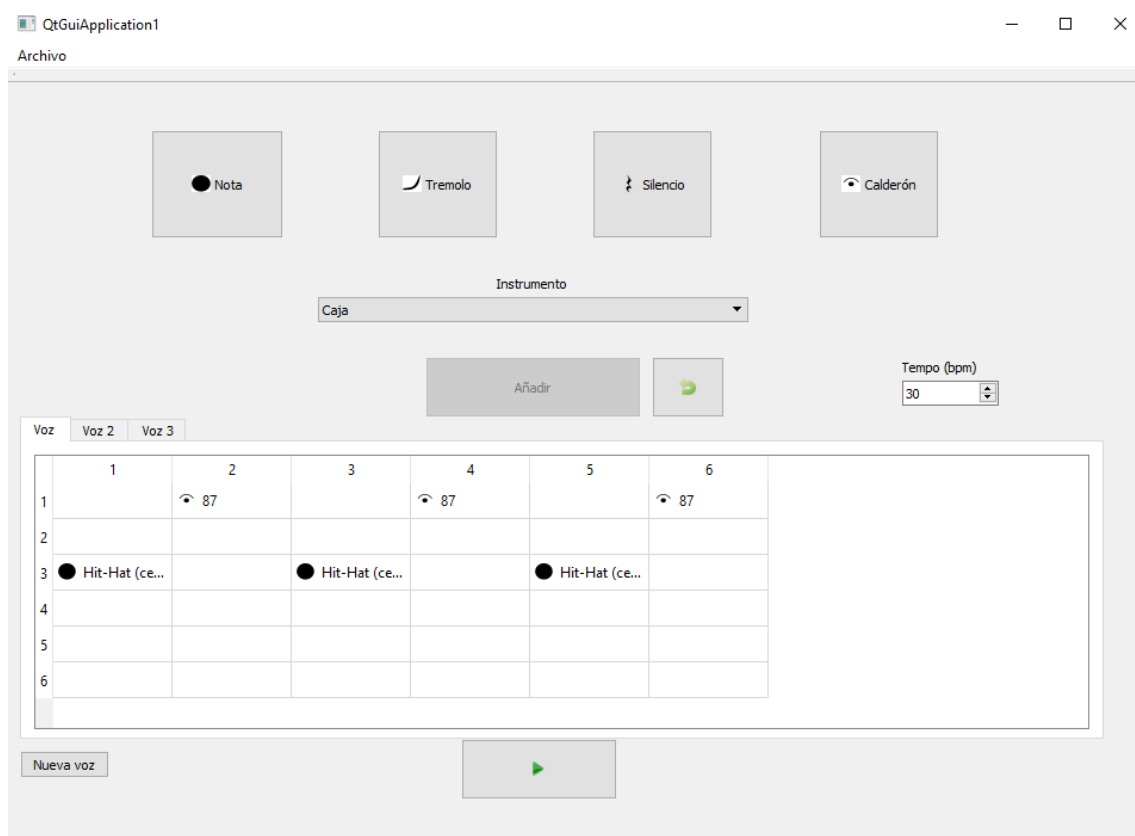
- Golpe de charles cerrado en cada compás
- Golpe de bombo en los tiempos 1 y 6
- Golpe de caja en los tiempos 3 y 7

Este compás se repite a lo largo de toda la canción con alguna que otra variación, incluyendo golpes de platos.



*Ilustración 59: Pantalla principal de la Aplicación*

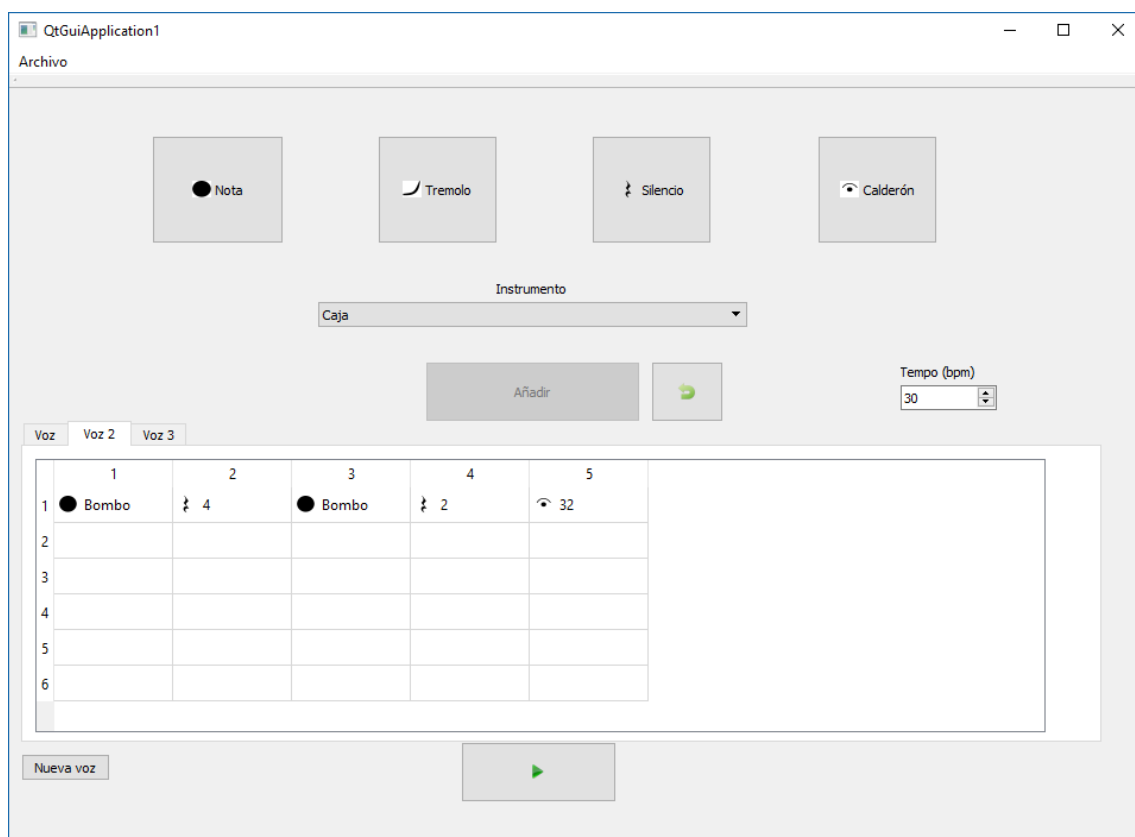
Vamos a comenzar por crear la primera parte de la partitura, que se va a corresponder con los distintos golpes de Hit-hat que iremos dando durante la partitura.



*Ilustración 60: Voz número1 de la prueba*

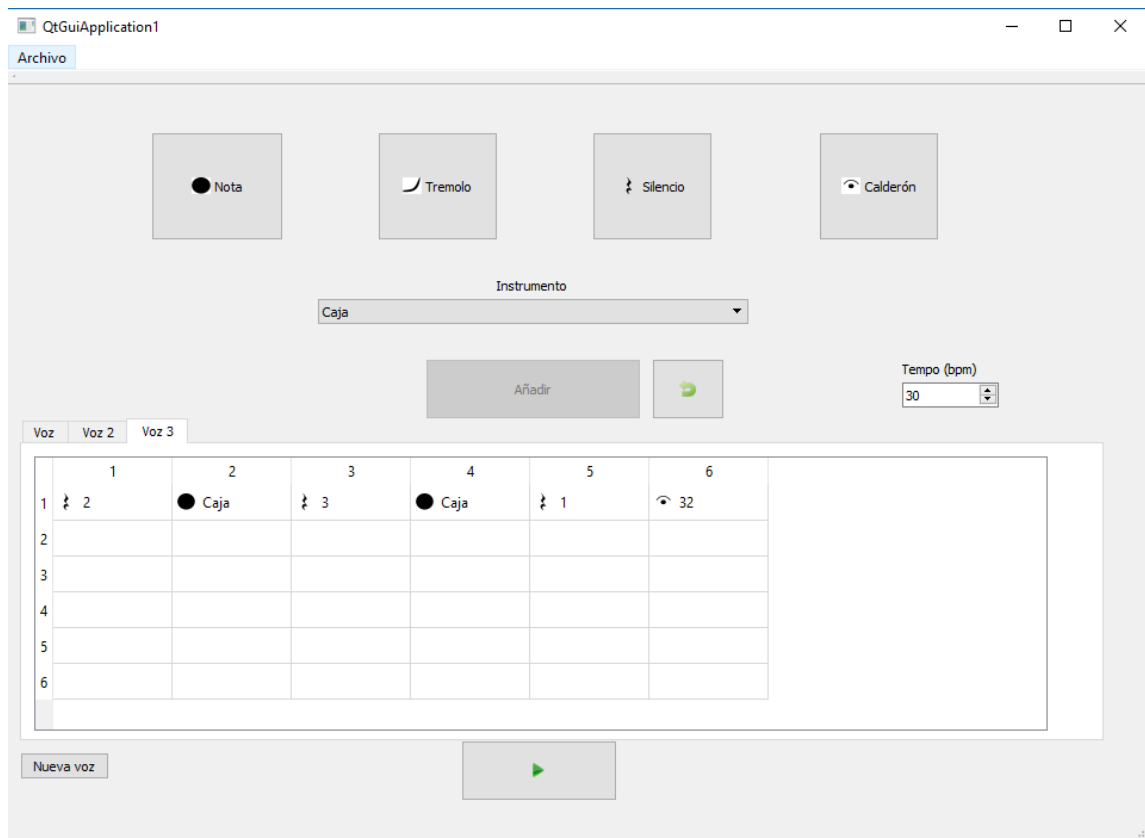
Esta voz de la partitura va a ser muy repetitiva, ya que solamente trabajaremos con los golpes de Hit-hat.

Ahora pasaremos con la siguiente voz, que se corresponderá con los diferentes golpes de bombo.



*Ilustración 61: Voz número2 de la prueba*

Con esto, veremos que el bombo se da en los tiempos número 1 y número 6, que se irán repitiendo a lo largo de la partitura. Seguidamente, haremos la voz correspondiente con la caja.



*Ilustración 62: Voz número3 de la prueba*

La voz de la caja, irá dando golpes en los tiempos 3 y 7, repitiendo este patrón durante toda la canción.

Finalmente, tendremos que añadir una nueva voz que será la que de los golpes de plato que se dan cuando se va a entrar al estribillo de la canción, así como durante él.

Para ello, pues tendremos que crear compases de espera y después que entre con los golpes correspondientes.



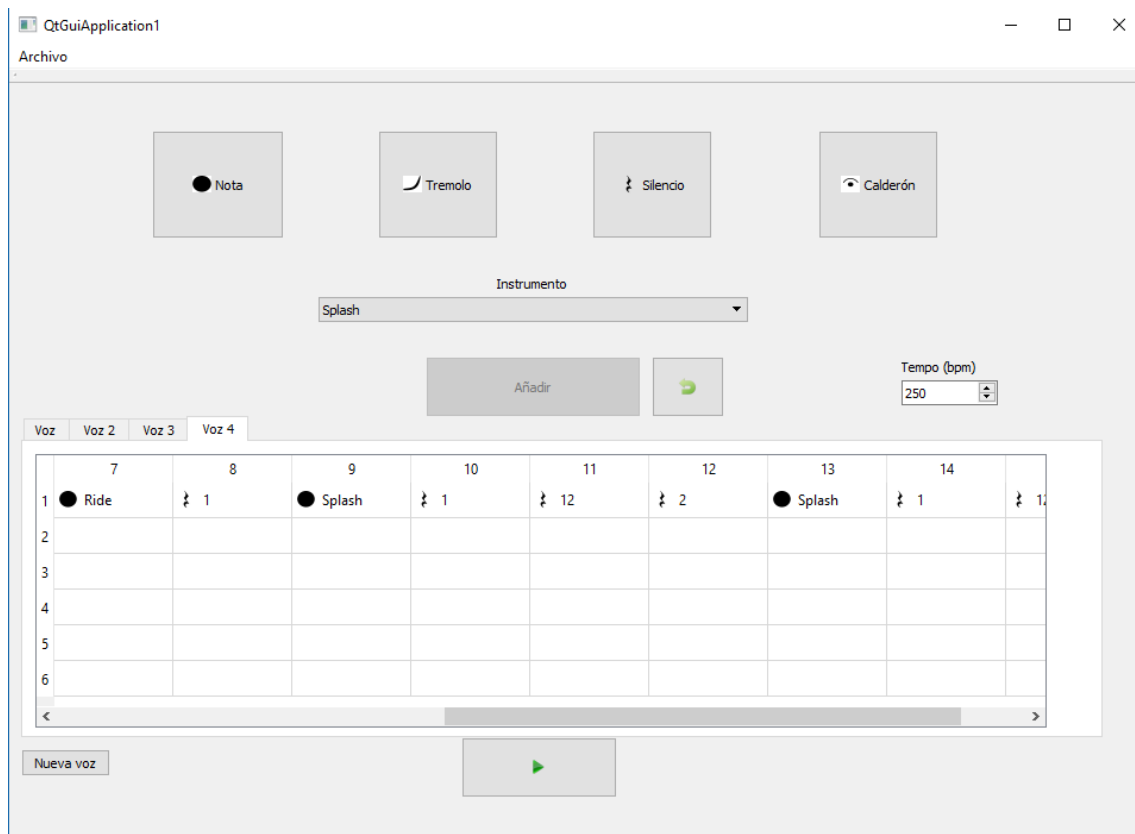


Ilustración 63: Voz número4 de la prueba

Con esto ya podemos decir que tenemos la partitura de una parte de esta canción, desde el inicio hasta el estribillo, ya que posteriormente, la canción continúa exactamente igual.

Creada la partitura procederemos a guardarla en la ubicación que indiquemos con el nombre que deseemos.

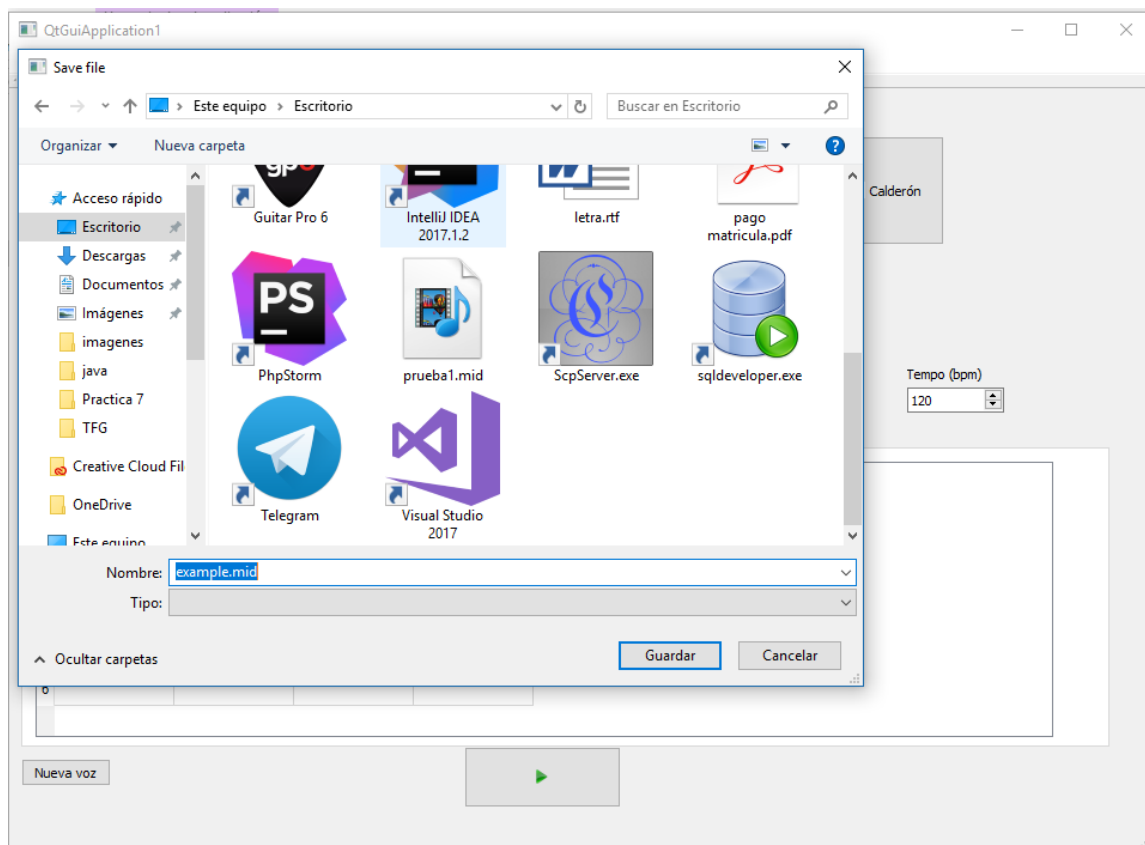


Ilustración 64: Explorador de archivos para almacenar el archivo

Con nuestro archivo almacenado, ahora pasaremos a abrir Cubase, importarlo y aplicarle el instrumento VST que queremos.

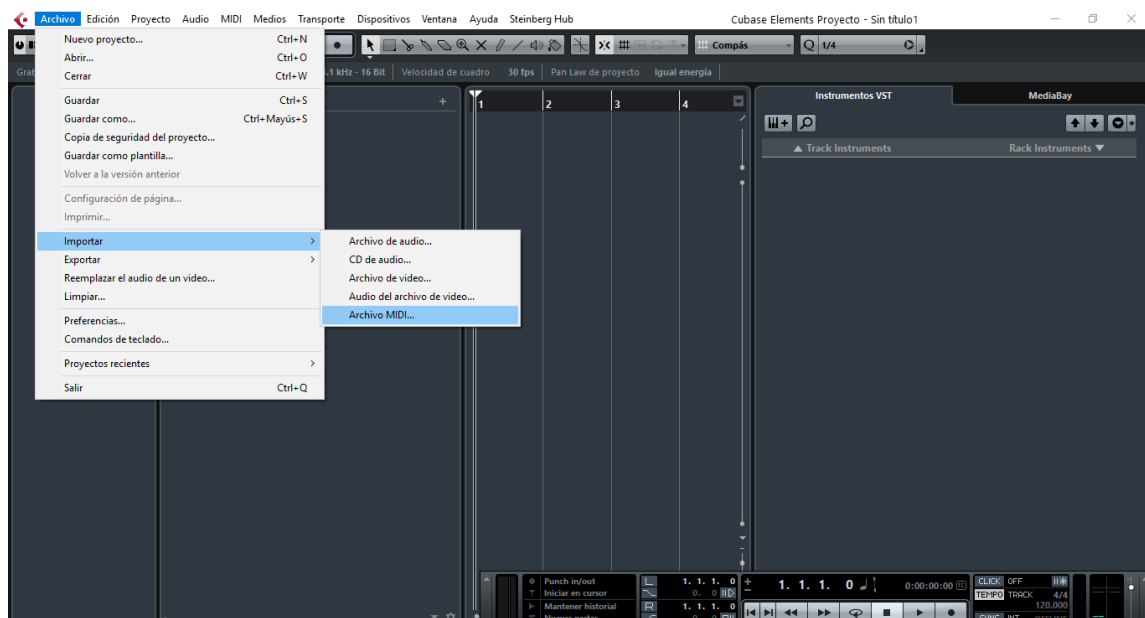
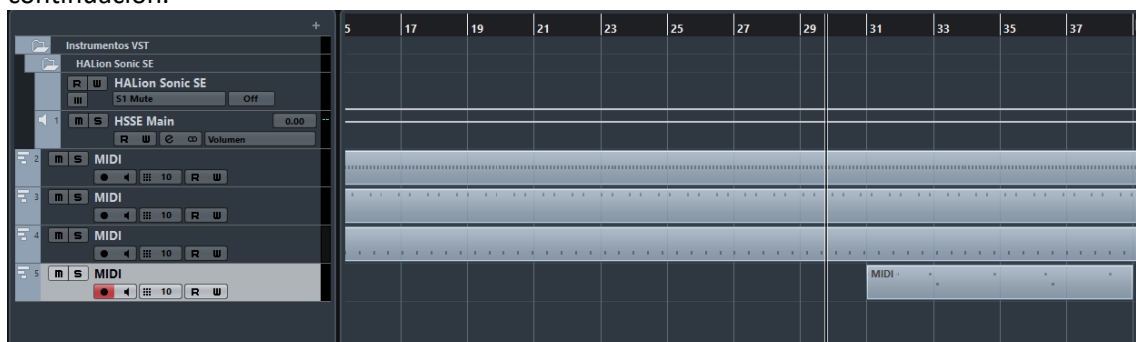


Ilustración 65: Importar MIDI en Cubase

Comenzaremos por importar nuestro archivo MIDI creado desde el Menú que mostramos en la anterior imagen.

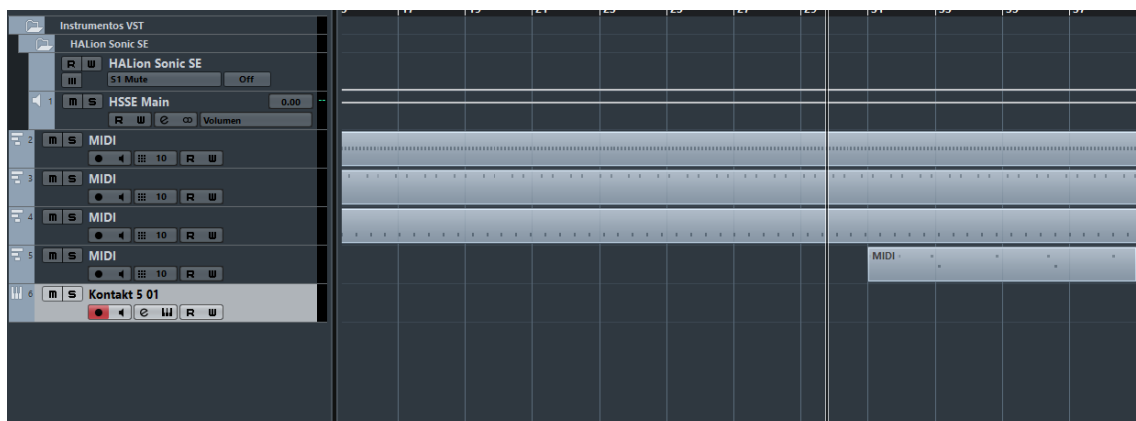
Una vez importado, el archivo MIDI se nos mostrará dentro de Cubase como vemos a continuación.



*Ilustración 66: Archivo MIDI abierto en Cubase*

Como podemos observar, vemos las tres voces que creamos anteriormente y si pasamos a reproducirlas, podremos escuchar un sonido que se corresponderá con la partitura que teníamos en mente.

Ahora pasaremos a aplicarle el instrumento VST para darle un sonido más “real”, y para ello tendremos que agregar una pista de “Instrumento VST” en el que nosotros escogeremos Kontakt5.



*Ilustración 67: Pista de instrumento*

Al agregar la pista de instrumento, tendremos que indicarle a nuestro archivo MIDI que queremos que coja el sonido de este instrumento, ya que Cubase por defecto nos aplica un Instrumento diferente y gracias a eso hemos podido escuchar la partitura anteriormente sin necesidad de agregar nosotros ningún instrumento.

Con Kontakt 5 agregado, podremos pasar a ver los diferentes sonidos de instrumentos que dispone, gracias a la multitud de librerías que dispone.



Ilustración 68: Kontakt 5 Instrument

Nosotros en este caso hemos seleccionado los sonidos de “Pop Kit” que son los sonidos de percusión de una batería tradicional y bastante fieles a la realidad, por otra parte, tenemos también otros sonidos de percusión como “Funk Kit” o “Street Knowledge Kit” que tienen otros sonidos más novedosos.

Una vez seleccionado el sonido que queremos podremos pasar a reproducir nuestra partitura y ver el nuevo sonido asignado. Además de esto, con Kontakt podemos modificar estos sonidos con todas las opciones de configuración que vemos, como es el volumen, la ganancia, el panning, delay, etc. Por lo que podemos ver que se pueden conseguir sonidos de todas formas y ajustarlos a las necesidades que nosotros queremos.

Ahora ya disponemos de nuestra partitura MIDI creada, vamos a probar por lo tanto a descargarnos la canción sobre la que hemos realizado la partitura para probar esta prueba. Para realizar la prueba, vamos a buscar una canción con la batería eliminada, de forma que únicamente suenen guitarras, bajo y voz y así poner nuestro MIDI sobre la canción.

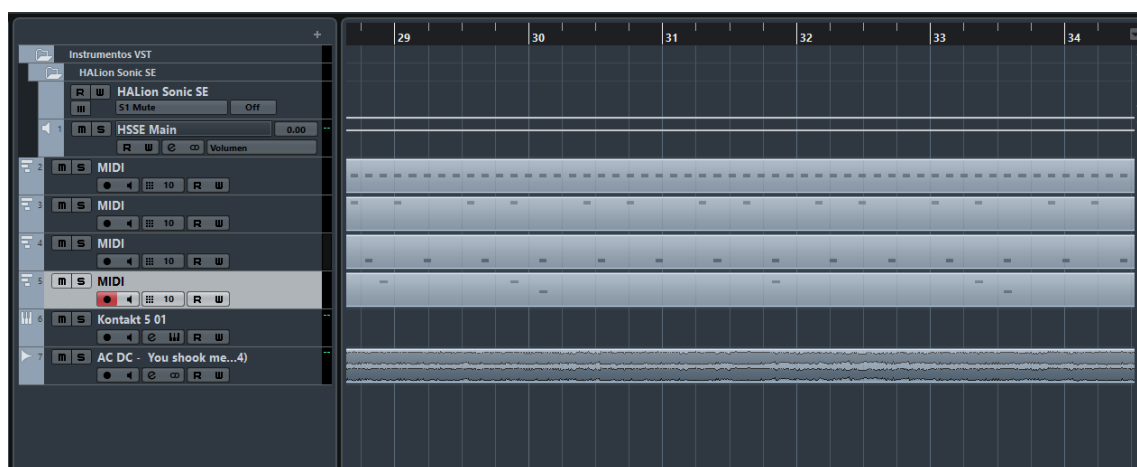


Ilustración 69: Reproducción de una canción con nuestra batería sobre ella

Para poder reproducirlas, debemos alinear los tiempos, es decir, antes de crear nuestra partitura debemos medir con un metrónomo el tempo de la canción original para posteriormente poder

alinearla con la original. Hecho esto, pasamos a reproducirla y tenemos nuestro resultado final.

## 9. Trabajo Futuro

Debido a las limitaciones de tiempo, han existido partes del proyecto que se han visto recortadas y que no han podido ser implementadas, como es el caso del desarrollo de un servicio web, de forma que podríamos trabajar con nuestro proyecto desde cualquier navegador.

Aunque no se ha desarrollado este servicio web, se ha creado el siguiente diagrama de contexto que nos explica cómo funcionaría el servicio web.

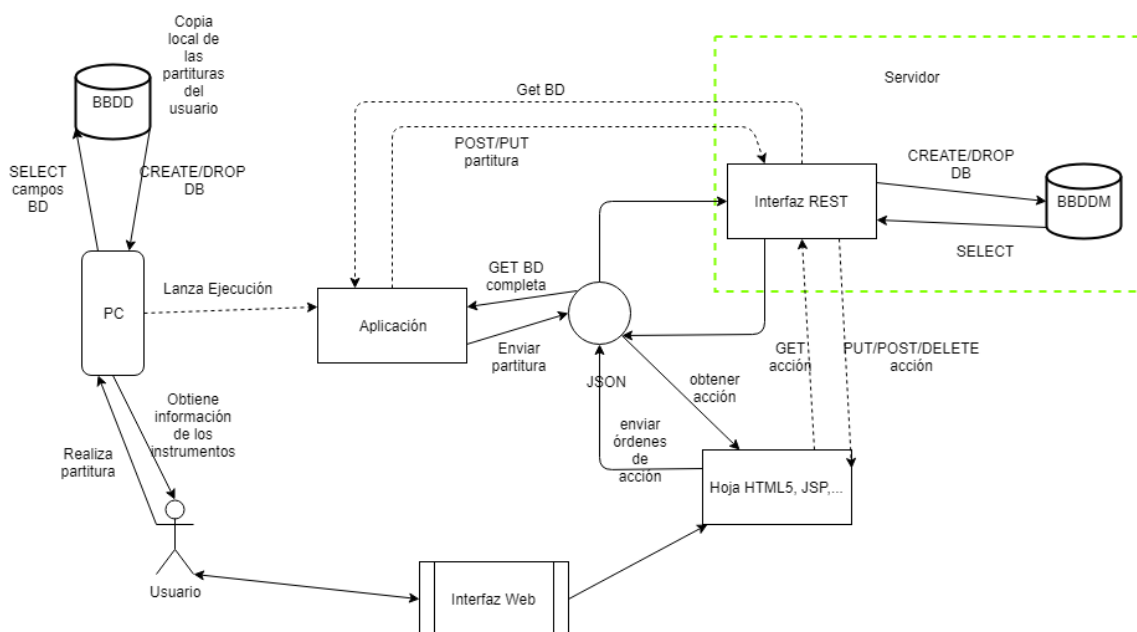


Ilustración 70: Diagrama de contexto

Como podemos ver, podríamos utilizar una base de datos, de forma que el usuario pudiera registrarse en nuestra plataforma y almacenar así todas las partituras que creara y recuperarlas siempre que las necesitara.

De esta forma, podríamos tener tanto una plataforma nativa como un servicio web que nos permitiría crear nuestra partitura de batería, almacenarla para continuar con ella posteriormente si no la terminamos, reproducirla, descargar el archivo MIDI para trabajar con él, etc.

Este proyecto una vez realizado en su completitud podrá ser utilizado para músicos no profesionales permitiéndoles de esta forma poder medir de forma precisa y sencilla los tiempos musicales y sobre todo por músicos profesionales para ampliar al máximo el campo de la duración y la intensidad.

La notación existente actualmente no permite precisar las duraciones de las notas a niveles tan precisos: no hay forma de indicar el 16% de una negra, seguido del 23% de ese 16 anterior (con este sistema de notación es posible), y respecto a la intensidad, el sistema tradicional de notación es subjetivo y limitado, se indica "piano" (suave) o "forte" (fuerte) con unos pocos niveles de intensidad, generalmente, desde pppp hasta ffff con un "mezzoforte" (mediofuerte)

en medio, o sea, 9 indicadores subjetivos de intensidad. Con este sistema de notación la intensidad se mide infinitamente en decibelios.

Como se puede observar, hay una gran cantidad de trabajo que podría ser realizada con más tiempo y que proporcionaría a la aplicación una funcionalidad muchísimo mayor.

## 10. Conclusiones

El objetivo fundamental de este trabajo es llevar música clásica contemporánea al ámbito sonoro, para lo cual se hace uso de una notación específica centrada especialmente en la duración y la intensidad. El fin de hacer sonar esta música es, utilizando palabras del Dr. Antonio Martín Moreno, hacer al ser humano más conocedor de sí mismo, y añadido, de aquello que le rodea.

Con este proyecto, podremos crear partituras con una notación específica para batería, como la propuesta por Julio Omella. A diferencia de GuitarPro no necesitaremos conocimientos de lenguaje musical para poder trabajar con nuestra aplicación, tampoco necesitaremos conocer cómo generar partituras MIDI como ocurre en GarageBand, ya que nuestra aplicación generará la partitura automáticamente y a diferencia de EZDrummer, podremos generar una partitura con una notación, cosa que no disponemos en EZDrummer ya que únicamente indicamos los golpes a dar en cada instrumento.

A nivel de cumplimentación de objetivos, no se ha conseguido realizar la integración del instrumento VST dentro de nuestra aplicación, aunque se ha conseguido aplicar utilizando una tercera aplicación, y por falta de tiempo, ha sido imposible crear un servicio web de este proyecto.

Pero se ha conseguido hacer una primera versión de una aplicación que puede ser bastante útil para todos aquellos músicos que toquen la percusión, ya que se puede generar una partitura que puede ser interpretada fácilmente y además generar un archivo reproducible que nos permite poder escuchar el trabajo creado en la partitura.

Este trabajo Fin de Grado me llamó mucho la atención cuando lo vi por primera vez, debido a que soy un verdadero apasionado de la música y el poder unir dos de mis verdaderas pasiones como es la música y la informática me pareció una oportunidad bastante buena.

El desarrollo del proyecto ha sido un trabajo en el que he invertido mucho más tiempo en documentarse del que esperaba. Esto se debe a que no existe mucha documentación del estándar MIDI y cómo trabajar con él, por lo que en esta parte tuve bastantes tropiezos hasta entender cómo funcionaba.

Gracias al conocimiento musical del que dispongo, la traducción de la notación de Julio no ha sido un trabajo difícil ya que podía entender a la perfección qué es lo que se pretendía en cada acción de las que se indicaban.

Por último, encontré una enorme barrera con Cubase y Kontakt5, debido a que quería integrar todo en una misma aplicación, pero el uso de estos softwares me supuso bastantes problemas y, finalmente, no conseguí integrar el uso de Kontakt 5 dentro de la parte cliente de mi aplicación.

Quizás con más tiempo para este trabajo, se puede sacar un plugin que podamos integrar en Cubase que haga uso de Kontakt 5 y de esta forma poder utilizar más muchas especificaciones de la notación de Julio Omella que en el estado actual de este trabajo no hemos podido transformar al formato MIDI, como es el timbre del instrumento, el golpeo de la baqueta (staccato, staccatissimo, etc).

Aparte de esto, estoy contento con el trabajo realizado, ya que he conseguido una aplicación funcional con la que podemos generar archivos MIDI a través de una interfaz gráfica bastante sencilla que es fácil de utilizar para cualquier persona, sin depender del conocimiento musical que esta persona tenga.

De nuevo y para finalizar, dar de nuevo las gracias a Julio Omella por dejarnos su notación y darnos permiso para utilizarla en este trabajo, siendo Julio su único autor.





# Índice de tablas

Tabla 1: Contenido Nodo Cabecera MIDI .....	34
Tabla 2: Contenido de la Cabecera Nodo Pista MIDI .....	35
Tabla 3: Contenido Nodo Evento MIDI .....	35
Tabla 4: Tipos de Evento MIDI .....	35
Tabla 5: Contenido del Nodo Meta-evento MIDI .....	36
Tabla 6: Tipos de metaevento .....	36
Tabla 7: Valores MIDI de Instrumentos de percusión.....	37

# Índice de imágenes

Ilustración 1: Captura de pantalla de GarageBand .....	20
Ilustración 2: Captura de pantalla Guitar Pro.....	21
Ilustración 3: Captura de pantalla EZDrummer .....	22
Ilustración 4: Tabla organización de tareas .....	23
Ilustración 5: Diagrama de Gantt.....	24
Ilustración 6: Medidas mínimas.....	24
Ilustración 7: Ejemplo de diferentes tempos .....	25
Ilustración 8: Ejemplo de staccato y staccatissimo.....	25
Ilustración 9: Ejemplo de combinación de staccato, staccatissimo y tenuto.....	25
Ilustración 10: Ejemplo de silencios.....	26
Ilustración 11: Ejemplo calderón .....	26
Ilustración 12: Ejemplo uso sordina.....	26
Ilustración 13: Ejemplo de trémolos .....	27
Ilustración 14: Símbolos de trémolo .....	27
Ilustración 15: Ejemplo símbolo trémolo .....	27
Ilustración 16: Ejemplo tutti .....	27
Ilustración 17: símbolos trémolo de tiempo .....	28
Ilustración 18: Ejemplo símbolo de trémolo en el tiempo.....	28
Ilustración 19: Símbolos trémolo de intensidad .....	28
Ilustración 20: Ejemplo símbolo trémolo de intensidad .....	29
Ilustración 21: Claves trémolo .....	29
Ilustración 22: Ejemplo claves de trémolo .....	29
Ilustración 23: Resumen símbolos de trémolo .....	29
Ilustración 24: Ejemplos de trémolos .....	30
Ilustración 25: Puntos de intensidad .....	30
Ilustración 26: Zoom de puntos de intensidad .....	30
Ilustración 27: Claves de intensidad .....	30
Ilustración 28: Símbolos de voces.....	31
Ilustración 29: Colores para voces .....	31
Ilustración 30: Desdoblamiento de una voz .....	31
Ilustración 31: Vientres y nodos .....	32
Ilustración 32: Ejemplo de uso de nodos y vientos .....	32
Ilustración 33: Ejemplo de uso de baquetas .....	32
Ilustración 34: Información inicial de partitura .....	33
Ilustración 35: Estructura de batería .....	37

Ilustración 36: Atributos principales de las clases Nota y Baqueta .....	39
Ilustración 37: Atributos principales de la clase Trémolo .....	40
Ilustración 38: Planteamiento del sistema .....	40
Ilustración 39: Diagrama de clases notación .....	41
Ilustración 40: Diagrama de clases MIDI .....	42
Ilustración 41: Diagrama de clases GUI .....	43
Ilustración 42: Diagrama de secuencia .....	44
Ilustración 43: Ejemplo de transformación de Tremolo NORMAL .....	48
Ilustración 44: Ejemplo de código de transformación de trémolo GRADUAL-TIEMPO .....	49
Ilustración 45: Vista de trémolo GRADUAL-TIEMPO en Cubase.....	49
Ilustración 46: Ejemplo de código de Trémolo GRADUAL-INTENSIDAD .....	50
Ilustración 47: Vista de la variación de intensidad de un Trémolo GRADUAL-INTENSIDAD en Cubase.....	50
Ilustración 48: Parte del constructor Partitura(String p) .....	51
Ilustración 49: Texto partitura .....	51
Ilustración 50: Obtener valores introducidos por el usuario .....	53
Ilustración 51: Ejemplo de separación de fiñas en el hexagrama .....	53
Ilustración 52: Representación de nota .....	53
Ilustración 53: Cambiar entradas de usuario según el tipo de trémolo.....	56
Ilustración 54: Ubicacion DLLs Kontakt 5 .....	59
Ilustración 55: Pantalla principal Kontakt 5.....	59
Ilustración 56: Ubicación VSTPlugins de Cubase .....	60
Ilustración 57: Kontakt 5 como VST Instrument en Cubase .....	60
Ilustración 58: Pantalla principal de la Aplicación .....	61
Ilustración 59: Voz número1 de la prueba .....	62
Ilustración 60: Voz número2 de la prueba .....	63
Ilustración 61: Voz número3 de la prueba .....	64
Ilustración 62: Voz número4 de la prueba .....	65
Ilustración 63: Explorador de archivos para almacenar el archivo.....	66
Ilustración 64: Importar MIDI en Cubase .....	66
Ilustración 65: Archivo MIDI abierto en Cubase .....	67
Ilustración 66: Pista de instrumento .....	67
Ilustración 67: Kontakt 5 Instrument .....	68
Ilustración 68: Reproducción de una canción con nuestra batería sobre ella .....	68
Ilustración 69: Diagrama de contexto .....	69

# Bibliografía

<https://www.steinberg.net/en/home.html>

[https://es.wikipedia.org/wiki/Virtual\\_Studio\\_Technology](https://es.wikipedia.org/wiki/Virtual_Studio_Technology)

<https://www.steinberg.net/en/company/developers.html>

<https://www.native-instruments.com/es/products/komplete/samplers/kontakt-5/>

<https://support.native-instruments.com/hc/es/articles/209569989-Administraci%C3%B3n-de-plug-ins-en-Cubase-8>

[https://es.wikipedia.org/wiki/Qt\\_\(biblioteca\)](https://es.wikipedia.org/wiki/Qt_(biblioteca))

<https://es.wikipedia.org/wiki/GarageBand>

[https://es.wikipedia.org/wiki/Guitar\\_Pro](https://es.wikipedia.org/wiki/Guitar_Pro)

<https://en.wikipedia.org/wiki/EZdrummer>

